



**Universität
Zürich^{UZH}**

TrOCR meets CharBERT

**Masterarbeit der Wirtschaftswissenschaftliche
Fakultät der Universität Zürich**

eingereicht von

Yung-Hsin Chen

Matrikelnummer 20-744-322

Institut für Informatik der Universität Zürich

Prof. Dr. Martin Volk

Institut für Computerlinguistik der Universität Zürich

Supervisor: Dr. Simon Clematide, Dr. Phillip Ströbel

Abgabedatum: 20.05.2024

Abstract

This is the place to put the English version of the abstract.

Zusammenfassung

Und hier sollte die Zusammenfassung auf Deutsch erscheinen.

Acknowledgement

I want to thank X, Y and Z for their precious help. And many thanks to whoever for proofreading the present text.

Contents

Abstract	i
Acknowledgement	ii
Contents	iii
List of Figures	vi
List of Tables	vi
List of Acronyms	vii
1 Introduction	1
1.1 Motivation	1
1.2 Research Questions	3
2 Related Work	4
2.1 Optical Character Recognition (OCR)	4
2.2 Post-OCR correction	6
2.3 TrOCR	6
2.4 CharBERT	7
2.5 Candidate Fusion	9
2.6 Glyph Embedding	10
3 Methodology	12
3.1 Design Concept	12
3.2 Data Collection and Processing	12
3.2.1 Data for OCR	13
3.2.2 Data for Training CharBERT _{SMALL}	16
3.3 Composite Model Architecture	16
3.3.1 Notations	17
3.3.2 Recogniser - TrOCR	17
3.3.3 Corrector - CharBERT	19
3.3.4 Composite Model	20

3.4	Glyph Incorporation	27
3.4.1	Get \mathcal{P}_{ij}	27
3.4.2	Training CharBERT $_{\mathcal{P}_{ij}}$	28
4	Experiment	33
4.1	Baseline Model	33
4.2	Composite Model Training and Evaluation Criteria	34
4.2.1	Training Details	34
4.2.2	Training Challenges and Solutions	35
4.2.3	Evaluation	37
4.3	CharBERT _{SMALL} Training and Evaluation Criteria	37
4.4	Composite Model Analysis	38
4.4.1	Component Analysis and Benchmarking	38
4.4.2	Validating Model Generalization and Adapability	39
5	Results	42
5.1	Metric	42
5.1.1	Character Error Rate (CER)	43
5.1.2	Word Error Rate (CER)	43
5.2	Results	43
5.2.1	Baseline Models	44
5.2.2	Freezing Specific Layers	44
5.2.3	Integrating Dropout Mechanisms	44
5.2.4	Comparing Combined Tensor Modules	45
5.2.5	Comparing Loss Functions	45
5.2.6	Benchmarking Against GPT-4	45
5.3	Graphics	45
6	Discussion	47
7	Conclusion	48
7.1	Future Work	48
	Glossary	49
	References	50
	Curriculum vitae	54
A	Tables	55
B	List of something	56

List of Figures

1	Digitization Process of Historical Documents	5
2	Stages of the Optical Character Recognition (OCR) Process	5
3	Experimental Results of CharBERT and Strong Pre-trained Models	9
4	GW Line Image	14
5	JH Line Image	14
6	Image Processing	16
7	Workflow in the Composite Model	17
8	TrOCR Input Processing	19
9	Adapted TrOCR	21
10	Adapted CharBERT	22
11	Tensor Transform	29
12	Tensor Combine Module 1	30
13	Tensor Combine Module 2	31
14	Tensor Combine Module 3 and 4	32
15	Rosetta	46

List of Tables

1 Some large table 55

List of Acronyms

OCR	Optical Character Recognition
LM	Language Model
LLM	Large Language Model
NLM	Noisy Language Model
MLM	masked language modeling
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
SOTA	State of the Art
BPE	Byte-Pair Encoding
ViT	Vision Transformer
TrOCR	Transformer-based Optical Character Recognition
GRU	Gated Recurrent Units
FFNN	Feedforward Neuron Network
Bi-GRU	Bidirectional GRU
GPU	Graphics Processing Unit
CER	Character Error Rate
WER	Word Error Rate
NER	named entity recognition
IR	information retrieval
LSTM	Long Short-Term Memory network
SOTA	State of the Art
DeiT	Data-Efficient Image Transformer
BEiT	Bidirectional Encoder Representation from Image Transformers
BERT	Bidirectional Encoder Representations from Transformers
OOV	Out of Vocabulary
GRU	Gated Recurrent Unit
NMT	Neural Machine Translation
JH	Joseph Hooker
GW	George Washington

HTR Handwritten Text Recognition

1 Introduction

1.1 Motivation

Optical Character Recognition (OCR) has become a key tool for digitizing printed documents [Singh et al., 2012]. While OCR tasks for modern printed materials are typically straightforward, digitizing ancient texts or handwritten documents introduces complex challenges [Neudecker et al., 2019]. Inadequate OCR can significantly affect downstream tasks such as text classification, named entity recognition (NER), information retrieval (IR), etc, leading to poor data utility.

Kettunen et al. [2022] explores the significant impact of OCR quality on the effectiveness of information retrieval tasks, particularly focusing on historical newspaper collections. Through an experimental setup involving 32 users who evaluated search results from historical newspapers, it was found that improved OCR quality significantly enhances the perceived usefulness of historical newspaper articles. The main findings includes 1) Higher OCR quality led to more favorable relevance assessments, with the average evaluation score for improved OCR results being 7.94% higher than those obtained with older, lower-quality OCR; 2) Users were more likely to find documents relevant when the underlying text was optically recognized with higher accuracy; 3) As OCR quality improves, it supports more reliable and effective searches, which is crucial for historians, researchers, and the general public engaging with historical databases; 4) Investments in better OCR methods could substantially increase the accessibility and utility of historical text archives.

Nguyen et al. [2021] also highlights how OCR errors can significantly affect downstream applications such as IR and various natural language processing (NLP) tasks. OCR inaccuracies can disrupt a wide range of NLP applications including NER, part-of-speech (POS) tagging, text summarization, and sentiment analysis. For instance, as the word error rate (WER) increases, the performance of NER tools drops significantly, illustrating a rapid decline from high accuracy to substantially lower levels as errors increase.

[Ehrmann et al. \[2023\]](#) outlines several challenges faced by Named Entity Recognition (NER) systems when applied to historical handwritten documents including 1) Noisy Input; 2) Dynamics of Language; 3) Lack of Resources.

Noisy Input Texts derived from historical documents often suffer from quality issues due to the condition of the source material and the process of digitization. OCR and Handwritten Text Recognition (HTR) systems may introduce errors such as misrecognized characters and tokenization problems, severely impacting the accuracy of downstream tasks. The challenge is compounded by the diverse nature of noise in historical texts, ranging from ink bleeds and paper deterioration to varying typographic conventions over time.

Dynamics of Language The language used in historical documents can significantly differ from modern language due to evolutionary changes in spelling, grammar, and syntax. NER systems must cope with historical spelling variations, outdated naming conventions, and the historical context of terms, which may vary significantly from their modern counterparts. Such linguistic dynamics pose a substantial challenge to maintaining high accuracy in entity recognition and classification.

Lack of Resources A significant scarcity of annotated corpora, language models, and other NLP resources specifically designed for historical document processing substantially hinders the development and training of models tailored to these texts. Moreover, the resources that do exist often lack standardization, adding further complexity to the development, training, and evaluation of effective models.

Consequently, post-OCR correction has emerged as a crucial approach to overcome these limitations and enhance the accuracy of digitized data. To mitigate these effects, [Nguyen et al. \[2021\]](#) explore advanced post-OCR correction techniques in their survey, which includes the integration of sophisticated language models and the application of both statistical and neural machine learning models. These methods, which range from manual to semi-automatic and fully automatic approaches, significantly enhance the quality of text. This improvement in text quality is crucial for the reliability of subsequent information retrieval (IR) and natural language processing (NLP) tasks, underscoring the ongoing need for enhancements in post-processing technologies to effectively manage OCR errors.

Several post-OCR correction methods have been applied and shown significant improvements. Most of these methods function sequentially, not in an end-to-end

manner. However, [Kang et al. \[2021\]](#) suggest that allowing backpropagation to influence both the recognizer and language model concurrently can yield better results than training them separately.

Therefore, this study aims to evaluate the effectiveness of integrating a recognizer with a language model to not only enhance performance but also enable models trained on modern texts to adjust to ancient English texts.

1.2 Research Questions

In this study, we aim to investigate and address several key research questions that are pivotal to advancing the field of Optical Character Recognition (OCR) and enhancing the integration of OCR with language models. These questions are designed to explore the depths and potential of emerging technologies within the context of text digitization and correction. Through this inquiry, we seek to not only validate new methodologies but also to push the boundaries of what these technologies can achieve in practical applications. The research questions will guide the experimental design and analysis, ensuring that the outcomes are robust, insightful, and contribute significantly to the existing body of knowledge.

1. **Domain Adaptation Through Model Fusion:** In the Candidate Fusion paper, they claimed that fusing the recogniser and LM can make the LM adjust to the domain-specific data. Can fusing TrOCR and CharBERT achieve the same conclusion? In other words, can CharBERT adjust to historical texts even it was trained on modern texts?
2. **Character-Level Information Enhancement:** In the CharBERT paper, they claimed that, by using the character level information in addition to the subword level information, the problems of incomplete modelling and fragile representation can be solved. However, the results shown in the paper did not show significant performance improvement over RoBERTa (a strong baseline LM model). Is this statement valid? Can TrOCR combined with CharBERT achieve the claim?
3. **Influence of Language Model on Decoder Output:** Does TrOCR decoder change its output with the presence of the language model?
4. **Impact of Training on Common Errors:** After integrating common errors made by TrOCR into the training process of CharBERT, do the results show improvements?

2 Related Work

This chapter explores the dynamic field of Optical Character Recognition (OCR), tracing the evolution of this pivotal technology and the development of sophisticated post-OCR correction techniques that enhance the utility of OCR outputs. A detailed exploration of TrOCR, a state-of-the-art (SOTA) OCR model, alongside CharBERT, a large language model, forms a significant portion of this chapter, with an emphasis on their architectures and training methodologies. These two models are central to this study due to their innovative integration potential. Additionally, the concept of candidate fusion is discussed, which leverages the strengths of both OCR and language models to improve text recognition accuracy. The chapter also examines the advantages of incorporating glyph analysis into model training, further refining the recognition process. Overall, this chapter reviews the current advancements in OCR technology and sets the stage for the introduction of a novel composite model designed to synergistically combine these technologies, tackling the challenges associated with precise text recognition and correction.

2.1 Optical Character Recognition (OCR)

According to [Philips and Tabrizi \[2020\]](#), the digitization of historical documents involves several critical phases, each leveraging advanced technologies and methodologies to transform ancient texts into accessible digital formats. The process begins with image acquisition, where physical documents are converted into digital images through scanning or photography. Following this, preprocessing steps such as binarization, using methods like Otsu’s algorithm [[Liu and Yu, 2009](#)] to simplify images into binary formats, and layout analysis to determine structural elements like text blocks, are essential for preparing images for text recognition. Text recognition is then performed using technologies like OCR and Handwritten Text Recognition (HTR), with modern approaches often employing Convolutional Neural Networks (CNNs) [[Zhang et al., 2017](#)] and Long Short-Term Memory networks (LSTMs) [[Breuel et al., 2013](#)] to enhance accuracy. This comprehensive process is depicted in [Figure 1](#).

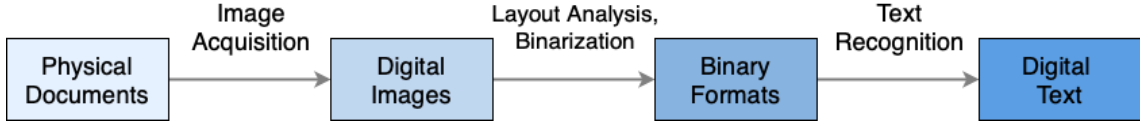


Figure 1: Digitization Process of Historical Documents

The paper by [Karthick et al. \[2019\]](#) elaborates on the OCR process, detailing the critical stages involved and exploring recent advancements in the field: 1) preprocessing; 2) feature extraction; 3) recognition. Initially, during preprocessing, the image quality is enhanced by removing noise, adjusting contrast, and segmenting the image to prepare it for analysis. This stage sets the foundation for feature extraction, where unique characteristics of the text are identified, which are crucial for accurate recognition. Following this, the recognition stage classifies these features into textual data. This text is then refined during the post-processing stage, where errors are corrected, and text clarity is improved. Before feature extraction, additional steps such as further segmentation and morphological processing are applied to enhance the text structure in the images. The process is illustrated in [Figure 2](#).

