Optical Character Recognition on Code Screenshots – Training a Transformer based OCR Engine

Real-World Challenges in Data Science and Analytics (Master Seminar)

FZI Forschungszentrum Informatik

Version 1.0

Date: February 10, 2023

Authors:
Nikolas Heß (2055101)
Alessio Negrini (2106547)

Abstract

Optical Character Recognition (OCR) defines the task of extracting machine readable text from images, such as digital text screenshots or analog printed documents. Traditional OCR models usually consist of convolutional neuronal networks (CNN) to process images and recurrent neuronal networks (RNN) for text generation. In recent years, however, Transformer-based OCR models have emerged as a viable alternative for the OCR task, in many cases even outperforming their CNN-RNN-based counterparts. In this work, a pre-trained Transformer-based OCR model (TrOCR) based on the work of Li et al. [19] was fine-tuned to successfully extract text from code screenshots. Following a brief introduction to the OCR task and the TrOCR model, our methodology and results regarding the pre-processing of our given dataset as well as the model training and evaluation is presented in this paper. The final model was fine-tuned on 450,000 labeled code lines for training and achieved a Character Error Rate (CER) mean of 0.20. The model was also evaluated using the Levenshtein Distance, achieving a mean of 0.92. The TrOCR model performs well on code line snippets, but learns noise from the IDE, such as arrows and indentation marks (pipes), causing a significant increase in the CER metric. The text, however, was predicted very accurately. By excluding such noisy samples from the test set, a mean CER of 0.11 was achieved. The final model managed to outperform the Tesseract model as well as other trained versions of the TrOCR model by Li et al. [19], which were used as performance benchmarks.

Keywords

Optical Character Recognition, Image2Text, Transformer, Code Screenshots, Computer Vision

Contents

List of Figures

List of Tables

1 Introduction

Optical Character Recognition (OCR) is a technology that enables the recognition and extraction of written text from an image or a scanned document. OCR has become increasingly important in recent years due to the growing digitization of information and the increasing volume of written content available in digital form. With the help of OCR, it is now possible to convert scanned documents, books, and other printed materials into machine-readable text, which can be easily stored, processed, and analyzed by computers. This makes OCR an essential tool for a wide range of industries, including libraries, archives, and businesses, as it enables them to effectively manage and utilize vast amounts of written information.

One of the earliest recognized OCR systems was created by Ray Kurzweil in 1974. It was a machine – originally designed for blind people – that could recognize printed text in just about any font and read it out loud [8]. In 1980, his company was sold to Xerox, which then continued to research OCR. Significant advancement were made in the early 1990s, when OCR models were implemented to digitize historic newspapers [8]. Previously, the only way to digitize documents was to manually retype the text – a tedious and error-prone process. Modern technologies can automate complex document-processing procedures and achieve almost flawless OCR accuracy [8]. There are numerous open source OCR services that are freely available to the public. Two well-known examples are Google Cloud Vision OCR and Tesseract [1].

OCR's most well-known application is making printed documents machine-readable. OCR allows for the transfer of written information from analog to digital form, making it one of the key forces behind digitization. When examining contemporary big data methodologies or machine learning models, the effects of these developments on our digital environment become more clear. Recent innovations, like the well-known ChatGPT model, were only made possible because the vast quantities of data they were trained on were readily available in digital form. Other important applications of OCR include assistance tools for visually impaired people (e.g. [2, 10]), automating and improving big data processing workflows (e.g. [13, 27, 29, 30]), improving autonomous control systems such as robots or self-driving cars (e.g. [24, 28, 38]) and many more.

Since Transformer-based models have produced groundbreaking results in recent years, it is only natural that the scientific community has made substantial advances in the field of transformer-based OCR techniques as well. Similar to many other related domains, the state-of-the-art appears to be shifting from previously successful hybrid models based on artificial neural networks using convolution and recurrence techniques to specific transformer-based architectures.

Therefore, in this elaboration, a pre-trained Transformer-based OCR model (TrOCR) based on the work of Li et al. [19] was implemented on a dataset of code screenshots. Li et al. fine-tuned and evaluated different variations of the TrOCR model on the downstream tasks of printed, handwritten, and scene text recognition. In this paper, on the other hand, their first-stage TrOCR model, which had been pre-trained on hundreds of millions of text-lines from online document images, was fine-tuned and evaluated on a dataset of code screenshots containing a variety different programming languages, fonts and IDE themes. The focus in this work lies on the text recognition task, meaning the translation of image-text to machine-readable text. Therefore, the text detection task, i.e. locating the text within the image, was skipped by using available information about the location of each code-line contained in so-called annotation files. Figure 1 illustrates the process of extracting code-lines from an IDE-screenshot containing several lines of code, skipping the data detection task using available annotation data and conducting the data recognition task by applying the TrOCR model from Li et al. [19]. Finally, predictions made by the model are evaluated using the *character error rate* (CER) and *Levenshtein distance*.

Section 2 provides a short overview of pertinent related work. In section 3, the Transformer model is introduced as the basis of the TrOCR model and the most important findings from the work of Li et al. are outlined briefly. Sections 4 and 5 cover our methodology and results for the pre-processing of our given dataset as well as the training and evaluation of the model. Lastly, our most important findings are summed up in section 6.

