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

List of Figures

List of Tables

List of Acronyms

OCR	Optical Character Recognition
LM	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	Feed Forward Neuron Network
Bi-GRU	Bidirectional GRU
GPU	Graphics Processing Unit
CER	Character Error Rate
WER	Word Error Rate

1 Introduction

1.1 Motivation

1.2 Research Questions

In this study, we aim to investigate and address the following key research questions.

1. 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. 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. Which layers should be frozen to results in better performance and why?
4. Does TrOCR decoder change its beam search output with the presence of the language model?

2 Related Work

2.1 TrOCR

OCR tasks typically involves text detection and text recognition. TrOCR is a Transformer based model which focuses on text recognition part of the OCR task, which convert images to texts. Therefore, the input should be sliced into line images, each with a single line of transcription on it. Unlike former progress in text recognition, TrOCR uses pre-trained image Transformer and text Transformer instead of CNN backbones and CTC. It eliminates the usage of external large LMs and can be extended for multilingual purposes by leveraging pre-trained 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 initialised by ViT-style models such as DeiT or BEiT, while the decoder initialisation uses BERT-style models such as RoBERTa or MiniLM. Both of them are Transformer encoder stuctures. Therefore, the missing encoder-decoder attention in TrOCR decoder is initialised randomly for training. Due to the fixed input length of Transformers, the image is first resized into 384×384, and then split into 16×16 patches before inputting the Transformer. Unlike CNN, Transformers don't have spatial information of the input data. Thus, positional encodings is added to the patches to preserve the spatial structures of the input images.

TrOCR is pre-trained on synthetic data and SROIE, IAM datasets sequentially. $\text{TrOCR}_{\text{LARGE}}$(total parameters=558M), initialised by $\text{BEiT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$, reached CER 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 are 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.

2.2 CharBERT

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

Pre-trained language models like BERT, 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 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 highlighs 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) Utilise 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 fuse them, ensuing a more robust

representation in case of typos. The second trick involves introducing character-level noise into words and 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 masked 10% of words and train the model to predict them. This enables CharBERT to the synthetic 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.

Although the performance of CharBERT does not exceed RoBERTa significantly, the model architecture of CharBERT is intriguing. Since models are usually better at the task they are pre-trained on, the NLM pre-training task makes CharBERT a potentially good corrector, which can be paired with the recognisor and compensate its language deficiencies. In addition, we can further pre-trained the NLM with common OCR mistakes instead of randomly introduced errors. This makes the corrector more familiar to OCR specific errors.

2.3 Candidate Fusion

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

The models utilised for demonstrating candidate fusion is 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 GRU, and the decoder is an undirectional multi-layerd 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 on handwritten datasets with the recogniser. This allows the LM to consider both its own knowledge and what the recogniser predicts when outputting the result text. The LM will adjust its predictions by taking into the recogniser's decision into account.

The model is being evaluated on IAM, GW and Rimes. 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 recogniser 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 recogniser 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.

Mention the paper with glyph embedding

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 Composite Model Architecture

The end-to-end composite model is composed of TrOCR and CharBERT. TrOCR does the text recognition task, while CharBERT serves as the corrector. The decoder input of TrOCR will go through CharBERT for correction before entering the decoder of TrOCR. By doing this, the TrOCR decoder will decode the input embedding corrected by CharBERT. In this section, I will first elaborate on the architectures of TrOCR and CharBERT, and how I fuse them together.

3.2.1 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 $\mathbf{W} = \{\mathbf{w}_1, ..., \mathbf{w}_i, ..., \mathbf{w}_D\}$ where each $\mathbf{w}_i$ stands for a token tokenised by TrOCR tokeniser and D signifies the length of the sequence.

3.2.2 Recogniser - TrOCR

TrOCR is composed of an image encoder and a text decoder. The image encoder takes pixel values of an image as the input, and the encoder output will then be 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, I will elaborate on how the inputs go through the decoder.

Decoder Inputs and Labels The text decoder takes encoder output, decoder input ids and padding attention mask[1] as input, and output the generated text. The decoder output will be compared to the label tensor for loss calculation. The loss will then be used for updating the gradients of the parameters during training.