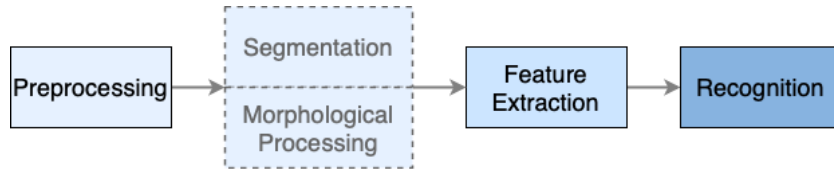


Figure 2: Stages of the Optical Character Recognition (OCR) Process

While traditional models typically handle the processes of preprocessing, feature extraction, and recognition in separate, discrete stages, there are modern OCR models that operate on an end-to-end basis [[Huang et al., 2021](#)][[Neudecker et al., 2019](#)][[Belay et al., 2020](#)]. These end-to-end models streamline the text recognition process by integrating all these stages into a single continuous workflow. This approach leverages advanced machine learning techniques, particularly deep learning, to directly process raw images of text into editable text outputs. This not only simplifies the OCR process by eliminating the need for manual feature engineering and separate processing stages but also enhances the system’s ability to handle varied and complex text presentations.

2.2 Post-OCR correction

After text recognition, post-processing corrects OCR errors through techniques such as spell checking and contextual adjustments based on language models. The final phase involves applying Natural Language Processing (NLP) for tasks like information retrieval, enabling further analysis and interpretation of the content. Tools like Tesseract [Smith, 2007] and Transkribus [Kahle et al., 2017] are pivotal in these processes, supporting both OCR and HTR tasks, while datasets such as the IAM Historical Document Database [Marti and Bunke, 1999] provide crucial benchmarks for training and evaluating technology performance. These stages collectively address the various challenges presented by the poor quality of many historical documents, the diversity of handwriting styles, and the computational demands of processing such data, underscoring the ongoing need for innovation in this field.

Mokhtar et al. [2018] points out that the structured and predictable nature of OCR errors makes them suitable for correction using machine learning models. They proposed two deep learning models: a word-based sequence-to-sequence model and a character-based model, allowing them to correct individual characters and handle words not seen during training. The findings show that while the word-based model struggles with unseen words, the character-based model, particularly when combined with normalization, performs well across different datasets, suggesting its practical applicability in improving OCR accuracy in real-world settings.

2.3 TrOCR

OCR tasks typically involve text detection and text recognition. TrOCR is a Transformer-based model which focuses on the text recognition part of the OCR task, converting images to texts. Therefore, the input should be sliced into line images, each with a single line of transcription on it. Unlike previous advancements in text recognition, TrOCR uses a pre-trained vision Transformer (ViT) [Dosovitskiy et al., 2020] for image feature extraction and a text Transformer [Vaswani et al., 2017] for sequence-to-sequence learning instead of CNN backbones [Wang et al., 2020a] and Connectionist Temporal Classification (CTC) [Graves et al., 2006]. It eliminates the usage of external large language models (LLMs) and can be extended for multilingual purposes by leveraging pre-trained language models (LMs) of different languages. In addition, unlike traditional OCR systems that rely on feature engineering and pre-/post-processing, TrOCR allows contextual learning, leading to SOTA results in challenging scenarios.

The encoder of TrOCR is initialized by pre-trained ViT-style models such as Data-Efficient Image Transformer (DeiT) [Touvron et al., 2021] or Bidirectional Encoder representation from Image Transformers (BEiT) [Bao et al., 2021], while the decoder initialization uses pre-trained Bidirectional Encoder Representations from Transformers (BERT)-style models such as RoBERTa [Liu et al., 2019] or MiniLM [Wang et al., 2020b]. Both of them are Transformer encoder structures. Therefore, the missing encoder-decoder attention in the TrOCR decoder is initialized randomly for training. Due to the fixed input length of Transformers, the image is first resized to 384×384 , and then split into 16×16 patches before inputting into the Transformer. Unlike CNNs, Transformers do not have spatial information of the input data. Thus, positional encodings are added to the patches to preserve the spatial structures of the input images.

Need full name?

TrOCR is pre-trained on synthetic data and SROIE [Huang et al., 2019], IAM datasets sequentially. TrOCR_{LARGE} (total parameters=558M), initialized by BEiT_{LARGE} and RoBERTa_{LARGE}, reached a character error rate (CER) [Klakov and Peters, 2002][Wang et al., 2003] of 2.89 by pre-training on the synthetic data and the IAM dataset.

So far, TrOCR is considered a SOTA model for OCR on both printed and handwritten tasks. Although its design and capabilities represent a significant advancement in the field of OCR, deficiencies still exist. In scenarios where the text layout is curved or vertical, TrOCR's performance is compromised. Note that these limitations are not unique to TrOCR but are rather common challenges for most OCR models. In this study, TrOCR will serve as the baseline. This foundational benchmark will be improved through integration with LMs to refine and correct the output results.

2.4 CharBERT

CharBERT [Ma et al., 2020] is an enhancement of BERT [Devlin et al., 2018], which aims to address problems in Byte-Pair Encoding (BPE) [Sennrich et al., 2015] used by pre-trained language models like BERT and RoBERTa. It has the same model structure and configuration as BERT and RoBERTa, depending on the initialization. During inference, it takes text as input and outputs a representative embedding of the text.

Pre-trained language models like BERT and RoBERTa have achieved outstanding results in NLP tasks. Both models use BPE to encode input data. BPE is capable of encoding almost all vocabularies, including out-of-vocabulary (OOV) words. It

breaks down OOV words into subwords until they are in its vocabulary dictionary. In addition, it allows efficiency in vocabulary space. For instance, BPE can represent different forms of a word (e.g., “run”, “running”, “runs”) with their common subwords, which results in a smaller set of total tokens. However, BPE has the problems of incomplete modeling and fragile representation.

Incomplete modeling refers to the inability of BPE to fully encapsulate the complete aspects of a word. BPE splits words into subword units. While these small pieces are helpful for understanding parts of the word, they may not entirely convey the meaning or nuances of the whole word. For instance, the word “understand”, which will be broken down into “under” and “stand” by BPE, means comprehending a concept. However, its meaning is not merely a combination of “under” and “stand”.

Fragile representation highlights BPE’s sensitivity to minor typos in a word. In other words, a small spelling mistake can result in drastic changes in the set of subwords. For instance, BPE processes “cat” as a known subword. However, if an error occurs, resulting in “cta”, BPE will break down the word into individual characters: “c”, “t”, and “a”. In this example, a minor rearrangement of letters in “cat” causes significant changes in how BPE interprets the word.

CharBERT aims to address these two problems with two tricks in the pre-training stage: 1) employ a dual-channel architectural approach for the subword and character; 2) utilize noisy language modeling (NLM) and masked language modeling (MLM) for unsupervised character representation learning. The first trick processes both subword and character-level information and fuses them, ensuring a more robust representation in case of typos. The second trick involves introducing character-level noise into words and training the model to correct these errors. This approach enhances the model’s ability to handle real-world text with variations and typos. Besides, it also masks 10% of words and trains the model to predict them. This enables CharBERT to synthesize information on a token level.

CharBERT is fine-tuned on several downstream tasks, including question answering, text classification, and sequence labeling. CharBERT outperforms BERT across all tasks. However, it encounters some challenging competition from RoBERTa, which the authors acknowledge as a robust baseline as shown in [Figure 3](#).

Although the performance of CharBERT does not significantly exceed that of RoBERTa, the model architecture of CharBERT is intriguing. Since models are usually better at the tasks they are pre-trained on, the NLM pre-training task makes CharBERT a potentially good corrector, which can be paired with the recognizer and compensate for its language deficiencies. In addition, we can further pre-train the NLM

Models	SQuAD				Text Classification			
	1.1		2.0		CoLA	MRPC	QQP	QNLI
	EM	F1	EM	F1	Corr	Acc	Acc	Acc
BERT (Devlin et al., 2019)	80.5	88.5	73.7	76.3	57.4	86.7	90.6	90.7
CharBERT	82.9	89.9	75.7	78.6	59.1	87.8	91.0	91.7
RoBERTa (Liu et al., 2019)	84.6	91.5	80.5	83.7	62.1	90.2	91.2	92.8
XLNet (Yang et al., 2019)	-	-	80.2	-	60.2	88.2	91.4	91.7
CharBERT _{RoBERTa}	84.0	90.9	81.1	84.5	61.8	90.4	91.6	93.4

Figure 3: Experimental Results of CharBERT and Strong Pre-trained Models

Image source: <https://aclanthology.org/2020.coling-main.4/>

with common OCR mistakes instead of randomly introduced errors. This makes the corrector more familiar with OCR-specific errors.

2.5 Candidate Fusion

Candidate fusion [Kang et al., 2021] is a technique that involves integrating the LM within the recognizer, i.e., letting the LM and the recognizer interact. Most SOTA word recognition models and LMs are trained separately. However, with candidate fusion, several advantages can be achieved. First, the recognizer is able to integrate insights from both its own processing and the input from the LM, resulting in a more comprehensive understanding. Second, the system is designed to assess the information supplied by the LM, allowing the recognizer to selectively weigh its importance. Lastly, the LM can learn from frequent errors generated by the recognizer, improving overall accuracy.

The models utilized for demonstrating candidate fusion have an encoder-decoder structure, where the encoder extracts features from input images, and the decoder converts the image features into text. The encoder is a CNN followed by a Gated Recurrent Unit (GRU) [Dey and Salem, 2017], and the decoder is an unidirectional multi-layered GRU. Both structures in the encoder provide spatial information about the input, allowing the decoder to follow the proper order.

In the paper, they employ two-step training. In the first step, the LM is pre-trained on a large corpus for it to understand general language and grammar. In the second step, the LM is further trained alongside handwritten datasets with the recognizer. This allows the LM to consider both its own knowledge and what the recognizer

predicts when outputting the result text. The LM will adjust its predictions by taking into account the recognizer’s decisions.

The model is being evaluated on IAM, GW [Fischer et al., 2012], and Rimes [Grosicki and El-Abed, 2011]. Among the datasets, the model shows significant improvement on GW over a strong baseline [Krishnan et al., 2018]. However, the performance of this model beyond the GW dataset remains questionable, as the improvements are relatively small. Additionally, these other datasets encompass more writers, contributing to a greater diversity in handwriting styles. Consequently, the model’s marked improvement on the GW dataset might indicate potential limitations in its adaptability to diverse handwriting styles.

Despite the potential deficiency in the model, the idea of fusing the recognizer and the LM and allowing both of them to learn from each other is a quality design worth exploring. This technique provides new opportunities for achieving more context-aware text recognition. In this study, TrOCR and CharBERT will serve as the recognizer and the LM, respectively, and will be combined. The composite model aims to explore the potential enhancements or effects they can bring when used together.

2.6 Glyph Embedding

Integrating the concept of glyphs into OCR offers substantial advantages, especially for the digitization of historical texts. This approach enhances the accuracy of character recognition by focusing on glyphs—the visual forms of characters—which is crucial for handling the varied typographic styles found in historical documents. Glyph-based OCR is adept at adapting to these styles and is particularly effective at recognizing characters in texts that are in poor condition, such as those with faded ink or smudges. By understanding the visual aspects of glyphs, such systems significantly reduce OCR errors, minimizing the need for labor-intensive post-correction. Additionally, since this method is based on visual characteristics rather than language, it can be applied across different languages and scripts, making it highly versatile.

Amrhein and Clematide [2018] explore how Neural Machine Translation, when applied at the character level, can be particularly effective for correcting glyph-based errors. This approach leverages the flexibility of neural machine translation (NMT) to adapt to the nuances of character shapes and their common misinterpretations, thereby improving the accuracy of OCR text post-correction. Overall, the inte-

gration of glyph analysis into OCR workflows as discussed in this paper not only improves the initial accuracy of text digitization but also enhances the efficiency of processing large volumes of historical documents.

3 Methodology

In this chapter, I will introduce the composite model of TrOCR and CharBERT by leveraging the ideas in Candidate Fusion mentioned in [Chapter 2](#). First, the design concept behind the model will be outlined. Following this, I will elaborate on data collection and preprocessing, and the architectures of the models developed in this study. Finally, I will delve into the details of the training process, including the utilisation of GPU resources, the optimiser and the loss function.

Rewrite
this

3.1 Design Concept

CharBERT mitigates the problems of incomplete modelling and fragile representation by including the character encoding in addition to the subword level information. Furthermore, having NLM as the pre-training task makes CharBERT effective at correcting character level typos, which is a desired feature for OCR correcting.

On the other hand, Candidate Fusion claims that having an interaction between the recogniser and the LM can enhance the performance of OCR. Thus, combining TrOCR and CharBERT is expected have an improvement on the OCR accuracy.

Rewrite
this

3.2 Data Collection and Processing

This section is dedicated to detailing the data collection process for OCR task training and CharBERT_{SMALL} training. We will outline the types and sources of data harnessed for this study, emphasizing the diversity and volume of the datasets to ensure comprehensive learning. Following the data collection overview, we will delve into the processing techniques applied to the collected data.

3.2.1 Data for OCR

We focus on the performances of handwritten datasets on the composite model. The data used in this study is the George Washington (GW) handwritten dataset¹ and the Joseph Hooker (JH) handwritten dataset². They serve as valuable benchmarks for developing and evaluating handwriting recognition systems. Note that to ensure comparability with existing studies, this research adheres to the established train-validation-test splits of the GW and JH datasets.

Although the IAM dataset is a more contemporary collection of English handwriting samples, renowned for its diversity in handwriting styles, this study did not use it since TrOCR is pre-trained on the IAM dataset. The most significant risk of fine-tuning the composite on the same dataset is overfitting. The model may become excessively tailored to the IAM dataset, performing well on this specific dataset but poorly on unseen, real-world data or other similar datasets. In addition, if the model has already been trained to a point where it performs optimally on the IAM dataset, further training the composite model on the same data might yield diminishing returns. The model may not improve significantly and could even start to fit the noise in the dataset as mentioned above.