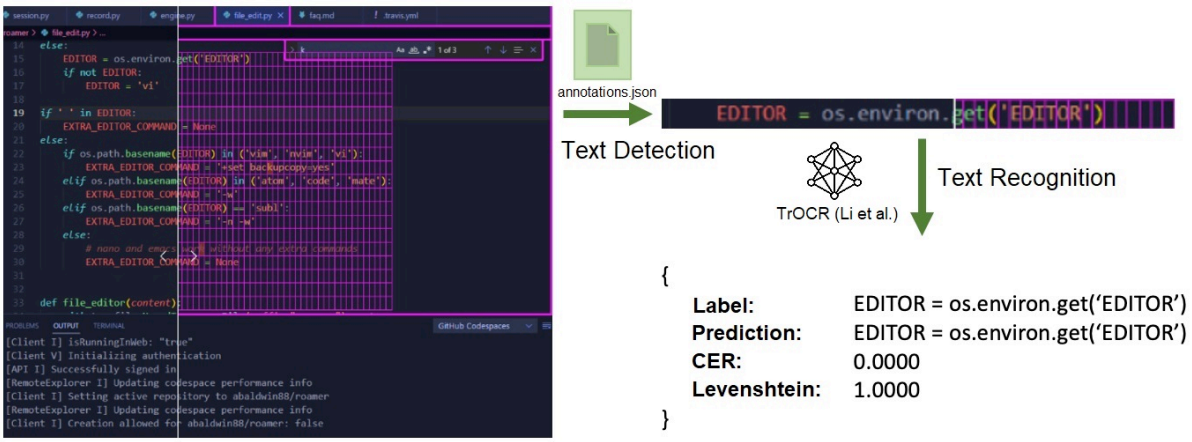

Figure 1: Example of the data used to train the model of this paper: A screenshot of an IDE, where lines are cropped from using annotation-files that contained the text locations within the image. These code line snippets were then used to train the model. To evaluate the model's performance, unseen code snippets from the test set were used to calculate evaluation metrics such as CER and Levenshtein distance. *(Author's Illustration)*

2 Related Work

According to Li et al. [19], OCR approaches are usually divided into two sub-tasks: Text detection and text recognition. Text detection can be regarded as a regular object detection problem, where the goal is to localize all text-blocks within an image. Therefore, to solve this task, traditional object detection models such as YOLOv7 [34] or DBNet [22] can be utilized. Text recognition, on the other hand, refers to the process of identifying the text-content of an image and transcribing it into machine-readable text. As the goal in this paper is to implement the TrOCR model [19] in order to generate text from code screenshots where the location of text-blocks are assumed to be known, the focus will be on the text recognition task.

The text recognition task can be regarded as an encoder-decoder problem [19]. In this context, the encoder is responsible for taking in an image of text and encoding it into a hidden feature representation that captures the important information in the image. The decoder then processes this encoded representation and generates the final output, which is the recognized text. Two models that are commonly used for this task are *convolutional neural networks* (CNNs) and *recurrent neural networks* (RNNs). CNNs are used to extract features from the image and RNNs are used to process the sequence of features and predict the characters. The combination of these two types of networks is referred to as CNN-RNN model, which is a popular choice for text recognition tasks (see e.g. [11, 17, 25, 31]). To improve the accuracy and robustness of the CNN-RNN model, a technique called *connectionist temporal classification* (CTC) [9] is usually applied. While this technique does generally improve performance, it represents an extra post-processing step and introduces the need for another external language model in addition to the CNN-RNN model.

In recent years, a series of Transformer-based approaches has emerged as an alternative to the already successful CNN-RNN hybrid models. These novel approaches leverage a so-called *self-attention* mechanism [33] and managed to achieve state-of-the-art results in various natural language processing (NLP) tasks, such as language translation, text summarizing and question answering. Transformer models are capable of processing entire input sequences simultaneously, making use of positional embeddings and the self-attention mechanism to parallelize the training process. Therefore – unlike RNNs – Transformer models allow for efficient training on massive datasets in a relatively short period of time and are used to implement large scale models as well as pre-trained systems that can be shared and fine-tuned for specific downstream tasks. Some recent Transformer-based models in the NLP domain are BERT [6] and the popular ChatGPT model, which is based on GPT-3 [4]. The versatility of the Transformer architecture has led to its implementation in a wide range of domains beyond natural language processing over the recent years, such as computer vision (CV) (e.g. [5, 15, 36]) and time series forecasting (TSF) (e.g. [21, 26, 37]).

As text recognition is an interdisciplinary task that combines CV and NLP steps to extract machine-readable text from input images, the potential of Transformers for this task has already been recognized by the scientific community (e.g. [7, 16, 19]). One recent example is the Transformer-based OCR (TrOCR) model by Li et al. [19]. In their approach, they replaced the CNN-RNN framework with two separate Transformer models: An image Transformer to replace the CNN and a text Transformer to replace the RNN. By leveraging self-supervised pre-training techniques (see e.g. [3]), they combined pre-trained Transformer models to form an end-to-end framework for text recognition. Due to the general features that were already learned by each Transformer during the pre-training on synthetic large-scale datasets, the resulting framework could then be fine-tuned to specific downstream tasks without the need of any complex pre-/post-processing steps or external language models. While the authors demonstrated this for the downstream tasks of text recognition on printed, handwritten and scene text-image data, the focus in this paper is the application and fine-tuning of TrOCR for text recognition on code screenshots.

Even though there are numerous other Transformer-based approaches addressing the text recognition task, the shown effectiveness and simplicity of implementation make the TrOCR model a straightforward choice for the text recognition task on code screenshots. For a detailed comparison of the TrOCR model to other similar approaches, please refer to the work of Li et al. [19].

3 Theoretical Foundation

In this section, an introduction to the Transformer architecture as the basis of the TrOCR model is given. In addition, the TrOCR approach is summarized and the central components of the model are explored. The following definitions and illustrations are mainly based on the work of Vaswani et al. [33] regarding Transformers and that of Li et al. [19] regarding the TrOCR approach. As the goal in this section is to only give a high-level overview, please refer to the work of Vaswani et al. [33] and Li et al. [19] for specific details.

3.1 Transformers

A Transformer is a type of neural network used for sequence transduction, that is based on an encoder-decoder structure, where the input sequence is first passed through an encoder to produce a hidden representation of the input, and then passed through a decoder to generate the output sequence. Transformer models have several key advantages over the CNN-RNN model mentioned in the related work section. Firstly, Transformers are able to effectively handle long-term dependencies in sequential data, a task that RNNs have difficulty with [12]. This is achieved through the use of self-attention mechanisms, which allow the model to weigh the importance of different parts of the input sequence when making predictions. One key advantage of utilizing the self-attention mechanism is that it does not depend on previous computations or elements in the processed sequence and considers all elements of the input sequence simultaneously. Therefore, Transformers are able to perform parallel computation, making them faster and more efficient than CNNs or RNNs. Furthermore, the Transformer architecture uses a so-called *multi-head attention* technique by splitting the attention mechanism into multiple "heads", allowing the model to attend to different parts of the input sequence simultaneously. In an NLP task, for example, one head may focus on specific words of an input sentence, while another head may focus on the overall structure of the sentence. This improves the model's ability to understand the underlying relationships between elements in a sequence and effectively handle sequences with long-term dependencies.