The encoder output is a set of features extracted from the pixel values. These features are the transformed representation of the input image, not only capturing the patterns within the image, but also understanding the arrangements and relations of each elements in the image.

The encoder's output represents a set of high-level features extracted from the input image. These features are essentially a transformed representation of the input image, capturing various aspects and patterns that are relevant for recognizing the text within the image.

Decoder input ids is a tensor of token ids converted from the label texts. The token ids includes three special tokens, which are <bos>, <eos> and padding. The id, <bos>, is represented by the number 0 and indicates the start of the text. The id, <eos>, is denoted by the number 2 and represents the end of the text. As for

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

padding, it is a series of number 1 appended after ¡eos¿ to make sure that all the input ids have the same length. The decoder input ids can be generated automatically by the TrOCR tokeniser. By default, the generated decoder input ids have <bos> in the beginning, and <eos>, paddings at the end. This can be used during training to simulate the process of inference[2].

The other input of the decoder, padding attention mask, controls the information that should be ignored by TrOCR. The padding attention mask consists of 0s and 1s. The 0s covers the padding parts of the decoder input ids to prevent the model from focusing on them.

The label tensor for calculating the loss is decoder input ids without the <bos> token. And since the TrOCR model is inherited from VisionEncoderDecoderModel, the class requires the label paddings to be -100. Hence, the padding token of the label tensor will be replaced by -100 instead of the 1s.

Inside the Decoder Let's first talk about the original decoder before adapting it to fit CharBERT. The decoder receives the decoder input ids, its padding attention mask and the encoder output. First, the decoder input ids converted into decoder embeddings. Next, the decoder embeddings is added with the positional encoding to help keep the spatial information before going through decoder stacks, including masked multi-head attention, multi-head attention and the FFNN. The padding attention mask will be applied to both masked multi-head attention and multi-head attention to prevent TrOCR from attending the padding tokens. Finally, the output of the decoder stacks will go through a linear layer and a softmax layer to generate an output sequence.

3.2.3 Corrector - CharBERT

CharBERT has a dual-channel architecture, which are the token and the character channels. The token channel has the same structure as BERT or RoBERTa, depending on the initialisation. In this study, we will focus on the $\text{CharBERT}_{\text{RoBERTa}}$. From this point forward in the document, any mention of "CharBERT" will specifically refer to $\text{CharBERT}_{\text{RoBERTa}}$, unless explicitly stated otherwise. Similar to BERT and RoBERTa, CharbERT takes texts as input, but instead of outputting

[2]During inference, the decoder output from the previous step will be the new decoder input. However, the decoder has no other context at the first step since no output sequence has been generated yet. Thus, the very first step of the decoder input is just this <bos> token. The decoder processes this initial input and generates a decoder outpu, which is being used to update the decoder input.

a single embedding, CharBERT outputs a token level embeddings and a character level embeddings. Since the model architecture of the token channel is identical to BERT/RoBERTa , the subsequent contents will focus on the character channel and heterogeneous interaction.

Character Channel & Heterogeneous Interaction The character channel first splits the input text into characters, convert them into IDs by looking up the dictionary. Next, it embeds the IDs, and then apply the bidirectional GRU (Bi-GRU) layer to generate the output embeddings. The output embeddings from the character and token level will then go through the transformer and the heterogeneous interaction. The heterogeneous interaction is consists of two parts, fusion and divide. The fusion allows both embeddings to enrich each other by FFNN and CNN, while the divide part ensures that both embeddings keep their unique features by a FFNN and a residual connection. The residual connection helps retain the respective information from the two embeddings. CharBERT then repeat the transformer and heterogeneous interaction to capture more features and information underlying the input texts.

3.2.4 Composite Model

The composite model is designed to combine TrOCR and CharBERT. The idea is that, during the inference phase, the decoder output is recycled as its input in a feedback loop. Before this recycled input is fed back into the decoder, it has to undergo correction and refinement by CharBERT. Thus, CharBERT will be integrated between the decoder input and the decoder stacks, ensuring that the input to the decoder is optimised on each iteration of the process. The problems with this integration are: 1) TrOCR decoder takes token IDs as input but CharBERT outputs are embeddings; 2) CharBERT takes texts as input but the TrOCR decoder input is a tensor; 3) TrOCR embedding representations do not match CharBERT embedding representations; 4) TrOCR decoder input is a single tensor but CharBERT has dual-channel outputs. The following contents will be discussing the details and approaches to address these two problems.