George Washington Dataset The GW dataset is a collection of historical letters and diaries handwritten by George Washington and his secretaries in 1755. This dataset comprises various types of documents, including personal correspondence, military orders, and official communications, which provide a unique window into the early American colonial period. The dataset is frequently utilized in research focused on recognizing historical handwriting, which poses unique challenges due to the use of archaic words and phrases, and the degradation of materials over time. The different handwriting styles in the dataset, ranging from Washington’s neat writing to the quicker, less tidy ones, make it very useful for developing strong handwriting recognition systems. This dataset not only supports the preservation and accessibility of these important texts but also aids in advancing the technology needed to interpret and digitize aged manuscripts. An example from the GW dataset is shown in [Figure 4](#).

¹The George Washington handwritten dataset can be downloaded here: <https://fki.tic.heia-fr.ch/databases/washington-database>

²The Joseph Hooker handwritten dataset can be downloaded here: <https://github.com/jschaefer738b/JosephHookerHTR>

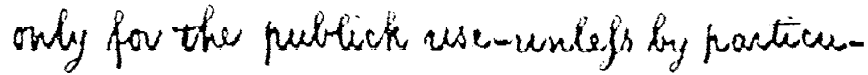
A sample of handwritten text from the GW dataset, showing the phrase "only for the publick use-unless by particu-". The handwriting is in a cursive style typical of the 18th century.

Figure 4: Sample of handwritten text from GW dataset. The excerpt reads: only for the publick use, unless by particu

Joseph Hooker Dataset The JH dataset consists of correspondence between Joseph Dalton Hooker (1817-1911) and other leading scientists of his time. This collection offers unparalleled insights into the scientific dialogues of the era, featuring interactions with notable figures such as Charles Darwin. As such, it is an invaluable resource for historians specializing in science, botany, and the Victorian period. Hooker’s letters are diverse in content, extending beyond mere botanical discussions to include observations from his travels, his perspectives on contemporary scientific debates, his personal reflections, and his professional communications. This rich variety provides a comprehensive view of his intellectual engagement and contributions. An example from the GW dataset is shown in [Figure 5](#).

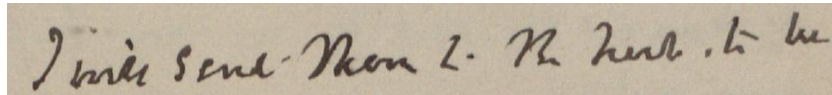
A sample of handwritten text from the JH dataset, showing the phrase "I will send them to the host to be". The handwriting is in a cursive style typical of the 19th century.

Figure 5: Sample of handwritten text from JH dataset. The excerpt reads: I will send them to the host to be

The Joseph Hooker and George Washington datasets, while serving distinct academic purposes, both play significant roles in the realm of handwritten text recognition (HTR) tasks. The Joseph Hooker dataset, with its extensive collection of 19th-century botanical writings and correspondence, provides a unique challenge for HTR technologies due to the scientific terminology and personal handwriting styles found in the documents. This variety enables developers to fine-tune HTR models to handle complex vocabulary and diverse manuscript formats, which is essential for digital humanities projects focusing on scientific archives.

Similarly, the George Washington dataset, consisting of an array of 18th-century materials including letters, diaries, and official documents, presents its own set of challenges for HTR. The historical significance of these documents demands a high level of accuracy in digital transcription. Washington’s dataset helps in training HTR systems to recognize older forms of English script and the idiosyncrasies of historical American handwriting, which are crucial for preserving and making accessible key documents from the formative years of the United States.

In both cases, these datasets not only support the preservation and accessibility of historical texts but also advance the development of HTR systems. They enable the refinement of algorithms capable of interpreting a wide range of handwritten styles and orthographic conventions, thus broadening the utility of HTR technologies across different historical and scientific domains.

Transcription Ground Truth Processing The transcription ground truths of the JH dataset are stored in XML files, where parsing these files is sufficient to retrieve the transcription texts. On the other hand, the transcription ground truths of the GW dataset follow a more complex format. In this dataset, individual characters within words are separated by hyphens (“-”), and words themselves are separated by vertical bars (“|”). Additionally, punctuation marks are represented by special characters, with a specific table detailing the replacements for each punctuation mark. The table for the punctuation replacement can be found in this [link](#). For instance, consider the transcription from the GW dataset for the phrase “of the Virginia Regiment.” This is encoded as

o-f|t-h-e|V-i-r-g-i-n-i-a|R-e-g-i-m-e-n-t-s_pt

, where `s_pt` is replaced with a period.

Image Processing In this study, images are resized and normalized prior to being input into the model. Resizing and normalizing images are standard preprocessing steps in image processing, particularly in the context of machine learning and computer vision. Each of these processes serves important purposes in preparing image data for models, enhancing model performance, and ensuring consistency. This process is illustrated in [Figure 6](#).

In this study, images are initially resized to 384x384 pixels to comply with the input requirements of the pretrained TrOCR model. Resizing images is a common preprocessing step in image processing pipelines, particularly for deep learning, where input dimensions need to be consistent across all data samples for the model to process them.

Following resizing, the images undergo normalization. The normalization specifies the mean for each of the three color channels (Red, Green, Blue), which are all set to 0.5. In this case, 0.5 is subtracted from each channel. This value shifts the range of pixel values from $[0, 1]$ to $[-0.5, 0.5]$. The normalization also sets the standard deviation for each color channel to 0.5. Dividing by 0.5 scales the range from $[-0.5,$

0.5] to $[-1, 1]$. This practice is common for neural network inputs as it tends to make the training process more stable and helps it converge faster. Together, resizing and normalizing reposition the data so that its distribution is symmetric around zero, reducing the bias that input values might otherwise introduce into the network's computations. This method ensures that no single pixel range overly influences the network due to its scale, allowing the model to focus more on learning the patterns rather than adapting to the scale of input data.

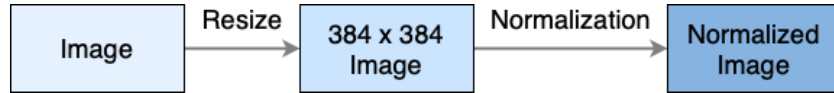


Figure 6: Flowchart illustrating the preprocessing steps for images used in the study.

3.2.2 Data for Training CharBERT_{SMALL}

In the original study, the authors trained CharBERT using a 12GB dataset from Wikipedia. This training process spanned 5 days, utilizing the computational power of two NVIDIA Tesla V100 GPUs. Given constraints in time and computational resources, our approach involves testing a scaled-down version of CharBERT, which we have designated as CharBERT_{SMALL}. This variant was trained on a significantly smaller dataset, specifically 1.13GB of English Wikipedia data, from which sentences were randomly sampled. For example, sentences like "Tony Macrie has been president of the Seashore Lines since he formed the railroad in 1984." were randomly sampled from articles, providing a rich yet manageable training substrate. This adaptation allows us to evaluate the performance of CharBERT under more restricted conditions, ensuring our experiments are feasible within our available resources.

Add eme

3.3 Composite Model Architecture

The end-to-end composite model described in this thesis integrates TrOCR and CharBERT to enhance text recognition accuracy. TrOCR is responsible for the initial text recognition task, while CharBERT functions as a corrector, refining the outputs from TrOCR. Specifically, the input to the TrOCR decoder is first corrected by CharBERT before it is processed by the TrOCR decoder. This step ensures that the TrOCR decoder works with embeddings that have been optimized by CharBERT, potentially improving the accuracy of the text recognition. In this

section, we will delve into the architectures of both TrOCR and CharBERT, and discuss the methodology behind their integration within the composite model.

As depicted in Figure 7, the process begins with the image being processed by the TrOCR encoder. Subsequently, the input to the TrOCR decoder is first refined by CharBERT (highlighted in purple) before being fed into the TrOCR Decoder, following the standard transformer architecture.

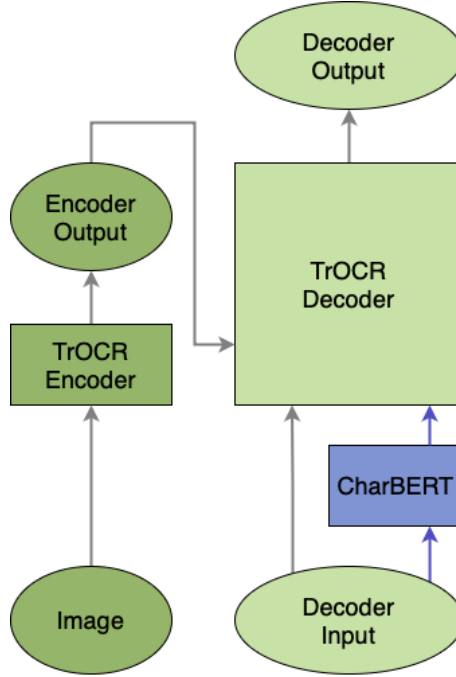


Figure 7: Schematic of the integrated workflow in the composite model combining TrOCR and CharBERT.

3.3.1 Notations

In this report, we denote a one-dimensional tensor (vector) using boldface lowercase letters, while a multi-dimensional tensor appears as boldface uppercase letters. The TrOCR sequence IDs are denoted by $\mathbf{W} = \mathbf{w}_1, \dots, \mathbf{w}_i, \dots, \mathbf{w}_D$, where each \mathbf{w}_i represents a token tokenized by the TrOCR tokenizer, and D signifies the length of the sequence.

3.3.2 Recogniser - TrOCR

TrOCR is composed of an image encoder and a text decoder. The image encoder takes the pixel values of an image as input, and the encoder output is then fed

into the decoder for text generation. This study involves adapting the decoder to integrate with CharBERT. Therefore, it will be crucial to focus on and provide a detailed explanation of its architecture. The following contents will first focus on the inputs of the decoder and the label that is used for loss calculation. Then, we will elaborate on how the inputs are processed through the decoder.

Decoder Inputs and Labels The text decoder takes the encoder output, decoder input IDs, and padding attention mask³ as inputs, and outputs the generated text. The decoder output is then compared to the label tensor for loss calculation. This loss is used to update the gradients of the parameters during training.

The encoder output is a set of features extracted from the pixel values of the image. These features represent a transformed version of the input image, capturing not only the patterns but also the arrangements and relationships among elements within the image.

Decoder input IDs are tensors of token IDs converted from the label texts. These include three special tokens: `<bos>`, `<eos>`, and padding. The ID `<bos>`, represented by the number 0, indicates the start of the text, while the ID `<eos>`, denoted by the number 2, signifies the end of the text. Padding, a series of the number 1, is appended after `<eos>` to ensure all input IDs are of uniform length. The decoder input IDs, generated automatically by the TrOCR tokenizer, start with `<bos>` and end with `<eos>` and padding, mimicking the inference process during training⁴. The whole process is illustrated in [Figure 8](#).

The padding attention mask, another input for the decoder, determines which parts of the data the TrOCR should ignore, consisting of 0s and 1s where the 0s cover the padding portions of the decoder input IDs.

The label tensor used for loss calculation is the decoder input IDs minus the `<bos>` token. Since the TrOCR model is based on the VisionEncoderDecoderModel, it requires label paddings to be set to -100. Thus, the padding token in the label tensor is replaced with -100 instead of 1s.

³Note that the padding attention mask should not be confused with the causal attention mask. The padding attention mask prevents the model from attending to the paddings, while the causal attention mask ensures that the prediction for a specific token in the sequence can only be influenced by previously generated tokens.

⁴During inference, the decoder output from the previous step becomes the new decoder input. However, as no output sequence has been generated at the first step, the initial decoder input comprises only the `<bos>` token. This is processed by the decoder to produce an output, which is then used to update the decoder input.

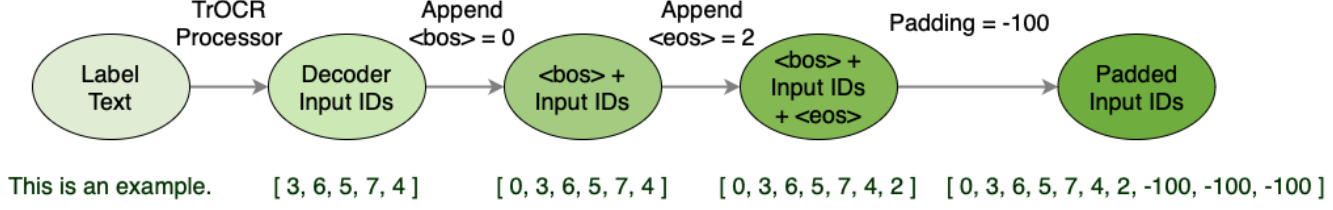


Figure 8: Processing pipeline for transforming label text into padded input IDs for the TrOCR model.

Inside the Decoder Let’s first consider the original decoder before adapting it to integrate with CharBERT. The decoder receives three primary inputs: the decoder input IDs, the padding attention mask, and the encoder output. Initially, the decoder input IDs are converted into decoder embeddings. These embeddings are then combined with positional encoding, which preserves spatial information, before being processed by the decoder stacks. The decoder stacks include masked multi-head attention, multi-head attention, and a feedforward neural network (FFNN). The padding attention mask is applied during both the masked multi-head attention and multi-head attention stages to prevent the TrOCR from attending to the padding tokens. Finally, the output from the decoder stacks passes through a linear layer and a softmax layer to generate the final output sequence.