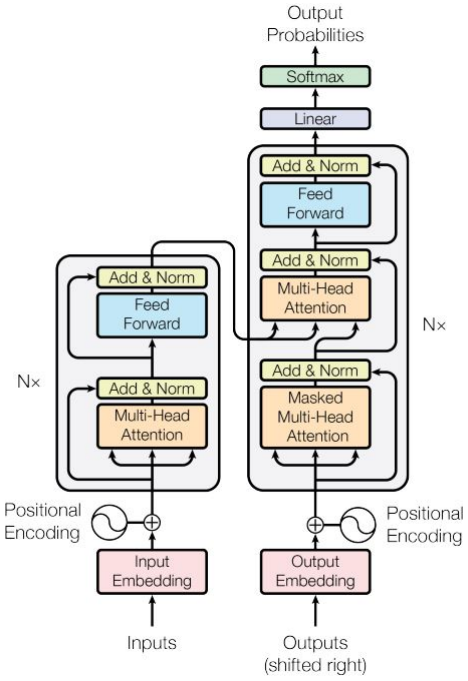

Figure 2: The Transformer architecture [33].

Figure 2 shows the encoder-decoder structure of the transformer architecture, with the encoder on the left and the decoder on the right.

The encoder is composed of a stack of layers, each layer consisting of a multi-head self-attention mechanism and a fully connected feed-forward neural network. It takes in the embedded input sequence and generates a hidden representation of the input, where the self-attention mechanism is utilized to weigh the importance of each sequence element and the feed-forward neural network helps to further process the input and produce a more complex representation.

The decoder is also composed of a stack of layers, including multi-head self-attention on the encoder-output and fully-connected feed-forward neural networks. Unlike the encoder, however, each decoder-layer incorporates an additional masked version of the multi-head self-attention mechanism that takes in output-embeddings offset by one position to the right. The masked self-attention and right-shift of the output prevents the model from taking future tokens into account when making predictions. Finally, a linear layer maps the fixed-length representation generated by the decoder to the output space and a softmax layer converts the output into a probability distribution over the possible output classes, which is used to generate the final output sequence.

As the Transformer architecture does not utilize recurrence or convolution and processes entire input sequences simultaneously, it lacks an explicit notion of the order of the elements in the input sequence. For this reason, both the encoder and decoder use positional encoding to provide information about the position of each element in the input sequence and enable the model to understand relationships between the different elements.

3.2 The TrOCR Model

The TrOCR approach by Li et al. [19] leverages previously introduced advantages of Transformers in order to implement an end-to-end OCR model for text recognition that is solely based on the Transformer architecture. The TrOCR model utilizes the standard Transformer encoder-decoder structure. A pre-trained image Transformer is implemented as encoder and a pre-trained text Transformer as decoder. The TrOCR model architecture is shown in figure 3:

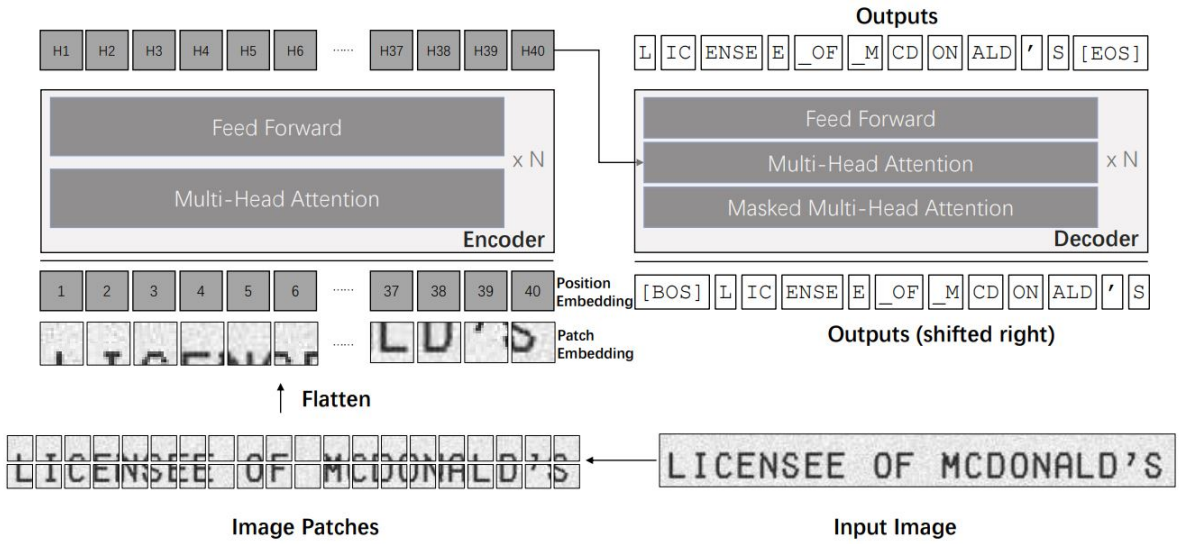

Figure 3: The TrOCR model architecture: An encoder-decoder model with a pre-trained image Transformer as the encoder (left) and a pre-trained text Transformer as the decoder (right) [19]

Since the encoder of a Transformer architecture expects a sequence of tokens as input, each image is first resized to 384×384 and then split into 16×16 patches. Afterwards, the patches are flattened and linearly projected to embedding vectors that match the dimension of the hidden representation throughout the Transformer architecture. The image Transformer takes in the embedded input and generates a hidden representation, which is then processed by the text Transformer to generate wordpiece-level text as the final output. During training, the decoder takes in right-shifted ground truth sequences and learns by comparing the generated output with the original ground truth sequence using the cross-entropy loss.

Both the image Transformer and text Transformer were initialized with publicly available models that have been pre-trained on large-scale datasets. While the DeiT [32] and BEiT [3] models were used for the encoder initialization, the decoder was initialized utilizing the RoBERTa [23] and MiniLM [35] models. Different initialization model combinations were evaluated by Li et al. [19] and the best ones were implemented for three different TrOCR models with varying amounts of model parameters: TrOCR$_{SMALL}$, TrOCR$_{BASE}$ and TrOCR$_{LARGE}$. The TrOCR$_{BASE}$ model, which was used in this paper, uses the encoder of BEiT$_{BASE}$ and the decoder of RoBERTa$_{LARGE}$, and consists of a total of 334M model parameters.