Adapted TrOCR Before delving into the potential solution to the first problem, it is essential to thoroughly examine the issue. The TrOCR decoder is designed to take token IDs as input, which it then maps to embeddings. These embeddings are then added with the positional encoding before being passed into the Transformer

decoder. However, the challenge arises due to the outputs from CharBERT after the correction being embeddings, not token IDs. This presents a compatibility issue. If we were to solve this problem by converting the CharBERT representations into token IDs before inputting TrOCR decoder, another issue emerges. Employing token IDs as intermediaries between model components is problematic, as token IDs are inherently integers. During the training process, the model weights are updated as floating-point numbers. Constraining these weights to integer values is not feasible. While rounding up the updated weights can technically convert floats to integers, but the underlying meaning of rounding it will be questionable. The rounding would likely distort the model's learning process and could be meaningless for the task. Therefore, the only reasonable way to solve this issue is to modify TrOCR decoder so that it takes embeddings as input.

There is a straightforward solution. By repositioning the embedding layer from the TrOCR decoder to precede CharBERT, token IDs will be initially converted to TrOCR embeddings, which will then be input into CharBERT for correction. Consequently, the adapted TrOCR can now accept embeddings directly instead of token IDs. This modification ensures that the outputs from CharBERT integrate with the TrOCR decoder and that there will be no IDs between model components.

Adapted CharBERT The second problem involves the mismatching data types of the input and the output. The TrOCR decoder input is an embedding according to the modification made in Adapted TrOCR. This embedding should be fed into CharBERT for correction. Unfortunately, CharBERT takes only texts as input. Thus, creating an adapted CharBERT model is necessary. The adapted CharBERT is modified so that it takes token and character embeddings as inputs. These two embeddings will then go through the token and character channels. To be specific, the adapted CharBERT will no longer take texts as input, convert them into IDs and embed them. The embeddings will be provided as the input directly.

Tensor Transform The third problem is more complex. The TrOCR decoder input is based on embeddings unique to TrOCR. Even for the same text, the embedding representations from TrOCR and CharBERT are markedly different, not to mention CharBERT has dual-channel embeddings. Not only do both models have different representations for the same text, the embedding dimensions are also incompatible. Therefore, even though CharBERT can take embeddings as inputs, the TrOCR decoder input still cannot be fed into CharBERT directly. To address this problem, an architecture consists of CNN and FFNN is employed. This approach serves to adjust

the dimension of TrOCR decoder input to align with dimensions of CharBERT token and character embeddings. Additionally, this dimensional transformation allows the model to match the representations between TrOCR and CharBERT effectively. To better illustrate the architecture of the tensor transform, the dimensions will be noted after each input/output between brackets, e.g., (batch size, embedding size).

The goal of the tensor transform is to convert the TrOCR decoder input size (batch size, 512, 1024) into the CharBERT token embeddings size (batch size, 510, 768) and CharBERT character embeddings size (batch size, 3060, 256), where the second dimension is the sequence length and the third dimension is the embedding size. In this tensor transform architecture, the transformation is separated into two stages. In the first stage, the decoder input first goes through a series of CNN layers interspersed with LeakyReLU and batch normalisation steps to adjust the sequence dimension (dim=1). Then, for the second stage, the output from the first stage goes through FFNN layers with LeakyReLU in between 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 applying the CNN layers, the dimension of the decoder input becomes (batch size, 510, 1024). The sequence dimension (dim=1) has become the desired dimension for CharBERT token embedding. The purpose of the CNN layers is to either expand or contract the sequence length, offering the advantage of preserving spatial information. This is the primary reason for selecting CNN instead of FFNN for adjusting the sequence dimensions. Moreover, it is worth noting that there are batch normalisation in between the convolutional layers. The batch normalisations are there to stabilise the deep model and to help maintaining healthier gradients with the presense of the activation functions[2]. Next, we enter the second stage of

[2]With the presense of activation functions in deep networks, gradient can easily explode or vanish. Batch normalisation helps in maintaining a healthier gradient flow in the network, which can improve the efficiency of backpropagation and thence the learning process.

the tensor transformation, which utilise the 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 is the CharBERT token embedding $\mathbf{T} = \{\mathbf{t}_1, ..., \mathbf{t}_n, ..., \mathbf{t}_N\}$, where N is the token sequence length with tensor size (batch size, 510, 768).