3.3.3 Corrector - CharBERT

CharBERT features a dual-channel architecture, comprising token and character channels. The token channel mirrors the structure of BERT or RoBERTa, depending on the initialization. In this study, we will specifically focus on CharBERT_{RoBERTa}. Henceforth in this document, any reference to ‘CharBERT’ will pertain exclusively to CharBERT_{RoBERTa}, unless otherwise specified. Unlike BERT and RoBERTa, which output a single embedding, CharBERT generates both token-level and character-level embeddings. Given that the token channel’s architecture is identical to that of BERT/RoBERTa, the subsequent sections will concentrate on the character channel and its heterogeneous interactions.

Character Channel & Heterogeneous Interaction ”The character channel of CharBERT begins by splitting the input text into individual characters and converting these characters into IDs via a dictionary lookup. These IDs are then embedded, followed by processing through a bidirectional GRU (Bi-GRU) layer to generate the

output embeddings. These embeddings, at both the character and token levels, are subsequently passed through a transformer and undergo heterogeneous interaction, which is comprised of two parts: fusion and divide. The fusion process allows the embeddings to enrich each other through the use of a feedforward neural network (FFNN) and a convolutional neural network (CNN), enhancing their mutual characteristics. Conversely, the divide process ensures that each set of embeddings retains its unique features, facilitated by an FFNN and a residual connection. This residual connection is crucial as it helps preserve the distinct information from each embedding type. CharBERT repeats the transformer and heterogeneous interaction cycles to capture more features and information inherent in the input texts.

3.3.4 Composite Model

The composite model is ingeniously designed to integrate TrOCR and CharBERT. During the inference phase, the decoder output is recycled back as input in a feed-back loop. Before this recycled input is fed back into the decoder, it undergoes correction and refinement by CharBERT. Consequently, CharBERT is positioned between the decoder input and the decoder stacks to ensure that the input to the decoder is optimized in each iteration of the process. However, integrating these systems presents several challenges: 1) The TrOCR decoder accepts token IDs as input, whereas CharBERT outputs embeddings; 2) CharBERT requires textual inputs, but the input to the TrOCR decoder is a tensor; 3) The embedding representations of TrOCR do not align with those of CharBERT; 4) The input to the TrOCR decoder is a single tensor, while CharBERT produces dual-channel outputs. The subsequent sections will discuss these issues in detail and explore potential approaches to resolve them.

Adapted TrOCR Before addressing the first problem, it is crucial to thoroughly understand the issue. The TrOCR decoder is specifically designed to accept token IDs as input, which are then mapped to embeddings. These embeddings are subsequently augmented with positional encoding before being fed into the Transformer decoder. The challenge arises from the outputs of CharBERT, which are embeddings rather than token IDs, leading to a compatibility issue. If we were to resolve this problem by converting CharBERT’s embeddings back into token IDs for input into the TrOCR decoder, a new issue surfaces. Using token IDs as intermediaries between model components is problematic because token IDs are integers, while during the training process, the model weights are updated as floating-point numbers. Converting these weights to integers is not practical; while rounding could technically

convert floats to integers, this approach would likely distort the learning process and could render the results meaningless for the intended task. Therefore, the only viable solution is to adapt the TrOCR decoder to accept embeddings directly.

A straightforward solution involves repositioning the embedding layer from the TrOCR decoder to precede CharBERT. This adjustment ensures that token IDs are initially converted to TrOCR embeddings, which are then input into CharBERT for correction. Consequently, the adapted TrOCR can accept embeddings directly, bypassing the need for token IDs. This modification not only ensures that the outputs from CharBERT are seamlessly integrated into the TrOCR decoder but also eliminates the use of token IDs between model components. This entire modification process is illustrated in Figure 9.

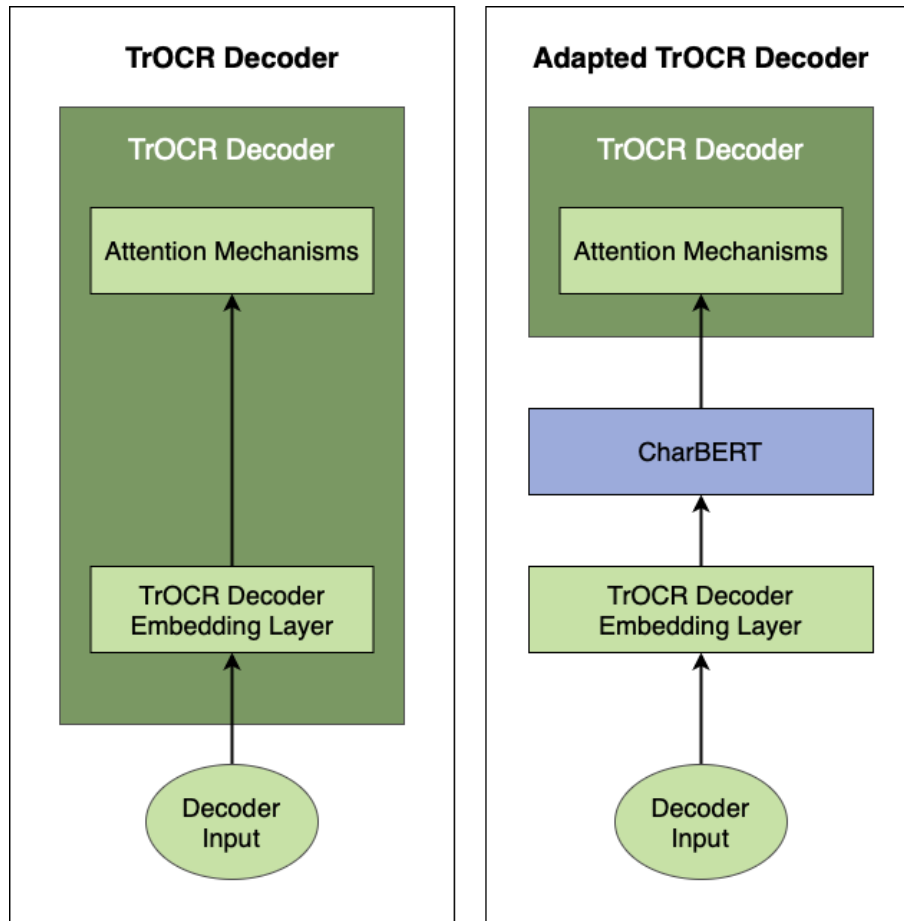


Figure 9: **Comparison of the TrOCR Decoder architecture**

On the left, the original TrOCR Decoder structure includes the Decoder Input, TrOCR Decoder Embedding Layer, and Attention Mechanisms. On the right, the adapted TrOCR Decoder structure shows the repositioning of the TrOCR Decoder Embedding Layer below CharBERT, indicating the integration of CharBERT into the decoder process to enhance input handling before the attention mechanisms.

Adapted CharBERT ”The second problem addresses the mismatch between the data types of input and output. According to the modifications described in [Adapted TrOCR](#), the TrOCR decoder input is now an embedding, which should be processed by CharBERT for correction. However, CharBERT traditionally only accepts text as input. Therefore, it becomes essential to develop an adapted version of CharBERT that can handle token and character embeddings directly as inputs. In this revised model, CharBERT no longer converts text into IDs and then into embeddings; instead, it receives pre-processed embeddings directly. This adaptation allows both token and character embeddings to be processed through their respective channels in CharBERT. A comparison between the original and the adapted CharBERT models is illustrated in [Figure 10](#).

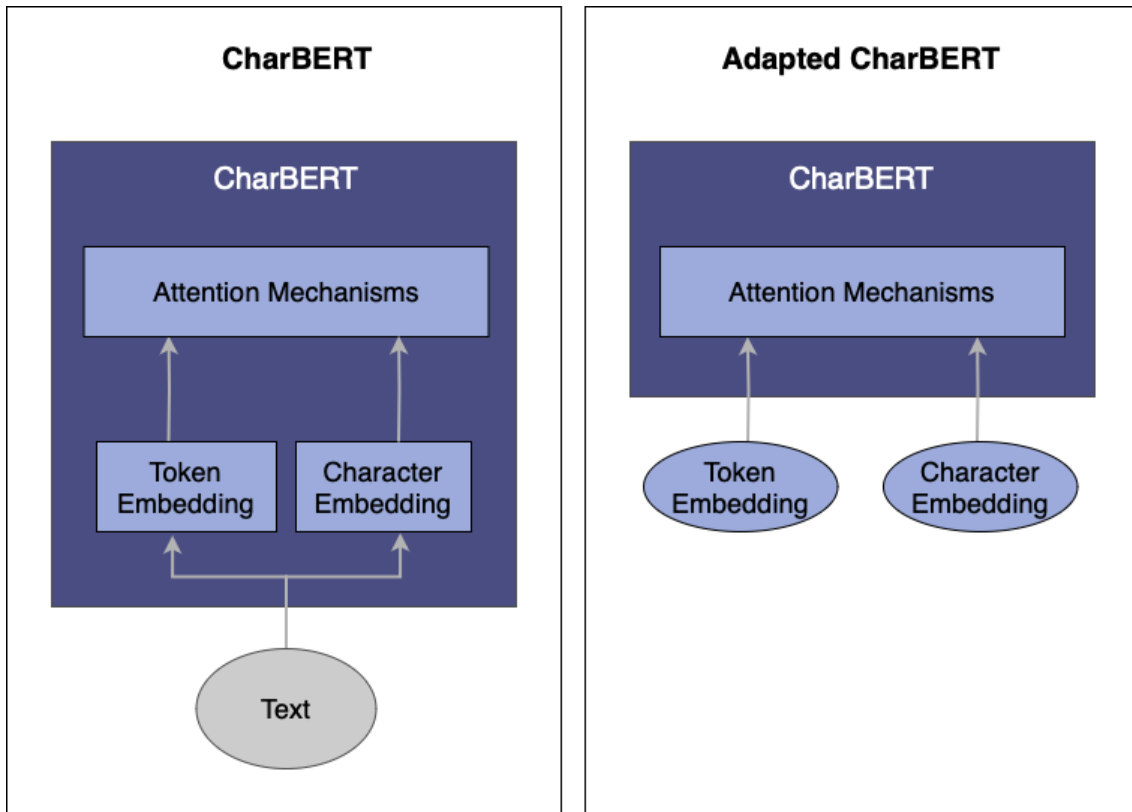


Figure 10: **Comparison of the CharBERT architecture**

On the left, the original CharBERT model processes text into token and character embeddings, which are then fed into the attention mechanisms. On the right, the adapted CharBERT directly receives token and character embeddings as inputs, bypassing the initial text processing stage, and processes them through the same attention mechanisms.

Tensor Transform The third problem presents a more complex challenge. The TrOCR decoder relies on embeddings that are unique to TrOCR. Even for identical

texts, the embedding representations generated by TrOCR and CharBERT differ significantly, further complicated by CharBERT’s dual-channel embeddings. Not only do the models represent the same text differently, but their embedding dimensions are also incompatible. As a result, even though CharBERT is capable of processing embeddings directly, the TrOCR decoder input cannot be directly integrated into CharBERT without modifications. To overcome this issue, an architecture comprising CNN and FFNN is utilized. This strategy is designed to adjust the dimensions of the TrOCR decoder input to match those of the CharBERT token and character embeddings. Additionally, this dimensional transformation facilitates an effective alignment of representations between TrOCR and CharBERT. To clarify the architecture of the tensor transformation, dimensions will be noted after each input and output, using the format (batch size, embedding size).

”The objective of the tensor transform architecture is to adjust the dimensions of the TrOCR decoder input, which are initially set at (batch size, 512, 1024), to match the dimensions required by CharBERT’s token embeddings (batch size, 510, 768) and character embeddings (batch size, 3060, 256). In this architecture, the second dimension represents the sequence length, and the third dimension represents the embedding size. The transformation process is divided into two distinct stages. In the first stage, the decoder input passes through a series of CNN layers, which are interspersed with LeakyReLU activation functions and batch normalization, specifically designed to adjust the sequence dimension (dim=1). Subsequently, the output from the first stage is processed through FFNN layers, with interspersed LeakyReLU activations, in the second stage to modify the embedding dimension (dim=2).

$$\begin{aligned}
\mathbf{t}_{1,j} &= \text{LeakyReLU}(\mathbf{b}_1 + \sum_{k=1}^3 \mathbf{W}_{1,k} \cdot \mathbf{e}_{i+k-1}) \quad ; \quad \mathbf{t}'_1 = \text{Batch_Norm}(\mathbf{t}_1) \\
\mathbf{t}_{2,l} &= \text{LeakyReLU}(\mathbf{b}_2 + \sum_{k=1}^3 \mathbf{W}_{2,k} \cdot \mathbf{t}'_{1,j+k-1}) \quad ; \quad \mathbf{t}'_2 = \text{Batch_Norm}(\mathbf{t}_2) \\
\mathbf{t}_{3,p} &= \text{LeakyReLU}(\mathbf{b}_3 + \sum_{k=1}^3 \mathbf{W}_{3,k} \cdot \mathbf{t}'_{2,l+k-1})
\end{aligned} \tag{3.1}$$

After the application of the CNN layers, the dimension of the decoder input is adjusted to (batch size, 510, 1024), aligning the sequence dimension (dim=1) with the desired dimension for the CharBERT token embedding. The CNN layers are specifically chosen for their ability to expand or contract the sequence length while preserving spatial information, making them more suitable than FFNNs for adjust-

ing sequence dimensions. Additionally, it is important to note the inclusion of batch normalization steps between the convolutional layers. These batch normalizations stabilize the deep model and maintain healthier gradients, especially important due to the presence of activation functions⁵. Following this, the tensor transformation process moves to the second stage, which utilizes FFNN layers.

$$\begin{aligned}\mathbf{T}_4 &= \text{LeakyReLU}(\mathbf{b}_4 + \mathbf{W}_4 \cdot \mathbf{T}_3) \\ \mathbf{T}_5 &= \text{LeakyReLU}(\mathbf{b}_5 + \mathbf{W}_5 \cdot \mathbf{T}_4) \\ \mathbf{T}_n &= \text{LeakyReLU}(\mathbf{b}_6 + \mathbf{W}_6 \cdot \mathbf{T}_5)\end{aligned}\tag{3.2}$$

The resulting tensor, the CharBERT token embedding, is denoted as $\mathbf{T} = \mathbf{t}_1, \dots, \mathbf{t}_n, \dots, \mathbf{t}_N$, where N represents the token sequence length and the tensor size is (batch size, 510, 768).