The TrOCR pre-training based on these initialized models was conducted on different text recognition tasks over two subsequent stages. During first-stage pre-training, each model was trained on 684M textline-images cropped from publicly available PDF document files. The first-stage pre-training yielded the so-called *TrOCR-stage1* models, which were used as a basis for the second-stage pre-training on smaller task-specific datasets resulting in three distinct sets of pre-trained models: *TrOCR-handwritten*, *TrOCR-printed* and *TrOCR-str*. The dataset used for second-stage pre-training of the *TrOCR-handwritten* models contained a total of 17.9M handwritten textline-images, that for the *TrOCR-printed* models contained 3.3M printed textline-images and that for the *TrOCR-str* models contained 16M scene text-images. All the different pre-trained TrOCR models are publicly available on the HuggingFace website [20]. For further details on the models as well as the training- and evaluation results, please refer to the work of Li et al. [19].

As neither of the available second-stage TrOCR models have been pre-trained specifically on code screenshots, the TrOCR$_{BASE}$-stage1 model will be used as a basis for an implementation on that specific downstream task. In the following sections, we demonstrate the implementation and fine-tuning of the this model on a dataset of code-screenshots and compare its performance to the Tesseract model as well as the other three second-stage pre-trained models.

4 Methodology

In the subsequent sections, the pre-processing steps required to ready our model for training are explained, including visualizations from our exploratory data analysis to enhance comprehension of the data. Additionally, we delve into the training process, explaining the rationale behind the chosen parameters and providing an overview of the training duration. Finally, the section is concluded by discussing the theory behind the chosen evaluation metrics, namely *character error rate* (CER) and *Levenshtein distance*.

4.1 Pre-Processing

The given data was automatically generated from GitHub repositories, along with screenshots of the integrated development environment (IDE) used and an annotations file containing information about the text in each corresponding line. The first task was to extract each code line from the 6,235 screenshots using the annotations files. These files contained all necessary data for the cropping process, including the x- and y-positions of each line. The screenshot cropping process is illustrated in Figure 4.

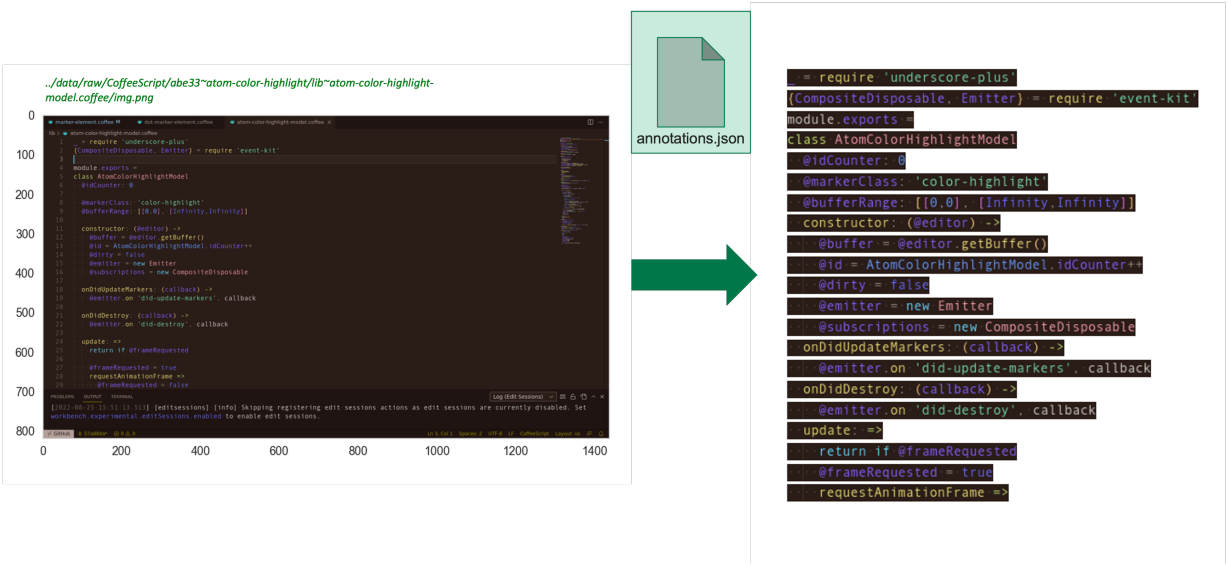

Figure 4: Example screenshot that shows the cropping procedures we had to take during the preprocessing step. For each screenshot we crop each code line using the information provided in the annotations.json. The code lines are then used for training the transformer. *(Author's Illustration)*

After the cropping of each code line was complete, some further pre-processing steps were necessary. Unfortunately, some values in the annotations.json violated our consistency conditions and the corresponding rows had to be removed. Violation of the consistency conditions included, for example:

- Negative positional values in x and / or y position
- Negative text sizes
- Chinese letters in the label / target, i.e. text column

Empty target values (None values) were another issue that had to be addressed. Rows where the label contained at least one chinese letter were automatically flagged so they could be eliminated from the training dataset.

To address problems caused by significant outliers, an outlier threshold $\theta = 110$ was defined. Each code-line with a text-length greater than the threshold was removed (disregarding whitespaces). This was done to prevent a significantly right skewed target distribution, which could potentially introduce bias to the model predictions. We also had to remove such outliers to find an adequate max padding size of our input encodings, without adding too much noise by zero paddings. The target length distribution can be seen in Figure 5.