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

The Tensor Transform layer aims to convert the TrOCR decoder input into CharBERT token embedding and character embedding. The CNN and FFNN layers not only match the dimensions between the tensors, but also learn to map the contextual information in TrOCR embedding to CharBERT embeddings.

Tensor Combine The fourth problem arises due to the fact that CharBERT produces two tensors - token and character representations, whereas TrOCR decoder input is a single tensor. Consequently, the solution would be to combine the two output tensors from CharBERT into a single tensor. In addition to the outputs from CharBERT, we can also add a residual connection from the original TrOCR decoder embedding to stabilise the deep model. The residual connection here can encourage the reuse of features from the original TrOCR decoder embedding and prevent gradient vanishing.

Among the three tensors, two of them comes from CharBERT. Therefore, the representation is not understood by TrOCR decoder and the dimensions do not match. The two of them should first undergo Tensor Transform before the combination.

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

where d_D = (batch size, 512, 1024); d_N = (batch size, 510, 768); d_M = (batch size, 3060, 256).

After performing the tensor transform module, three tensors will all be of size (batch size, 512, 1024), which is the same as the original TrOCR decoder input size. They

can then be combined and fed into the TrOCR decoder stack. In this study, we examine four tensor combine module architectures: 1) simply adding all of them; 2) mean pooling; 3) using linear layers as the attention net; 4) using convolutional layers as the attention net.

Tensor Combine 1: Adding The first tensor transform module is simply aggregating three tensors into a single tensor, operates by performing an element-wise addition of the three input tensors. Given the TrOCR decoder embeddings $\mathbf{E}$, the CharBERT transformed token representation $\mathbf{T}'$ and the CharBERT transformed character representation $\mathbf{C}'$, each contributing equally to the formation of the combined tensor (each input tensor's information is weighted equally), the operation can be mathematically expressed as:

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

where $i = 1, 2, ...,$ batch size; $j = 1, 2, ..., 512$; $k = 1, 2, ..., 1024$.

Tensor Combine 2: Mean Pooling The second tensor combine module is designed to dynamically allocate attention weights to each word across the three input tensors. The weights are obtained by performing max pooling along the embedding axis of three tensors stacked together. After obtaining the weights, it will be applied to the stacked embeddings. Note that the three tensors are all feature-wise normalised before stacking. The feature-wise normalising helps the model converge faster and achieve better generalisation.

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

Tensor Combine 3: Linear Layers as Attention Net The third tensor combine module further extend the second module by leveraging linear layers as the attention net. Instead of simply perfoming mean pooling, it uses linear layers with activation functions in between to capture more relevant information upon deciding

the weights. The process can be expressed as:

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

For $i = 1, 2, ..., 512$: Conv Attention Net

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

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

Tensor Combine 4: Convolutional Layers as Attention Net The fourth tensor combine module replaces the linear layers with convolutional layers. The equations below shows the attention net, which are different from the one in Tensor Transform - Linear Layers as Attention Net.

For $i = 1, 2, ..., 512$: Linear Attention Net

$$\begin{aligned}
\mathbf{S'} &= \mathbf{S}[:, :, i, :] \in \mathbb{R}^{(\text{batch size, 3, 1024})} \\
\mathbf{S'}_1 &= \text{LeakyReLU}(\mathbf{b}_7 + \mathbf{W}_7 \cdot \mathbf{S'}) \\
\mathbf{S'}_2 &= \text{LeakyReLU}(\mathbf{b}_8 + \mathbf{W}_8 \cdot \mathbf{S'}_1) \\
\mathbf{S'}_3 &= \mathbf{b}_9 + \mathbf{W}_9 * \mathbf{S'}_2 \\
\mathbf{P}_i &= \text{softmax}\Big(\frac{1}{1024} \sum_{j=1}^{1024} \mathbf{S'}_{3,(a,b,j)}\Big) \in \mathbb{R}^{(\text{batch size, 3})}
\end{aligned} \tag{3.9}$$

Change to conv version

3.3 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.