Similarly, operations are applied to the TrOCR decoder input but adjusted for different dimension expansion to produce the CharBERT character embedding $\mathbf{C} = \mathbf{c}_1, \dots, \mathbf{c}_m, \dots, \mathbf{c}_M$, where M denotes the character sequence length. It is established that M is typically six times that of N , based on the assumption that the average word contains six characters.

The Tensor Transform layer is specifically designed to convert the TrOCR decoder input into CharBERT token and character embeddings. The CNN and FFNN layers are instrumental not only in aligning the dimensions between the tensors but also in learning to map the contextual information from TrOCR embeddings to those of CharBERT, effectively adapting the embeddings for integrated processing.

Tensor Combine The fourth problem arises because CharBERT produces two separate tensors—token and character representations—while the TrOCR decoder input requires a single tensor. To address this, the solution involves combining the two output tensors from CharBERT into a single tensor. Additionally, to stabilize the deep model, a residual connection from the original TrOCR decoder embedding can be added. This residual connection helps to reuse features from the original TrOCR decoder embedding and prevents gradient vanishing.

Among the three tensors involved, two are derived from CharBERT. Consequently, these representations are not inherently compatible with the TrOCR decoder, and their dimensions do not match. Thus, they must first undergo a transformation via

⁵With activation functions in deep networks, gradients can easily explode or vanish. Batch normalization helps maintain a healthier gradient flow in the network, which can improve the efficiency of backpropagation and thereby enhance the learning process.

the **Tensor Transform** process before they can be effectively combined.

$$\begin{aligned}\mathbf{T} \in \mathbb{R}^{d_N} &\rightarrow \mathbf{T}' = \text{Tensor_Transform}(\mathbf{T}) \in \mathbb{R}^{d_D} \\ \mathbf{C} \in \mathbb{R}^{d_M} &\rightarrow \mathbf{C}' = \text{Tensor_Transform}(\mathbf{C}) \in \mathbb{R}^{d_D}\end{aligned}\tag{3.3}$$

where $d_D = (\text{batch size}, 512, 1024)$; $d_N = (\text{batch size}, 510, 768)$; $d_M = (\text{batch size}, 3060, 256)$.

After the tensor transform module is applied, the three tensors will all conform to the size $(\text{batch size}, 512, 1024)$, matching the original input size of the TrOCR decoder. These tensors can then be merged and fed into the TrOCR decoder stack. In this study, we explore four different architectures for the tensor combination module: 1) simple addition of all tensors; 2) mean pooling; 3) utilization of linear layers as an attention network; 4) implementation of convolutional layers as an attention network.

Tensor Combine 1: Adding The first tensor transform module aggregates three tensors into a single tensor by performing an element-wise addition of the three input tensors. Specifically, this involves the TrOCR decoder embeddings \mathbf{E} , the CharBERT transformed token representation \mathbf{T}' , and the CharBERT transformed character representation \mathbf{C}' . Each of these contributes equally to the formation of the combined tensor, ensuring that the information from each input tensor is weighted equally. The operation can be mathematically expressed as follows:

$$\mathbf{E}'_{1,ijk} = \mathbf{E}_{ijk} + \mathbf{T}'_{ijk} + \mathbf{C}'_{ijk}\tag{3.4}$$

where $i = 1, 2, \dots, \text{batch size}$; $j = 1, 2, \dots, 512$; $k = 1, 2, \dots, 1024$.

Tensor Combine 2: Mean Pooling The second tensor combination module is designed to dynamically allocate attention weights to each word across the three input tensors. These weights are derived by performing max pooling along the embedding axis of the three tensors, once they are stacked together. After obtaining these weights, they are applied to the stacked embeddings. It is important to note that all three tensors undergo feature-wise normalization prior to stacking. This

normalization aids in faster model convergence and promotes better generalization.

$$\begin{aligned}
\mathbf{T}'_{\text{norm}} &= \text{Feature_Norm}(\mathbf{T}') \\
\mathbf{C}'_{\text{norm}} &= \text{Feature_Norm}(\mathbf{C}') \\
\mathbf{E}''_{\text{norm}} &= \text{Feature_Norm}(\mathbf{E}') \\
\mathbf{S} &= \text{Stack}(\mathbf{E}', \mathbf{T}'_{\text{norm}}, \mathbf{C}'_{\text{norm}}; \text{axis} = 1) \in \mathbb{R}^{(\text{batch size}, 3, 512, 1024)} \\
\mathbf{P} &= \frac{1}{1024} \sum_{j=1}^{1024} \mathbf{S}_{a,b,c,j} \in \mathbb{R}^{(\text{batch size}, 3, 512)} \\
\mathbf{E}'' &= \mathbf{P} \cdot \mathbf{S}
\end{aligned} \tag{3.5}$$

Tensor Combine 3: Linear Layers as Attention Net ”The third tensor combination module builds upon the second module by incorporating linear layers as the attention network. Rather than merely performing mean pooling, this module employs linear layers interspersed with activation functions to more effectively capture relevant information when determining the weights. The process is described as follows:

$$\begin{aligned}
\mathbf{T}'_{\text{norm}} &= \text{Feature_Norm}(\mathbf{T}') \\
\mathbf{C}'_{\text{norm}} &= \text{Feature_Norm}(\mathbf{C}') \\
\mathbf{P} &= \text{Stack}(\mathbf{E}, \mathbf{T}'_{\text{norm}}, \mathbf{C}'_{\text{norm}}; \text{axis} = 1) \in \mathbb{R}^{(\text{batch size}, 3, 512, 1024)}
\end{aligned} \tag{3.6}$$

For $i = 1, 2, \dots, 512$:

Conv Attention Net

$$\begin{aligned}
\mathbf{S}' &= \mathbf{S}[:, :, i, :] \in \mathbb{R}^{(\text{batch size}, 3, 1024)} \\
\mathbf{S}'_1 &= \text{LeakyReLU}(\mathbf{b}_7 + \mathbf{W}_7 \cdot \mathbf{S}') \\
\mathbf{S}'_2 &= \text{LeakyReLU}(\mathbf{b}_8 + \mathbf{W}_8 \cdot \mathbf{S}'_1) \\
\mathbf{S}'_3 &= \mathbf{b}_9 + \mathbf{W}_9 * \mathbf{S}'_2 \\
\mathbf{P}_i &= \text{softmax}\left(\frac{1}{1024} \sum_{j=1}^{1024} \mathbf{S}'_{3,(a,b,j)}\right) \in \mathbb{R}^{(\text{batch size}, 3)}
\end{aligned} \tag{3.7}$$

$$\begin{aligned}
\mathbf{P} &= [\mathbf{P}_1; \mathbf{P}_2; \dots; \mathbf{P}_{512}] \in \mathbb{R}^{(\text{batch size}, 3, 512)} \\
\mathbf{E}' &= \mathbf{P} \cdot \mathbf{S}
\end{aligned} \tag{3.8}$$

Tensor Combine 4: Convolutional Layers as Attention Net ”The fourth tensor combination module substitutes linear layers with convolutional layers in the attention network. The equations below illustrate this attention network, which differs

from the one described in [Tensor Transform - Linear Layers as Attention Net](#).

Linear Attention Net

For $i = 1, 2, \dots, 512$:

$$\mathbf{S}' = \mathbf{S}[:, :, i, :] \in \mathbb{R}^{(\text{batch size}, 3, 1024)}$$

$$\mathbf{S}'_1 = \text{LeakyReLU}(\mathbf{b}_7 + \mathbf{W}_7 \cdot \mathbf{S}')$$

$$\mathbf{S}'_2 = \text{LeakyReLU}(\mathbf{b}_8 + \mathbf{W}_8 \cdot \mathbf{S}'_1)$$

$$\mathbf{S}'_3 = \mathbf{b}_9 + \mathbf{W}_9 * \mathbf{S}'_2$$

$$\mathbf{P}_i = \text{softmax}\left(\frac{1}{1024} \sum_{j=1}^{1024} \mathbf{S}'_{3,(a,b,j)}\right) \in \mathbb{R}^{(\text{batch size}, 3)}$$

(3.9)

Change
to conv
version

3.4 Glyph Incorporation

In our methodology, we enhance the training process by specifically targeting commonly misrecognized characters, such as “.” and “,” or “O” and “o”, with the aim of reducing the likelihood of these errors in future recognitions. To achieve this, we commence by determining the probability \mathcal{P}_{ij} , where i represents the correct character that has been erroneously recognized as character j . This strategy is a deviation from the CharBERT NLM training approach, which incorporates character-level errors into the text at random.

By leveraging \mathcal{P}_{ij} , we refine our training methodology to introduce errors in a more systematic manner, based on the observed probabilities of specific misrecognitions. This targeted approach allows us to focus the model’s learning on correcting these particular errors, enhancing its accuracy and reliability in distinguishing between characters that are commonly confused.

3.4.1 Get \mathcal{P}_{ij}

To obtain \mathcal{P}_{ij} , it is first necessary to calculate the frequency of each character misrecognized by the recognizer (TrOCR), termed as *misrecognized frequency*. This process begins with employing the basic TrOCR model to process the GW and IAM datasets. After generating text outputs with TrOCR, these are compared with the corresponding text labels to determine the frequencies of characters that were misrecognized.

To align the generated outputs with the labels, we utilize the “Bio” package in Python, specifically referring to Biopython. This toolkit is primarily designed for biological computation, handling data such as DNA, RNA, and protein sequences, and offers extensive features for sequence analysis and alignments.

Although Biopython is not inherently designed for processing natural language texts, its tools for sequence matching and regular expressions can be adapted for text analysis applications. For instance, in our context, we can apply Biopython’s methodologies to perform precise character-by-character matching within text strings, identifying and annotating discrepancies. This approach demonstrates how Biopython’s core functionalities can be repurposed for non-biological applications, providing valuable methods for text data analysis and enhancing our understanding of text misrecognition patterns.

Add example

3.4.2 Training CharBERT \mathcal{P}_{ij}

To train CharBERT_{SMALL}, we rigorously follow the methodologies delineated in the original CharBERT study. Specifically, the text undergoes processing through the dual-channel architecture, which encompasses both token and character channels, as elaborated in [Subsection 3.3.3](#). Additionally, we adhere to the hyperparameter settings outlined in the original CharBERT publication, details of which are available at this [link](#).

This adherence ensures that our training regimen remains consistent with established benchmarks, facilitating reliable comparisons and validations of our findings. The dual-channel architecture enables CharBERT to comprehensively analyze textual information at both granular and holistic levels, enhancing its learning efficacy. By maintaining the original hyperparameters, we aim to replicate the environment under which CharBERT was initially validated, thereby ensuring that any deviations in performance or outcomes can be attributed to the differences in training data or integration nuances rather than changes in the fundamental training conditions.