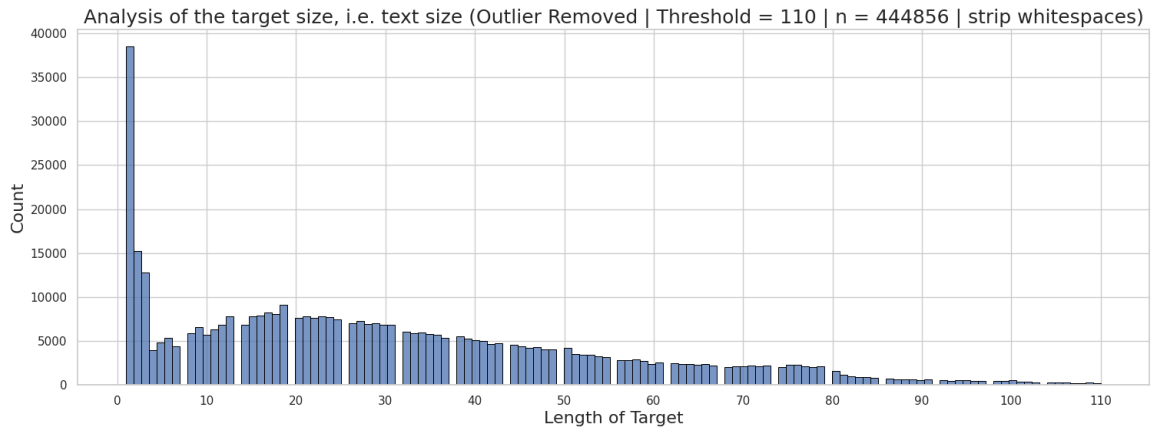

Figure 5: The target text-length distribution. Whitespaces are stripped beforehand. The size of the labels of most code lines are in the interval of (0, 60] with a relatively long tail of longer labels. *(Author's Illustration)*

Conducting all mentioned pre-processing steps yielded a final dataframe of around 450,000 usable code lines snippets. This dataframe was then used for the training process and is illustrated in Table 4.1. As the input of the TrOCR model is the image, only the image path, i.e. 'line_img_path' and its label 'text' were necessary.

	line_img_path	font	theme	language	line_number	text	text_size	contains_chinese
line211130	../data/raw/Scala/aselab~scala-activerecord/ac...	Cousine	Leios	Scala	20.0	m1.save()	9	False
line324829	../data/raw/Rust/redox-os~tfs/atomic-hashmap~s...	Luculent	Zenburn	Rust	47.0	/// collisions, which can otherwise make the h...	85	False
line439746	../data/raw/Ruby/hanami~hanami/lib~hanami~vers...	monofur	Theme	Ruby	17.0	end	3	False
line334170	../data/raw/Perl/HackerDom~checksystem/lib~CS....	Droid Sans Mono	Dark	Perl	127.0	}	1	False
line420720	../data/raw/JavaScript/chartjs~Chart.js/src~co...	Oxygen Mono	Monokai++	JavaScript	39.0	dash: [],	9	False
line105786	../data/raw/Python/martomo~SublimeTextXdebug/t...	Oxygen Mono	Trepid	Python	70.0	context_view = self.get_view_by_title(...	55	False
line282886	../data/raw/Kotlin/Syex~MoviesMPP/app~src~comm...	Courier New	elly	Kotlin	33.0	assertEquals(TestUtils.pop...	41	False
line365185	../data/raw/C/lc-soft~LCUI/test~test_settings....	Consolas	Theme	C	17.0	{	1	False
line143765	../data/raw/CoffeeScript/sinisterchipmunk~jax/...	Anonymous Pro	Elements	CoffeeScript	32.0	if obj.castShadow	17	False
line185321	../data/raw/Dart/JideGuru~FlutterTravel/lib~ut...	Crystal	Theme	Dart	77.0	];	2	False
line343478	../data/raw/Perl/Difegue~LANraragi/lib~LANrara...	Courier New	Night Owl	Perl	4.0	use warnings;	13	False
line237396	../data/raw/Swift/philackm~ScrollableGraphView...	Menlo	Theme	Swift	9.0	/// The shape to draw for each data point.	42	False
line436612	../data/raw/Ruby/jnunemaker~flipper/spec~spec_...	ProggyCleanTT	Theme	Ruby	3.0	require 'pp'	12	False
line261862	../data/raw/HTML/divshot~geo-bootstrap/bootstr...	Ubuntu Mono	Monokai Vibrant	HTML	11.0	<link href="../assets/css/bootstrap.css" r...	58	False
line301624	../data/raw/Rust/wasmerio~wasmer/lib~api~src~s...	Menlo	luxark	Rust	33.0	/// let export = instance.exports.get_function...	64	False
line124391	../data/raw/Java/Cleveroad~AdaptiveTableLayout...	monofur	Dark Feminine Italic	Java	26.0	import com.cleveroad.sample.R;	30	False
line282395	../data/raw/Kotlin/skydoves~GithubFollows/app~...	Andale Mono	Electron vue	Kotlin	18.0	* FITNESS FOR A PARTICULAR PURPOSE AND NONINF...	77	False
line224184	../data/raw/Scala/plokhotnyuk~jsoniter-scala/j...	Oxygen Mono	Linux Themes for VS Code	Scala	24.0	intercept[Throwable](b.avSystemGenCodec())	42	False
line42427	../data/raw/Elixir/spandex-project~spandex/lib...	Fira Code	Theme	Elixir	6.0	* 'id': The trace ID, which consistently ref...	80	False
line451705	../data/raw/Objective-C/forkingdog~UITableView...	Edlo	Kanagawa	Objective-C	1.0	// The MIT License (MIT)	24	False

Table 1: Preview of the full train data with a shape of $(358647, 8)$, which is used to train the transformer. The model expects the image (extracted from the *'line_image_path'* column) and the label (*'text'* column).

With all necessary data for training now obtained, the training process can now be examined in the following section.

4.2 Training

The available data was split into a train, validation and test set with a proportion of around 80%, 10%, 10%. The test set served as hold-out-set to allow for the evaluation of the models generalization ability on unseen data. An

emphasis was set on making sure that the test set contained certain themes and fonts that were not part of the training set to further evaluate the model's generalization ability and avoid overfitting.

The model was trained over 10 epochs with a batch size of 64. The Adam optimizer and cross entropy loss were used with a learning rate of $5e^{-5}$, the same as in the original paper. After each epoch, the model's performance was evaluated on the validation set using the CER. After the training process terminated, the final model was evaluated on the unseen test set using the CER and Levenshtein distance. The training and validation results after each epoch and the evaluation results on the test set are presented in Section 5.