3.3.1 Get $\mathcal{P}_{ij}$

To obtain $\mathcal{P}_{ij}$, it is essential first to calculate the frequency of each character misrecognised by the recognizer (TrOCR), termed as *misrecognised frequency*. This process involves initially using the plain TrOCR model to process the GW and IAM datasets. Upon obtaining the text outputs generated by TrOCR, a comparison is made with the corresponding text labels to obtain the frequencies of the characters that were misrecognised.

To match the generated outputs from the labels, we use the "Bio" package in Python, refering to Biopython. It is a set of tools for biological computation. Biopython is primarily designed to work with data from bioinformatics, such as sequences of DNA, RNA, and proteins. It provides capabilities for DNA and protein sequence analysis and alignments.

While Biopython is not directly designed for processing natural language texts, its tools and principles can still be indirectly applied to text data. Biopython's capabilities for sequence matching and regular expressions can inspire similar applications in text analysis. For example, in our scenario, Biopython's framework can be leveraged to perform character-by-character matching within text strings, identifying and annotating any discrepancies encountered. This approach exemplifies how Biopython's core functions can be creatively repurposed beyond their intended biological applications, offering valuable insights and techniques for handling and analyzing text data.

add example

3.3.2 Training CharBERT$_{\mathcal{P}_{ij}}$

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 $\text{CharBERT}_{\text{SMALL}}$. 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 $\text{CharBERT}_{\text{SMALL}}$. 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.

4.1 Data Collection and Processing

This section is dedicated to detailing the data collection process for OCR task training and $\text{CharBERT}_{\text{SMALL}}$ 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.

4.1.1 Data for OCR

We focus on the performances of handwritten datasets on the composite model. The data used in this study is George Washington (GW) handwritten dataset and IAM handwritten dataset. They serve as a valuable benchmark 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 IAM datasets.

Add data downloading link

The GW dataset is a collection of historical letters and dairies handwritten by George Washington and this secretaries in 1755. The dataset is often used in research focused on recognising historical handwriting, which poses unique challenges due to the use of archaic words and phrases, and the degradation of materials over time.

The IAM dataset is a more comtemporary collection of English handwriting samples. It contains forms written by hundreds of writers, and is thus renowned for its diversity in handwriting styles, the variability of written content.

While the George Washington Handwritten Dataset provides a niche focus on historical documents, making it ideal for projects related to historical document analysis and preservation, the IAM Handwriting Database offers a broad spectrum of modern handwriting samples, making it suitable for a wide range of handwriting recognition applications. Both datasets have played crucial roles in advancing the field of handwriting recognition.

Add data sample here

Transcription Ground Truth Processing The transcription ground truths of the IAM 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 data processing purposes, we modify the transcription texts from the GW dataset by removing the hyphens, replacing vertical bars and special characters representing punctuations with spaces and their respective punctuation marks. This process ensures that the transcription texts are standardized and easily readable.

Add text example

4.1.2 Data for Training CharBERT$_{\text{SMALL}}$

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$_{\text{SMALL}}$. This variant was trained on a significantly

smaller dataset, specifically 1.13GB of English Wikipedia data, from which sentences were randomly sampled. This adaptation allows us to evaluate the performance of CharBERT under more restricted conditions, ensuring our experiments are feasible within our available resources.

4.2 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 different 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).

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. Camparing 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.

4.3 Composite Model Training and Evaluation Criteria

In this section, we will be exploring the training and evaluation process of the composite model. Details including optimiser, loss function, hyperparameters, and the challenges encountered training deep models and how to mitigate it will also be discussed.

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.

4.3.1 Training Details

When training deep learning models, the selection of optimisation, loss computation techniques, and the hyperparameters settings play a crucial role in the effeiciency 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$-$5. This settingn ensures a controlled and gradual adaptation of the model parameters, facilitating a smooth convergence towards the local minima. Additionally, a weight decay parameter of 1e$-$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.