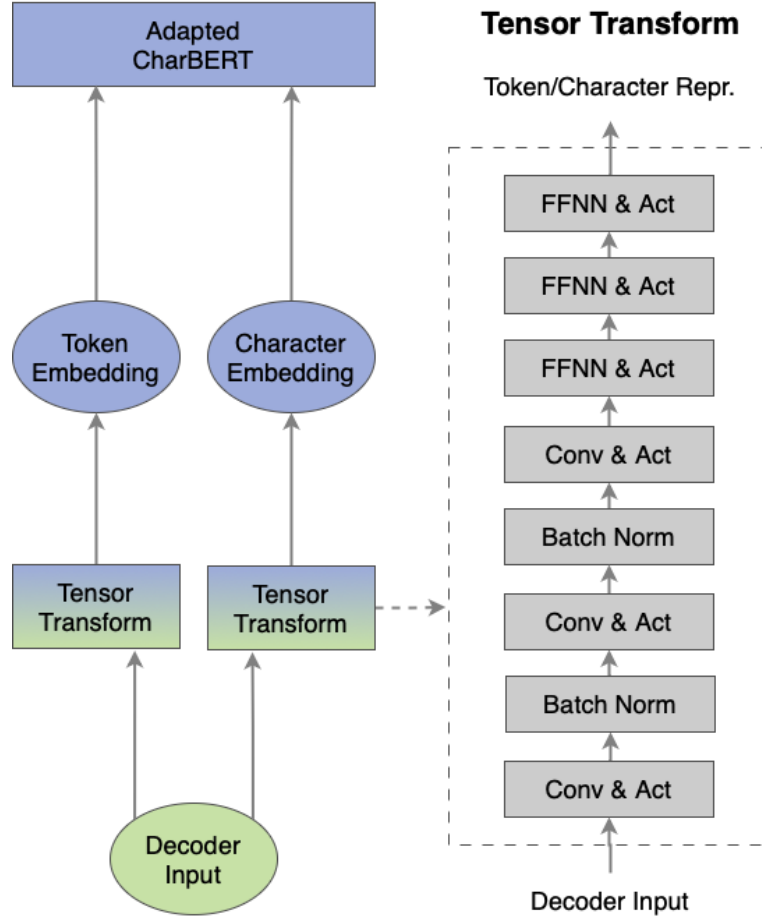


Figure 11: **Architectural flow of the adapted CharBERT and its integration with the TrOCR decoder through tensor transformation**

The diagram shows how token and character embeddings from the adapted CharBERT are processed through separate tensor transformation pathways to align their dimensions and formats with the requirements of the TrOCR decoder input. Each transformation pathway involves a series of convolutional (Conv) and activation (Act) layers, interspersed with batch normalization (Batch Norm) and further refined by feedforward neural networks (FFNN) and additional activation layers, culminating in the combined decoder input.

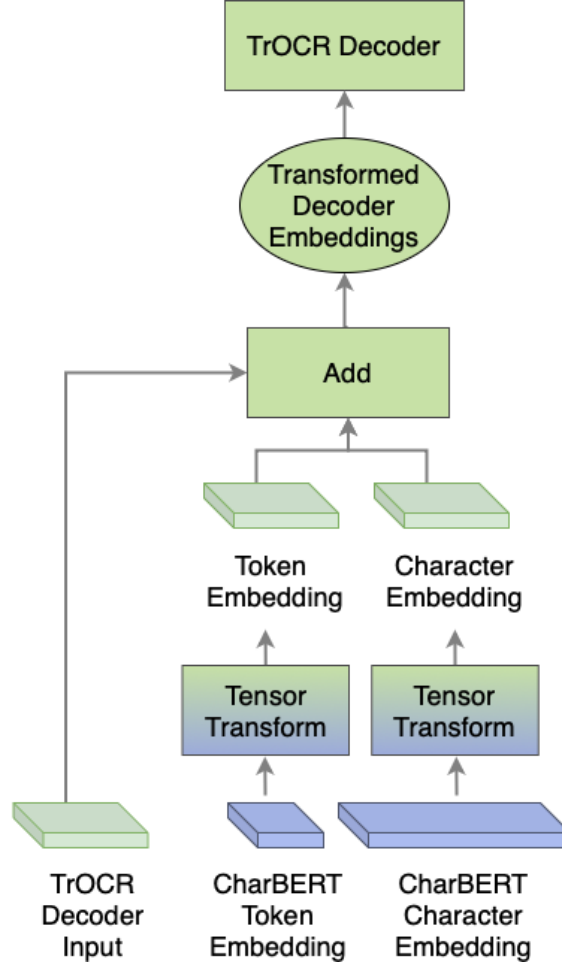


Figure 12: **Architecture of the Tensor Combine Module 1**

This diagram illustrates the flow from the TrOCR decoder and CharBERT token and character embeddings through their respective tensor transform processes. The transformed embeddings are then combined using an element-wise addition to form the transformed decoder embeddings, which are subsequently fed back into the TrOCR decoder.

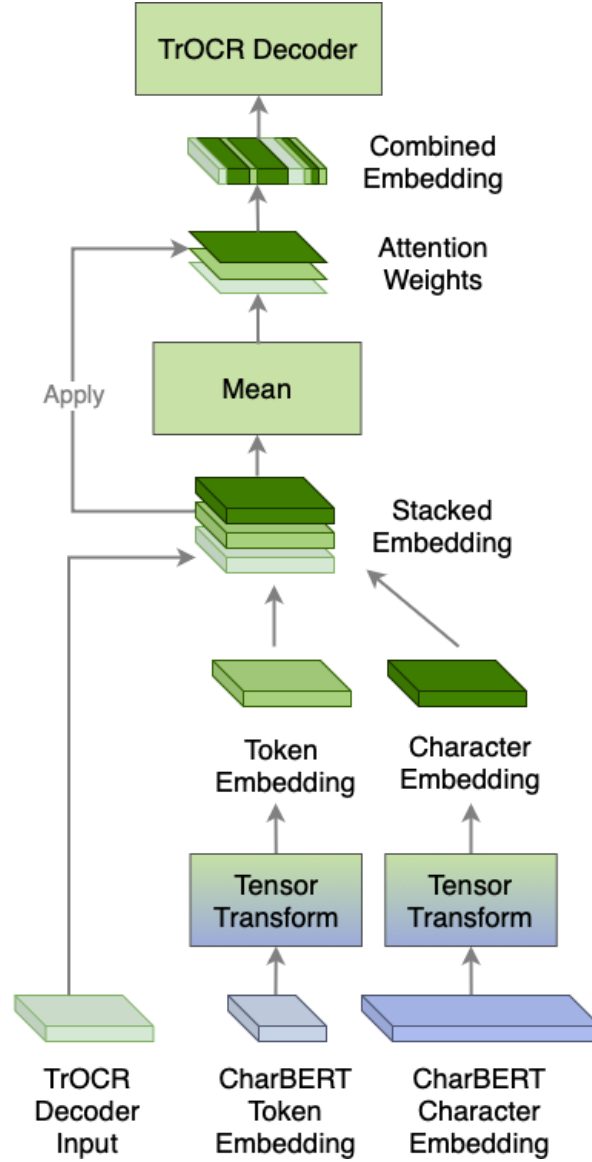
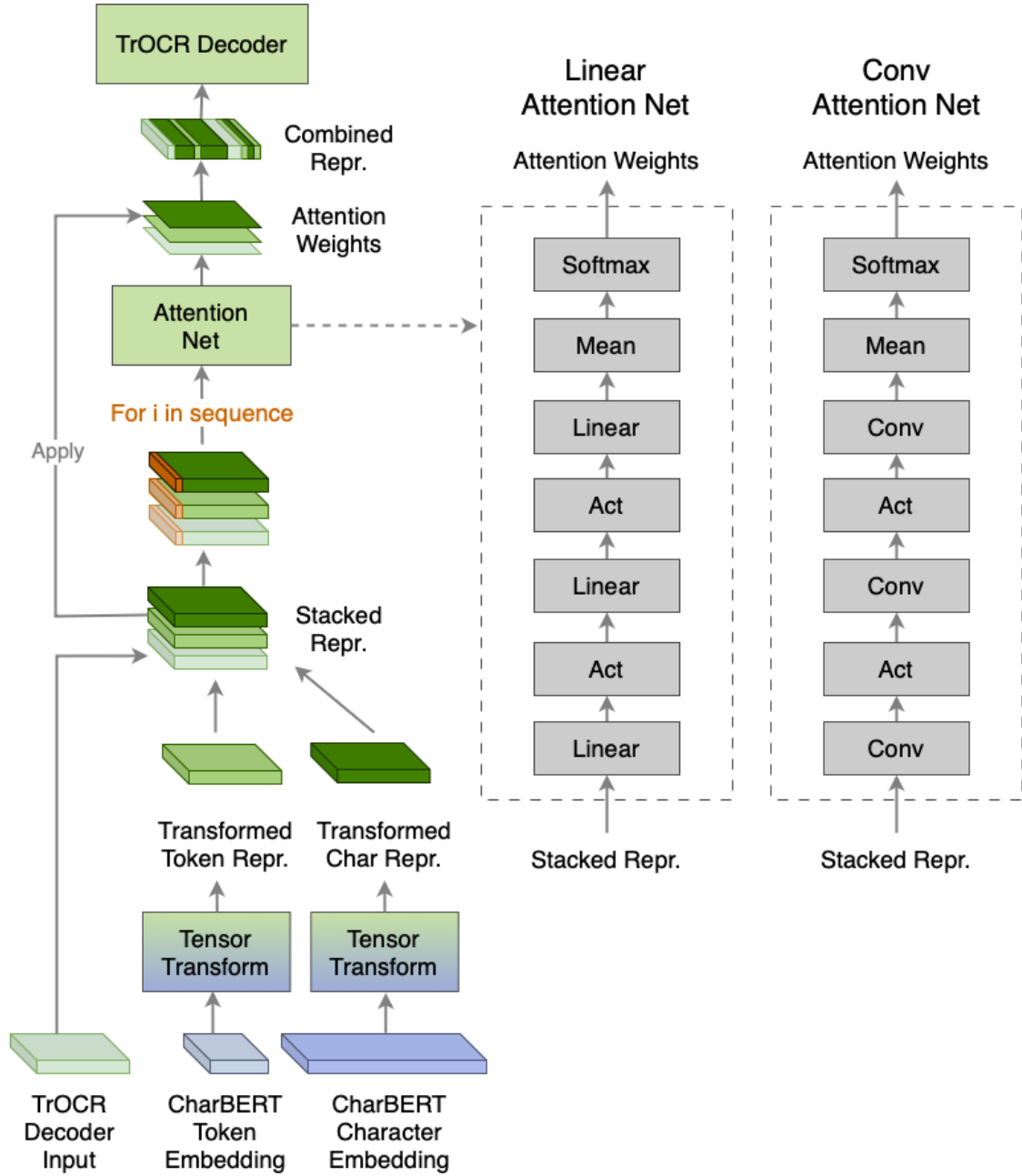


Figure 13: **Architecture of the Tensor Combine Module 2**

This module utilizes mean pooling to dynamically allocate attention weights across the three input tensors. After the tensors are stacked, attention weights are derived and applied, followed by mean pooling to merge the embeddings into a single combined embedding, which is then processed by the TrOCR decoder.

Figure 14: **Architecture of the Tensor Combine Module 3 and 4**

For each input sequence, the tensors are stacked and transformed through either linear or convolutional layers (detailed in the right two panels) to generate attention weights via a softmax function. The weights are then applied to compute a weighted mean, resulting in an attention-modified combined representation, which is subsequently fed into the TrOCR decoder.

4 Experiment

This chapter presents a detailed examination of the experimental setup and evaluation criteria in our study. Beginning with the data collection and processing section, we outline the sources and types of data utilised to benchmark the performance of our composite model and the training of CharBERT_{SMALL}. This is followed by a discussion on the baseline model, where we compare both the pre-trained and fine-tuned versions of the TrOCR model to establish performance benchmarks. The subsequent sections delve into the training and evaluation criteria for the composite model and CharBERT_{SMALL}. They highlights the strategic choices made in terms of optimizers, loss functions, and hyperparameters. In addition, we address common training challenges and our solutions. Finally, the analysis section explores the impact of specific model components and features on overall performance. This comprehensive approach not only enables a deeper understanding of the models' capabilities and limitations but also guides the refinement of OCR technologies for enhanced performance and efficiency.

Rewrite
this

4.1 Baseline Model

In this study, we consider both the pre-trained and fine-tuned versions of the TrOCR model as baseline models for comparison. The rationale behind using the fine-tuned TrOCR model alongside its pre-trained counterpart is to ascertain the performance benchmark. If the fine-tuned TrOCR model outperform the composite model, it would suggest that further fine-tuning of the TrOCR model is a more efficient approach than employing the larger, more complex composite model. This comparison allows us to evaluate the efficiency of the fine-tuning process relative to the integration of additional model components.

Pre-trained TrOCR We use the pre-trained [handwritten large TrOCR](#) as the baseline model. The model is trained on synthetic data and IAM handwritten dataset, and evaluated on IAM handwritten dataset. To make fair comparisons across dif-

ferent models or approaches in their paper, the output text predicted by TrOCR is filtered to include only characters from the 36-character set, which consists of 26 lowercase letters (a-z) and 10 digits (0-9).

Mention
generalisation

Fine-tuned TrOCR Another stronger baseline of ours is the fine-tuned TrOCR on GW and IAM handwritten datasets. We further include this baseline model for the following reasons:

First of all, fine-tuning TrOCR on a specific dataset can enhance its performance significantly. Comparing the composite model with both the original and fine-tuned TrOCR allows us to assess the added value of integrating CharBERT. In other words, demonstrating that the composite model outperforms not just the original but also a fine-tuned version of TrOCR helps justify this added complexity.

In addition, including both the original and fine-tuned versions of TrOCR as baselines provides a more comprehensive view of how the composite model performs across different stages of model optimization. This comparison is crucial for understanding whether the improvements come from fine-tuning, integrating CharBERT, or both.

Lastly, fine-tuning allows the model to adapt to the specific characteristics and distribution of your data, which might be significantly different from the data TrOCR was originally trained on. By evaluating against a fine-tuned baseline, we ensure that comparisons take into account the model's ability to handle data-specific challenges.

4.2 Composite Model Training and Evaluation Criteria

In this section, we delve into the training and evaluation process for the composite model. The section will cover key aspects such as the choice of optimizer, the loss function employed, and the hyperparameters set for the training. Additionally, the challenges commonly faced when training deep learning models, and insights into strategies and solutions to mitigate these issues will also be discussed.