4.3 Evaluation metrics

For evaluating the models performance the *character error rate* (CER) was used on the validation set after each epoch as well as on the test set. The CER is defined as follows [14]:

$$CER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{1}$$

where ...

- S is the number of substitutions
- D is the number of deletions
- I is the number of insertions
- C is the number of correct characters
- N is the number of characters in the reference (N=S+D+C)

The CER is a common metric to evaluate the performance of automatic speech recognition system (ASR) and OCR models. CER is very similiar to the *word error rate* (WER). However, CER operates on the character-level, whereas WER only takes complete words into account. The CER is always ≥ 0, where 0 means a perfect fit, i.e. a perfect match between the reference and the prediction. However, it can also be greater than 1 if the number of insertions is too high. The CER value indicates the percentage of characters that were incorrectly predicted [14].

Another metric used for the evaluation of our model is the so called *Levenshtein distance* [18]. It describes the minimum number of edits of characters (insertion, deletion or substitution) to transform one string to another and leverages dynamic programming. The Levenshtein Distance for string a and b, where $|x|$ denotes the length of a string x, is defined as follows [18]:

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise,} \end{cases} \tag{2}$$

where the tail of a string x is defined as the substring that consists of all the characters in x, except the first one. The n-th character of the string x, can be accessed using index notation $x[n]$, counting from 0.

For a better comparison, we used the normalized Levenshtein distance (ratio), that takes values in [0, 1], where 1 indicates a perfect matching.

5 Results and Comparison

In this section, the training and evaluation results are discussed and the final model is compared to the Tesseract model [1] as well as the TrOCR$_{BASE}$ models available on HuggingFace [20]. Furthermore, the model will be evaluated on a binarized version of the test set to explore the impact of different color-themes on the model performance. Lastly, an analysis of the model's tendency to predict and insert noise characters derived from the IDE is performed.

5.1 Training Results

The model was trained on a single NVIDIA A100 80GB GPU over a time span of four days. The training result after 10 epochs is illustrated in Figure 6. Although the model steadily improves over the course of the training both regarding the training loss and the CER on the validation set, it still leaves some room for improvement.

One significant problem was the fact, that the model appeared to have learned background-noise from the IDE screenshots. Many screenshots contained indicators for indentations such as '|' or '→' which were not part of the target labels. However, as they were still visible in the screenshots, the model inserted these noise characters into predictions on numerous occasions, worsening the overall CER score. Since the CER is very sensitive to insertions, it took up values of over 50 and distorted the mean CER. As demonstrated later in this chapter, the mean CER is significantly lower if test samples containing such noise characters are excluded from the evaluation.

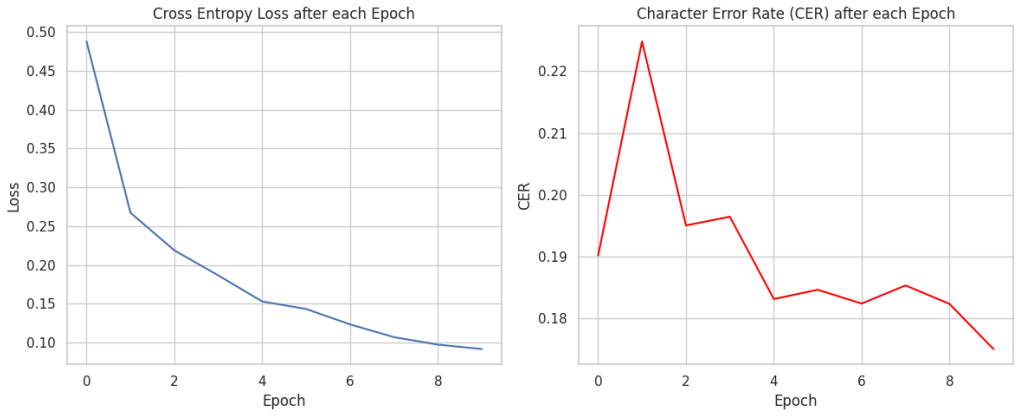

Figure 6: Cross Entropy Loss and mean CER after each epoch. The CER is calculated on the validation set after each epoch. The model yields a favorable result, but the CER leaves room for improvement due to noise characters such as '→' and '|' that the model falsely predicts. *(Author's Illustration)*

5.2 Evaluation Results

The test set contained 44,277 rows of unseen code line snippets. A sample of 7 predictions is demonstrated in Figure 7, where a sample containing noise characters mentioned above can be seen (third last sample). Each sample in the figure contains the sample label, the model's prediction, the computed CER and finally the corresponding code line input-image.

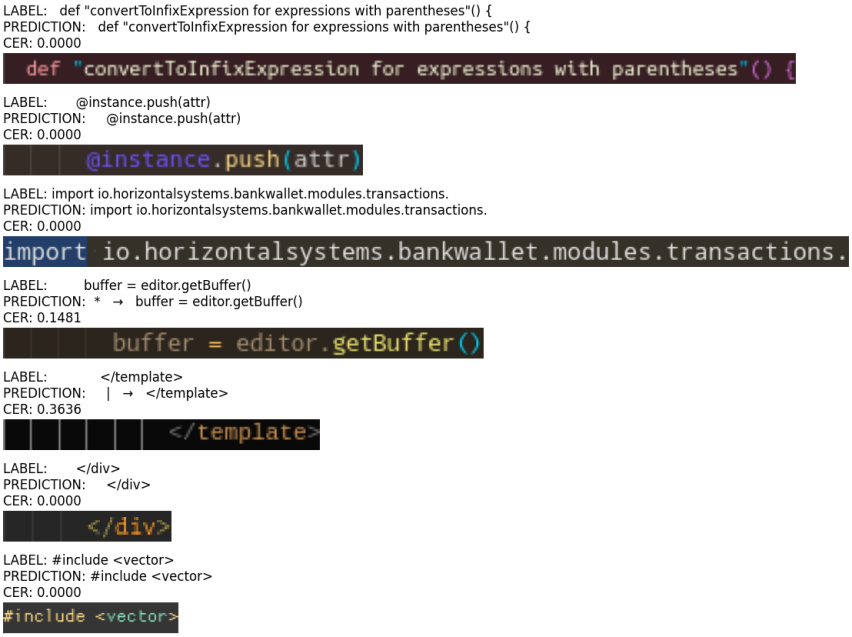

Figure 7: Models prediction on n = 7 random samples from the test set. For each sample, the model prediction, label, computed CER and input-image is shown. In the third last sample, the model predicts noise characters that increase the CER significantly, even though the actual text is transcribed perfectly *(Author's Illustration)*.