4.3.2 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 noisein 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 normalisation in the composite model architecture. The details of residual connection and batch normalisation is discussed in Composite Model.

Make sure they are mentioned

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.

4.3.3 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.

4.4 CharBERT$_{\text{SMALL}}$ Training and Evaluation Criteria

The training and evaluation of CharBERT$_{\text{SMALL}}$ 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

4.5 Analysis

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.

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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 two models to work together.

4.5.5 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.

5 Results

5.1 Metric

The metrices 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 WER 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 illustratice 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 precentage, 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.

$$\text{CER} = \frac{I + D + S}{N} \tag{5.1}$$

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 (CER)

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

$$\text{WER} = \frac{I + D + S}{N} \tag{5.2}$$

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

5.2 Results

5.2.1 Freezing Specific Layers

5.2.2 Integrating Dropout Mechanisms

5.2.3 Comparing Combined Tensor Modules

5.2.4 Comparing Loss Functions

5.2.5 Benchmarking Against GPT-4

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 1 is easy.

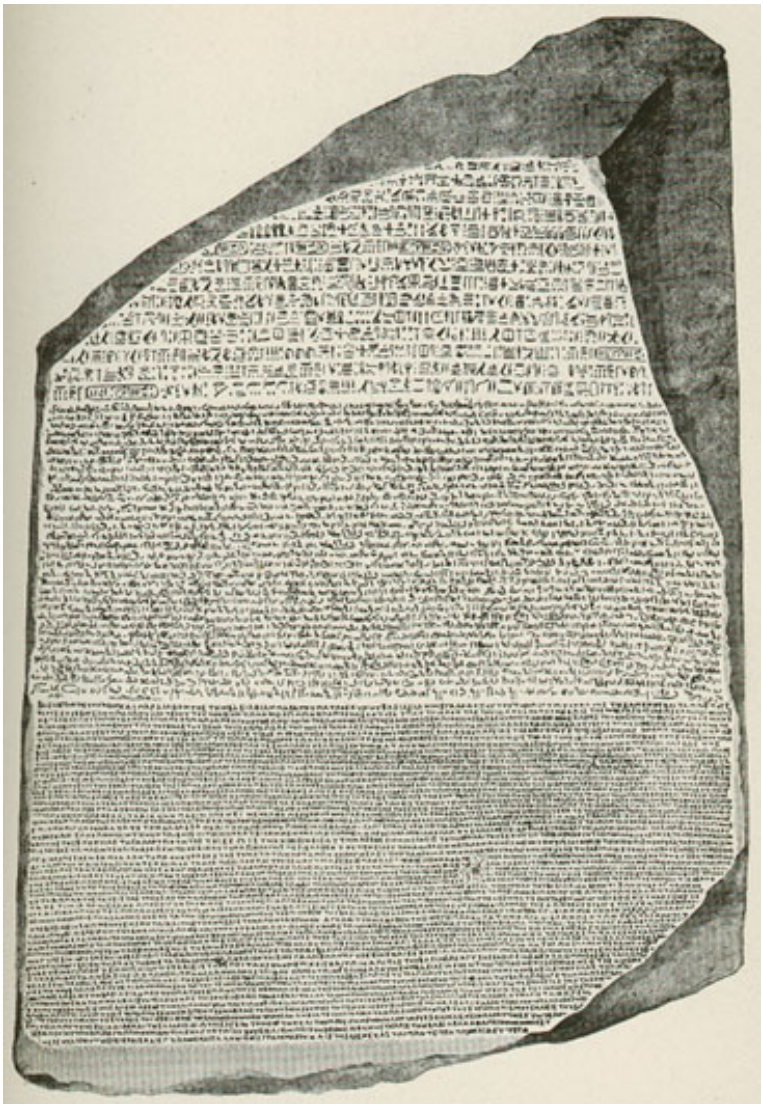

Figure 1: The Rosetta Stone

6 Discussion

In this project we have done so much.[1]

We could show that ...

Future research is needed.
The show must go on.

[1] Thanks 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

[1] Thanks 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.

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

References

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.

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.

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.

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	POS	number of labels 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	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`