4.2.1 Training Details

When training deep learning models, the selection of optimisation, loss computation techniques, and the hyperparameters settings play a crucial role in the efficiency and effectiveness of the training process. In this study, we use Adam as the optimiser

and cross-entropy for loss computation.

Optimiser Utilising Adam as the optimiser provides a sophisticated approach by adopting an adaptive learning rate, which is particularly adept at managing sparse gradients tasks such as NLP problems. Besides, due to its efficient computation of adaptive learning rates, Adam often leads to faster convergence on training data. This can significantly reduce the time and computational resources needed to train deep models, making the process more efficient.

For the training of the composite model, the learning rate has been meticulously set to $1e-5$. This setting ensures a controlled and gradual adaptation of the model parameters, facilitating a smooth convergence towards the local minima. Additionally, a weight decay parameter of $1e-5$ is employed to enhance the model's ability to navigate the optimization landscape efficiently. This careful calibration prevents the optimizer from overshooting the local minima, thereby promoting stability in the training process and improving the model's overall performance.

Loss Function OCR tasks involves predicting the probability distribution of possible characters for a given input image. It is a multi-class classification problem, with each character representing a unique class. Cross-entropy loss is naturally suited for multi-class settings. This makes it directly applicable to the task of classifying images into characters. In addition, unlike other loss functions that might focus solely on the accuracy of the classification, cross-entropy loss encourages the model not just to predict the correct class but to do so with high confidence. High-confidence wrong predictions are penalised more, encouraging the model to be cautious about making predictions when unsure, which is often the case with less frequent characters.

4.2.2 Training Challenges and Solutions

Training deep learning models involves navigating a series of common challenges that can significantly impact their performance and effectiveness. Common challenges include overfitting, vanishing/exploding gradients, high computational costs, and data quantity. The following discussion will delve into these challenges and the strategies used in this study to mitigate them.

Overfitting Overfitting occurs when a model learns the training data too well, capturing noise in the training set instead of learning the underlying patterns, which

results in poor generalisation to new, unseen data. Deep learning models, by their very nature, have a large number of parameters, allowing them to model intricate patterns and relationships in the data. However, it also means they have the capacity to memorise irrelevant details in the training data, leading to overfitting. Lack of regularisation techniques, or poorly chosen learning rate and batch size can easily lead to overfitting.

To mitigate overfitting, we implemented dropout layers between both linear and convolutional layers within the composite model architecture. Dropout serves as a form of regularization that, by temporarily dropping out units from the network, prevents the model from becoming overly dependent on any single element of the training data, thereby enhancing its generalization capabilities.

Additionally, we explored the impact of batch size on model training. Smaller batch sizes result in more noise during the gradient updates, which can have a regularizing effect. However, very small batch sizes can lead to extremely noisy gradient estimates, which might make training unstable or lead to convergence on suboptimal solutions. Through iterative testing, we determined that a batch size of 8 strikes an optimal balance, offering sufficient regularization to mitigate overfitting while maintaining stable and effective training dynamics.

Vanishing/Exploding Gradients Training deep models often encounters exploding or vanishing gradient problems due to their complex architectures and the long chains of computations involved. If the gradients are large (greater than 1), they can exponentially increase as they propagate back through the layers, leading to exploding gradients. Conversely, if the gradients are small (less than 1), they can exponentially decrease, leading to vanishing gradients.

Certain activation functions, like the sigmoid or tanh, squish a large input space into a small output range in a non-linear fashion. For inputs with large magnitudes, the gradients can be extremely small, leading to vanishing gradients. In addition, improper initialization of weights can exacerbate the exploding or vanishing gradient problems. For instance, large initial weights can lead to exploding gradients, while small initial weights can contribute to vanishing gradients.

To mitigate the vanishing/exploding gradients, we deploy strategies such as Xavier initialisation, Leaky ReLU activation function, gradient clipping, residual connection and batch normalization in the composite model architecture. The details of residual connection and batch normalisation is discussed in [Composite Model](#).

Make
sure they
are men-
tioned

High Computational Costs Deep models, especially those with many layers and parameters, require significant computational resources and time to train. Hardware accelerators like GPUs can reduce training times. In this study, a single A100 GPU, equipped with 80GB of RAM, was utilized to accelerate the training process. When applied to the GW dataset and the IAM dataset, the training durations were approximately 2 minutes per epoch and 30 minutes per epoch, respectively.

Data Quantity Deep models typically require large datasets to effectively learn and generalize due to their complex architectures and the vast number of parameters they contain. Training these models from scratch on limited data often leads to overfitting. To mitigate this issue, we employ transfer learning. In this study, we take pre-trained CharBERT and TrOCR model, which are developed on large and comprehensive datasets, and adapt it to our specific task.

4.2.3 Evaluation

The validation set serves the purpose of hyperparameter tuning and model selection. Meanwhile, the testing set is reserved for the final evaluation. For the GW dataset, we employ a 4-fold cross-validation approach. This method divides the dataset into four equally sized segments, using each in turn for testing while the remaining three serve as the training set. The final results for the GW dataset represent the average performance across these four folds.

In this study, we focus on word error rate (WER) and character error rate (CER) as our primary evaluation metrics. These metrics are critical for assessing the model's accuracy in recognizing and reproducing text, providing the model's performance in terms of both word-level and character-level precision.

4.3 CharBERT_{SMALL} Training and Evaluation Criteria

The training and evaluation of CharBERT_{SMALL} adhere closely to the methodology outlined by the original authors. The model is trained using the pre-training objectives, i.e., MLM and NLM, over the large text corpus. The training involves backpropagation and optimisation of weights to minimize the prediction error for both MLM and NLM tasks. While adhering to the [hyperparameters recommended by the original authors](#), we have adjusted the batch size to 16, up from the suggested size of 4, to accelerate the computation process. The training was executed on five

A100 GPUs, each equipped with 80GB of RAM, and was completed over the course of six days, encompassing five training epochs.

Finish
this

The results is evaluated

4.4 Composite Model Analysis

In this section, we conduct an in-depth analysis of composite neural network models, aiming to understand their performance, robustness, and adaptability across various tasks and datasets. Through a series of component analyses and benchmarking exercises, we investigate the efficacy of different model configurations and techniques, such as freezing specific layers, integrating dropout mechanisms, comparing tensor modules, and benchmarking against state-of-the-art models like GPT-4. Additionally, we delve into methods for validating model generalization and adaptability, exploring strategies for testing the model's performance on diverse datasets and assessing its ability to transfer learned patterns to new, unseen data environments. By examining the interplay between different components and techniques within composite models, we aim to provide comprehensive insights into their effectiveness and suitability for real-world applications.

4.4.1 Component Analysis and Benchmarking

Think
a better
name

In this section, we delve into various strategies aimed at enhancing the performance and robustness of our neural network models. To understand the contribution of specific components or features of a model to its overall performance, we conducted a comprehensive analysis. This analysis encompassed strategies such as 1) freezing specific layers; 2) integrating dropout mechanisms; 3) comparing combined tensor modules; 4) benchmarking against GPT-4. Each of these approaches was carefully implemented to isolate and understand the impact of various model elements on its efficiency. By comparing different combinations of tensor modules and loss functions, we gain valuable insights into the architecture's effectiveness and identify optimal configurations. Moreover, benchmarking our models against industry standards like GPT-4 provides a comprehensive evaluation of their capabilities, shedding light on areas of improvement and innovation.

Freezing Specific Layers When a layer is frozen, its weights remain unchanged during the training updates. This is often done to understand the contribution of

specific layers to the model's learning and performance. By freezing specific layers and training the remaining layers, we can assess how changes in certain parts of the model affect its overall performance. This helps identify which layers are critical for learning the task at hand and which might be redundant or less influential.

Integrating Dropout Mechanisms Dropout is a regularization technique used to prevent overfitting in neural networks. By integrating dropout mechanisms into different parts of the model, we can analyze its effect on reducing overfitting and improving generalization. This helps to understand the importance of each component in achieving a balance between model complexity and its ability to perform well on unseen data.

Comparing Combined Tensor Modules This involves experimenting with different tensor combination modules within the model. By comparing how different combinations of these modules affect performance, we can understand how the architecture and data flow within the model contribute to its success or limitations.

Integrating Batch Normalization Batch normalization is a technique used in deep learning to improve the stability and speed of training neural networks. It helps to stabilize the learning process by normalizing the inputs to each layer. This reduces the likelihood of vanishing or exploding gradients, which can occur when training deep neural networks, especially those with many layers. In addition, by normalizing the inputs, batch normalization allows for faster convergence during training. This means that the network can reach satisfactory performance levels with fewer training iterations, speeding up the overall training process.

Comparing Loss Functions Given the involvement of two models in our study, we explored various combinations of loss functions derived from the outputs of both models. This method not only allow us to assess the individual contributions of each model's output to the overall task, but also identify the most effective loss functions for the

Benchmarking Against GPT-4 Benchmarking against a model like GPT-4 provides a high standard for performance and innovation. It allows us to gauge how far your model has come in terms of understanding and generating text compared to leading models in the field. This comparison can highlight the strengths and

weaknesses of our approach, offering insights into what components or features are contributing most to its performance.

4.4.2 Validating Model Generalization and Adaptability

In this subsection, we explore methods to validate and enhance the robustness and adaptability of our machine learning model. By testing its generalization capabilities through exposure to diverse datasets and assessing its adaptability to new data environments, we aim to develop a model that performs reliably across a wide range of scenarios. Through a combination of dataset amalgamation and cross-training approaches, we investigate how our model can effectively learn from and adapt to varied data characteristics, ultimately ensuring its reliability and versatility in real-world applications.

Testing Model Generalization The purpose of this method is to enhance the model’s generalization capabilities. By exposing the model to a broader array of data characteristics and complexities during the training phase, it is better equipped to handle new and varied types of data that it might encounter in real-world applications. This approach aims to reduce the model’s potential biases towards any single dataset’s idiosyncrasies and promotes a more robust performance across different environments.

To test the model generalization capability, we combine the training datasets from both GW dataset and JH dataset and use this amalgamated dataset to train the model. This method is implemented by merging all the data points from both datasets into a single training set, which is then utilized to develop a model that is expected to capture and learn from the diversity and range of features presented by both datasets.

The advantages of this step are manifold. Firstly, it increases the robustness of the model by training it across a diverse set of data, which helps mitigate overfitting to specific dataset features. Secondly, this strategy enhances the model’s adaptability, making it capable of performing well in a wider range of scenarios, thereby improving its practical applicability. Finally, combining datasets can provide a richer and more comprehensive set of data for the model to learn from, which can lead to the discovery of more generalized patterns and relationships within the data, contributing to better overall performance in tasks requiring high levels of data interpretation and analysis. This step is crucial for developing a versatile model that not only performs well in controlled test conditions but also thrives in diverse operational

environments.

Testing Model Adapability The primary purpose of this method is to determine how well the model can adapt to different datasets and to identify any potential biases or limitations in its training. By testing the model on both the dataset it was trained on and an unrelated dataset, it's possible to gauge how effectively the model can transfer its learned patterns and predictions to new, unseen data, which is crucial for real-world applications.

This model evaluation strategy involves a cross-training and testing approach using two separate datasets, GW dataset and JH dataset. In this step, the model is first trained on GW dataset and then tested on both the test set of GW dataset and the test set of JH dataset. This process is then repeated by training the model on JH dataset and similarly testing it on the test sets of both datasets. This dual-training approach allows the model to be evaluated under two different learning conditions, which helps in assessing both its dataset-specific performance and its generalization capabilities across varied data types.

This method provides a comprehensive understanding of the model's performance and robustness, offering insights into how training on one dataset might influence performance on another. This method also highlights the model's ability to generalize, which is critical for deploying machine learning models in environments where they will encounter varied types of data. Additionally, this strategy helps in fine-tuning the model by identifying specific weaknesses or overfitting issues, guiding further refinements to enhance its accuracy and adaptability. Overall, Step 2 ensures that the model not only performs well in a controlled, homogeneous dataset environment but is also effective and reliable when faced with diverse and dynamic data conditions.

5 Results

5.1 Metric

The metrics used in this study are character error rate (CER) and word error rate (WER). They are two metrics commonly used to evaluate the performance of systems in tasks related to OCR. Both metrics measure how accurately the system transcribes or recognises text compared to the ground truth transcription. While CER operates on the character level, WER evaluates the accuracy at the word level.

The main difference between CER and WER is their granularity. CER is more fine-grained, making it more useful for evaluating tasks where character-level accuracy is crucial, such as OCR. On the other hand, WER is more suitable when inaccuracies at the word level significantly influence the interpretability of the transcribed text, such as speech recognition tasks.

The other difference between CER and WER is the sensitivity to errors. CER can be more sensitive to minor errors. In situations where altering a single character can drastically change a word's meaning, this sensitivity is crucial for assessing the system's performance accurately. For instance, changing "hat" to "hot" by substituting "a" with "o" changes the meaning completely. CER is good at capturing these subtle but critical errors, making it indispensable for tasks requiring high character-level precision. On the other hand, WER is more about capturing the errors affecting the listener's or reader's ability to understand the intended message at the word level. An illustrative example of this is the digitisation of a restaurant menu where the original text, "Chicken with Lemon Sauce" was misinterpreted by the OCR system as "Chicken with Lemon Source". This resulted in a WER of 20%. Although this single-word error does not significantly diminish the readability of the item, it introduces a notable semantic error. The term "Sauce" referring to a culinary liquid, while the term "Source" implies origin. This can easily mislead the readers and potentially affecting their understanding of the menu item. Thus, WER highlights how even minor word-level errors can have impacts on the content's semantic integrity.