Table 2 shows the results of the CER and Levenshtein distance metric computations for different test set variations. The first two rows show the models performance on the original test set with and without noise characters ('|', '→'). As already mentioned, the models performance significantly improves when predictions containing noise characters are excluded from the test set. Rows three and four show the performance of the model on a binarized version of the test set, where each code line consists of black code on white background. This was done to evaluate whether the model takes color themes into account when making predictions. Since the model performs very similarly on both sets, it can be assumed that the color does not significantly affect the model prediction and the main focus lies on the actual text content of each input-image. Unfortunately, the problem regarding noise characters persisted, as they were still visible in binarized samples.

	Mean CER	SD CER	Median CER	Mean Levenshtein	SD Levenshtein	Median Levenshtein	Support
Test Set	0.2008	0.8132	0.0000	0.9194	0.1383	1.0000	44,277
Test Set w/o Noise	0.1134	0.5167	0.0000	0.9403	0.1247	1.0000	36,413
Binarized Test Set	0.1882	0.8172	0.0000	0.9069	0.1562	1.0000	44,277
Binarized Test Set w/o Noise	0.1436	0.8209	0.0000	0.9223	0.1504	1.0000	38,096

Table 2: Evaluation of the fine-tuned TrOCR model with CER and Levenshtein distance for different kind of test set variations. Exluding objects that contain at least one noise character leads to a significant improvement of the models performance.

5.3 Comparison

To put the performance of our final model into perspective, the Tesseract model [1] was used as a benchmark. Tesseract is a popular open source and easy to use OCR engine, which uses CNNs instead of Transformers. In the comparison, we also differentiated between all test samples and those that do not include any noise

characters. Furthermore, the following base stage models from the work of Li et al. [19] available on HuggingFace [20] were used as benchmarks:

- TROCR$_{BASE}$-stage1: The stage 1 pre-trained model that was used as a basis for our fine-tuned model.
- TROCR$_{BASE}$-printed: The stage 2 model pre-trained for printed character recognition.
- TROCR$_{BASE}$-handwritten: The stage 2 model pre-trained for handwritten character recognition.
- TROCR$_{BASE}$-str: The stage 2 model pre-trained for scene text recognition.

As the evaluation metric the CER was used, since it is widely used in the research of OCR engines and was also chosen in the original TrOCR paper by Li et al. [19] The final results can be seen in Table 3.

	Support	**Mean**	**SD**	**Min.**	**Q25**	**Q50**	**Q75**	**Max.**
Fine-tuned Model	44,277	0.2008	0.8132	0.0000	0.0000	0.0000	0.2000	102.00
Fine-tuned Model w/o Noise	36,413	0.1134	0.5167	0.0000	0.0000	0.0000	0.0600	24.000
Tesseract	44,277	0.4434	0.4625	0.0000	0.0000	0.1765	0.9375	14.000
Tesseract w/o Noise	42,688	0.4455	0.4645	0.0000	0.0000	0.1750	0.9474	14.000
Base Stage 1	44,277	0.7127	2.2413	0.0000	0.1000	0.2759	0.7213	84.000
Base Stage 1 w/o Noise	42,307	0.7197	2.2872	0.0000	0.0952	0.2667	0.7264	84.000
Base Printed	44,277	0.9483	0.8573	0.0000	0.7778	0.8772	1.0000	37.000
Base Printed w/o Noise	43,797	0.9486	0.8617	0.0000	0.7750	0.8762	1.0000	37.000
Base Handwritten	44,277	1.0931	1.2170	0.0000	0.6418	0.8485	1.0741	101.00
Base Handwritten w/o Noise	44,277	1.0931	1.2170	0.0000	0.6418	0.8485	1.0741	101.00
Base STR	44,277	1.0150	0.6953	0.0000	0.8125	0.9630	1.0000	12.000
Base STR w/o Noise	44,277	1.0150	0.6953	0.0000	0.8125	0.9630	1.0000	12.000

Table 3: Comparison of the trained model (first two rows), which was fine-tuned on the TROCR$_{BASE}$-stage1 model with the Tesseract model [1] and the other stage 2 models by Li et al. [19]. For the evaluation metric the CER was used. All models were tested on the test set consisting of unseen data samples. With a mean CER of 0.2, our fine-tuned model significantly outperformed Tesseract and the other models.

Our fined-tuned model achieved a mean CER of 0.2 with a median of 0.00. It significantly outperformed the Tesseract model, which only achieved a mean CER of 0.44 with a median of 0.18. When excluding noisy predictions, our model's performance improved even further to a mean of 0.11, whereas the Tesseract model's performance stayed the same. This becomes even more clear when comparing the standard deviations and max-values regarding the CER of our fine-tuned model and the Tesseract model on test sets with and without noisy predictions. While our model significantly improved at all fronts, the measures barely even changed for the Tesseract model. This indicates that, although the Tesseract model performed worse overall, it struggled less with the noise captured within code screenshots, as it was not trained on the same training dataset.

Clearly, the other pre-trained TrOCR models were not able to achieve the same level of performance as the fine tuned model. This was somewhat expected, given that they were trained in different contexts. We included them for the sake of completeness. Comparably, TROCR$_{BASE}$-stage1 model performs the best relative to all pre-trained TrOCR models. This supports our choice of the stage 1 model as a basis for our fine-tuned model. Furthermore, the impact of the fine-tuning becomes clear when comparing the TROCR$_{BASE}$-stage1 model to our final fine-tuned model. While the original stage 1 model only achieves a mean CER of 0.71 on the test set, the fine-tuned model achieves a mean CER of 0.2, which amounts to a relative improvement of +355%.

Overall we can conclude that our fine-tuned model based on the TROCR$_{BASE}$-stage1 model by Li et al. [19] yields a very favorable result considering the lack of hyperparameter tuning, which was not conducted due to time and resources limitations.