Despite the differences, both CER and WER are critical for benchmarking the improving models in OCR, speech recognition and similar domains. They help developers identify shortcomings, compare different model architectures and track progress over time.

5.1.1 Character Error Rate (CER)

CER is often expressed as a percentage, representing the proportion of characters that are incorrectly predicted by the system. It is calculated based on the number of character level operations required to transform the system output into the ground truth text. These operations include insertions, deletions and substitutions of characters.

$$\text{CER} = \frac{I + D + S}{N} \quad (5.1)$$

where I , D and S are number of insertions, deletions and substitutions needed to match the ground truth respectively.

5.1.2 Word Error Rate (WER)

WER is calculated similarly to CER but at the word level. It also measures the number of insertions, deletions and substitutions at the word-level required to change the system's output into the ground truth.

$$\text{WER} = \frac{I + D + S}{N} \quad (5.2)$$

where I , D and S are number of insertions, deletions and substitutions needed to match the ground truth respectively.

5.2 Results

As outlined in the Analysis section in Chapter Experiment (referenced as Analysis), several analytical experiments were conducted. These include comparing the effects of freezing specific layers, integrating dropout mechanisms, evaluating different tensor module combinations, assessing various loss functions, and benchmarking against GPT-4.

5.2.1 Baseline Models

Dataset	Fine-Tune	WER	CER	File Name
GW	False	37.76	15.40	gw
GW	True			gw_ft
JH	False	91.31	58.57	iam
JH	True			iam_ft

5.2.2 Freezing Specific Layers

Dataset	Freeze	WER	CER	File Name
GW	None	17.13	10.17	gw_none
GW	TrOCR encoder	25.09	15.39	gw_encoder
GW	TrOCR decoder	17.46	10.49	gw_decoder
GW	TrOCR encoder, decoder	12.84	5.88	gw_01
GW	TrOCR decoder output projection layers			
JH	None	36.06	22.11	iam_none
JH	TrOCR encoder	34.74	22.32	iam_encoder
JH	TrOCR decoder	35.17	21.95	iam_decoder
JH	TrOCR encoder, decoder	35.34	20.72	
JH	TrOCR decoder output projection layers			

5.2.3 Integrating Dropout Mechanisms

Dataset	Dropout Rate	WER	CER	File Name
GW	-	13.11	5.78	gw_00
GW	0.1	12.84	5.88	gw_01
GW	0.2	12.88	6.26	gw_02
GW	0.5	13.80	6.39	gw_05
JH	-	35.62	22.17	iam_00
JH	0.1	36.38	22.92	iam_01
JH	0.2	35.13	22.76	iam_02
JH	0.5	35.30	22.77	iam_05

5.2.4 Comparing Combined Tensor Modules

Dataset	Combined Tensor Modules	WER	CER	File Name
GW	1	13.63	6.61	gw_c1
GW	2	30.66	18.85	gw_c2
GW	3	12.84	5.88	gw_01
GW	4	13.31	6.29	gw_c4
JH	1	34.52	20.61	iam_c1
JH	2	44.99	30.14	iam_c2
JH	3	36.38	22.92	iam_01
JH	4			iam_c4

5.2.5 Comparing Loss Functions

Dataset	Loss Function	WER	CER	File Name
GW	1	12.84	5.88	gw_01
GW	2			
GW	3			
JH	1			
JHM	2			
JH	3			

5.2.6 Benchmarking Against GPT-4

Model	WER	CER
TrOCR-CharBERT		
GPT-4 Post Correction		
GPT-4 Image Recognition		

5.3 Graphics

To include a graphic that appears in the list of figures, use the predefined fig command:

And then reference it as Figure 15 is easy.

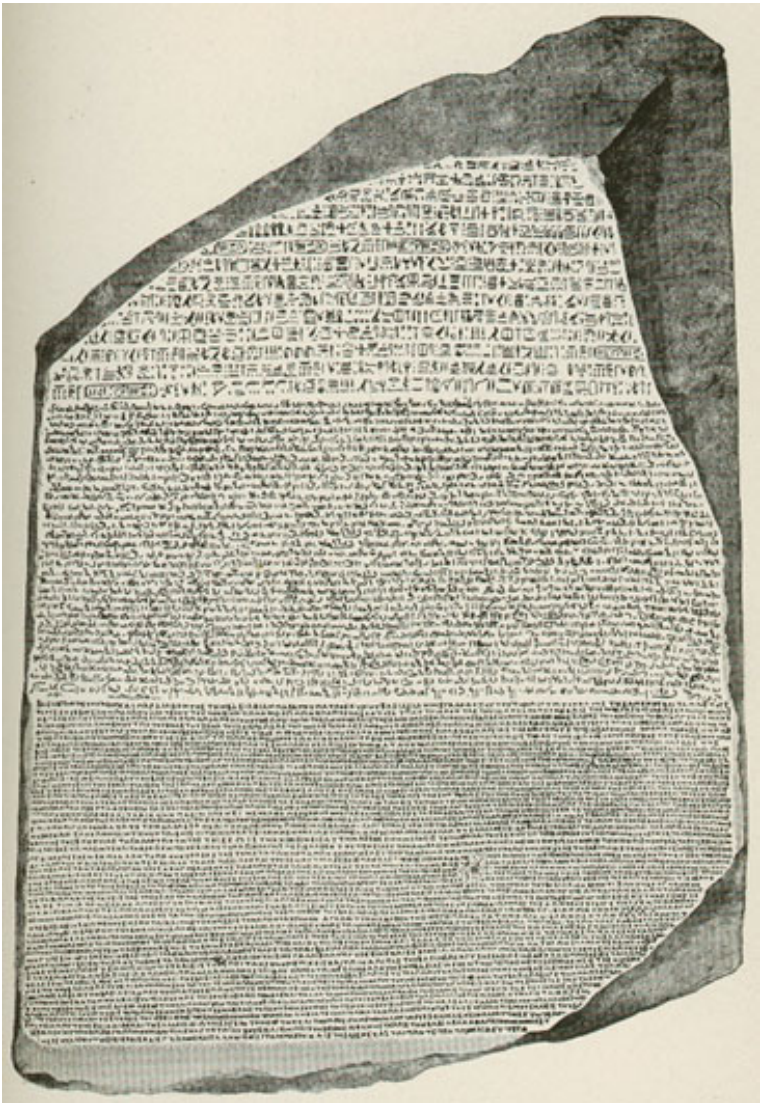


Figure 15: The Rosetta Stone

6 Discussion

In this project we have done so much.¹

We could show that ...

Future research is needed.

The show must go on.

¹Thanks to many people that helped me.

7 Conclusion

In this project we have done so much.¹

We could show that ...

7.1 Future Work

¹Thanks to many people that helped me.

Glossary

Of course there are plenty of glossaries out there! One (not too serious) example is the online MT glossary of Kevin Knight ² in which MT itself is defined as

techniques for allowing construction workers and architects from all over the world to communicate better with each other so they can get back to work on that really tall tower.

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

clitic A morpheme that has the syntactic characteristics of a word, but is phonologically and lexically bound to another word, for example *n't* in the word *hasn't*. Possessive forms can also be clitics, e.g. The dog's dinner. When **part-of-speech tagging** is carried out on a corpus, clitics are often separated from the word they are joined to.

²Machine Translation Glossary (Kevin Knight): <http://www.isi.edu/natural-language/people/dvl.html>

References

- C. Amrhein and S. Clematide. Supervised ocr error detection and correction using statistical and neural machine translation methods. *Journal for Language Technology and Computational Linguistics (JLCL)*, 33(1):49–76, 2018.
- H. Bao, L. Dong, S. Piao, and F. Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.
- B. Belay, T. Habtegebrial, M. Meshesha, M. Liwicki, G. Belay, and D. Stricker. Amharic ocr: an end-to-end learning. *Applied Sciences*, 10(3):1117, 2020.
- T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi, and F. Shafait. High-performance ocr for printed english and fraktur using lstm networks. In *2013 12th international conference on document analysis and recognition*, pages 683–687. IEEE, 2013.
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- R. Dey and F. M. Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- M. Ehrmann, A. Hamdi, E. L. Pontes, M. Romanello, and A. Doucet. Named entity recognition and classification in historical documents: A survey. *ACM Computing Surveys*, 56(2):1–47, 2023.
- A. Fischer, A. Keller, V. Frinken, and H. Bunke. Lexicon-free handwritten word spotting using character hmms. *Pattern recognition letters*, 33(7):934–942, 2012.

- A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.
- E. Grosicki and H. El-Abed. Icdar 2011-french handwriting recognition competition. In *2011 International Conference on Document Analysis and Recognition*, pages 1459–1463. IEEE, 2011.
- J. Huang, G. Pang, R. Kovvuri, M. Toh, K. J. Liang, P. Krishnan, X. Yin, and T. Hassner. A multiplexed network for end-to-end, multilingual ocr. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4547–4557, 2021.
- Z. Huang, K. Chen, J. He, X. Bai, D. Karatzas, S. Lu, and C. Jawahar. Icdar2019 competition on scanned receipt ocr and information extraction. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1516–1520. IEEE, 2019.
- P. Kahle, S. Colutto, G. Hackl, and G. Mühlberger. Transkribus-a service platform for transcription, recognition and retrieval of historical documents. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 4, pages 19–24. IEEE, 2017.
- L. Kang, P. Riba, M. Villegas, A. Fornés, and M. Rusiñol. Candidate fusion: Integrating language modelling into a sequence-to-sequence handwritten word recognition architecture. *Pattern Recognition*, 112:107790, 2021.
- K. Karthick, K. Ravindrakumar, R. Francis, and S. Ilankannan. Steps involved in text recognition and recent research in ocr; a study. *International Journal of Recent Technology and Engineering*, 8(1):2277–3878, 2019.
- K. Kettunen, H. Keskustalo, S. Kumpulainen, T. Pääkkönen, and J. Rautiainen. Ocr quality affects perceived usefulness of historical newspaper clippings—a user study. *arXiv preprint arXiv:2203.03557*, 2022.
- D. Klakow and J. Peters. Testing the correlation of word error rate and perplexity. *Speech Communication*, 38(1-2):19–28, 2002.
- P. Krishnan, K. Dutta, and C. Jawahar. Word spotting and recognition using deep embedding. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 1–6, 2018. doi: 10.1109/DAS.2018.70.

- D. Liu and J. Yu. Otsu method and k-means. In *2009 Ninth International conference on hybrid intelligent systems*, volume 1, pages 344–349. IEEE, 2009.
- Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- W. Ma, Y. Cui, C. Si, T. Liu, S. Wang, and G. Hu. Charbert: character-aware pre-trained language model. *arXiv preprint arXiv:2011.01513*, 2020.
- U.-V. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR’99 (Cat. No. PR00318)*, pages 705–708. IEEE, 1999.
- K. Mokhtar, S. S. Bukhari, and A. Dengel. Ocr error correction: State-of-the-art vs an nmt-based approach. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 429–434. IEEE, 2018.
- C. Neudecker, K. Baierer, M. Federbusch, M. Boenig, K.-M. Würzner, V. Hartmann, and E. Herrmann. Ocr-d: An end-to-end open source ocr framework for historical printed documents. In *Proceedings of the 3rd international conference on digital access to textual cultural heritage*, pages 53–58, 2019.
- T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet. Survey of post-ocr processing approaches. *ACM Computing Surveys (CSUR)*, 54(6):1–37, 2021.
- J. Philips and N. Tabrizi. Historical document processing: A survey of techniques, tools, and trends. *KDIR*, pages 341–349, 2020.
- R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
- A. Singh, K. Bacchuwar, and A. Bhasin. A survey of ocr applications. *International Journal of Machine Learning and Computing*, 2(3):314, 2012.
- R. Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.
- H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020a.
- W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33: 5776–5788, 2020b.
- Y.-Y. Wang, A. Acero, and C. Chelba. Is word error rate a good indicator for spoken language understanding accuracy. In *2003 IEEE workshop on automatic speech recognition and understanding (IEEE Cat. No. 03EX721)*, pages 577–582. IEEE, 2003.
- H. Zhang, D. Liu, and Z. Xiong. Cnn-based text image super-resolution tailored for ocr. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2017.

Curriculum vitae

Personal Information

Yung-Hsin Chen

yung-hsin.chen@uzh.ch

Education

2016-2020 Bachelor's degree in Physics
at National Tsing-Hua University (NTHU)

since 2020 Master's degree in Informatics
at University of Zurich (UZH)

Part-time Activities and Internships

2021 Delta Electronics Inc.
Data Scientist Intern

2022-2023 Swiss Re
Data Analyst

A Tables

Part of speech	POS type	number of labels	
		POS	in my corpus
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	DET	35	280
14	Total	35	280

Table 1: Some very large table in the appendix

B List of something

This appendix contains a list of things I used for my work.

- apples
 - export2someformat
- bananas
- oranges
 - bleu4orange
 - rouge2orange