6 Conclusion

In this elaboration, we proposed a Transformer-based OCR model based on the work of Li et al. [19] and fine-tuned it on a dataset of code screenshots. The final model outperformed the popular Tesseract model [1], achieving a mean CER of 0.20 and median of 0.00 compared to Tesseract's mean CER of 0.44 and median of 0.18. We briefly touched upon important related work in the domain of OCR and summarized the key theoretical concepts behind the Transformer architecture [33] as well as the TrOCR model by Li et al. [19]. Subsequently, we explained the methodology behind taken pre-processing steps and the training and evaluation of our final model. Finally, we discussed the results of the training and performance evaluation of our model on an unseen test set in comparison to Tesseract as well as other pre-trained TrOCR models available on the HuggingFace website [20].

Although our fine-tuned model significantly outperformed the models used as benchmarks and achieved a very favorable result overall, a problem regarding noise in the training data was identified as well. This problem was caused by indicators for indentations within IDEs ('|', '→') that occured on input images but weren't part of corresponding target labels. Therefore, the model predicted noise characters on numerous occasions, leading to a significantly higher CER and thus lowering its mean performance. This problem should be addressed in future works to further improve the model. A simple suggestion that might help solving problem is to extend the training dataset, considering that we only used around 450,000 code line samples whereas Li et al. used between 3.3M and 17.9M samples to train their stage 2 models. Another possible solution might be putting in additional effort towards cleaning and better preparing the already available training dataset. Adding an extra pre-processing step to remove noise characters by implementing different binarization techniques may also help addressing this issue.

In summary, our fine-tuned TrOCR model already showed very promising results for the text recognition task on code screenshots. We highlighted the importance of addressing the noise issue as the model performs even better when simply excluding predictions containing noise characters ('|', '→'), achieving a mean CER of 0.11. While our results appear to be a promising milestone, more work needs to be done in order to reach the final goal of an end-to-end OCR engine for code screenshots. Our suggestions for the next steps are, firstly, addressing the mentioned noise problem, and secondly, working on a model for text detection to pair with the TrOCR model, which so far only covers the text recognition task.

References

[1] Tesseract open source ocr engine. https://github.com/tesseract-ocr/tesseract, 2023-17-01.

[2] ANI, R., MARIA, E., JOYCE, J. J., SAKKARAVARTHY, V., AND RAJA, M. A. Smart specs: Voice assisted text reading system for visually impaired persons using tts method. In *2017 International Conference on Innovations in Green Energy and Healthcare Technologies (IGEHT)* (2017), pp. 1–6.

[3] BAO, H., DONG, L., PIAO, S., AND WEI, F. Beit: Bert pre-training of image transformers, 2021.

[4] BROWN, T. B., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J., DHARIWAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., AGARWAL, S., HERBERT-VOSS, A., KRUEGER, G., HENIGHAN, T., CHILD, R., RAMESH, A., ZIEGLER, D. M., WU, J., WINTER, C., HESSE, C., CHEN, M., SIGLER, E., LITWIN, M., GRAY, S., CHESS, B., CLARK, J., BERNER, C., MCCANDLISH, S., RADFORD, A., SUTSKEVER, I., AND AMODEI, D. Language models are few-shot learners, 2020.

[5] CARION, N., MASSA, F., SYNNAEVE, G., USUNIER, N., KIRILLOV, A., AND ZAGORUYKO, S. End-to-end object detection with transformers, 2020.

[6] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[7] DIAZ, D. H., QIN, S., INGLE, R., FUJII, Y., AND BISSACCO, A. Rethinking text line recognition models, 2021.

[8] EDUCATION, I. C. What is optical character recognition (ocr)? https://www.ibm.com/cloud/blog/optical-character-recognition, 2023-14-01.

[9] GRAVES, A., FERNÁNDEZ, S., GOMEZ, F., AND SCHMIDHUBER, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 369–376.

[10] HAIRUMAN, I. F. B., AND FOONG, O.-M. Ocr signage recognition with skew slant correction for visually impaired people. In *2011 11th International Conference on Hybrid Intelligent Systems (HIS)* (2011), pp. 306–310.

[11] HEMANTH, G., JAYASREE, M., VENII, S. K., AKSHAYA, P., AND SARANYA, R. Cnn-rnn based handwritten text recognition. *ICTACT Journal on Soft Computing 12* (2021).

[12] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[13] HONG, T. H., YUN, C. H., PARK, J. W., LEE, H. G., JUNG, H. S., AND LEE, Y. W. Big data processing with mapreduce for e-book. *International Journal of Multimedia and Ubiquitous Engineering 8*, 1 (2013), 151–162.

[14] HUGGINGFACE. Cer - a hugging face space by evaluate-metric. https://huggingface.co/spaces/evaluate-metric/cer, 2023-26-01.

[15] JAMIL, S., PIRAN, M. J., AND KWON, O.-J. A comprehensive survey of transformers for computer vision, 2022.

[16] KANG, L., RIBA, P., RUSIÑOL, M., FORNÉS, A., AND VILLEGAS, M. Pay attention to what you read: Non-recurrent handwritten text-line recognition, 2020.

[17] LEI, Z., ZHAO, S., SONG, H., AND SHEN, J. Scene text recognition using residual convolutional recurrent neural network. *Machine Vision and Applications 29*, 5 (2018), 861–871.

[18] LEVENSHTEIN, V. I., ET AL. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, Soviet Union, pp. 707–710.

[19] LI, M., LV, T., CHEN, J., CUI, L., LU, Y., FLORENCIO, D., ZHANG, C., LI, Z., AND WEI, F. Trocr: Transformer-based optical character recognition with pre-trained models, 2021.

[20] LI, M., LV, T., CHEN, J., CUI, L., LU, Y., FLORENCIO, D., ZHANG, C., LI, Z., AND WEI, F. Huggingface: Trocr models. https://huggingface.co/models?other=trocr, 2023-28-01.

[21] LI, S., JIN, X., XUAN, Y., ZHOU, X., CHEN, W., WANG, Y.-X., AND YAN, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems 32* (2019).

[22] LIAO, M., WAN, Z., YAO, C., CHEN, K., AND BAI, X. Real-time scene text detection with differentiable binarization. In *Proceedings of the AAAI conference on artificial intelligence* (2020), vol. 34, pp. 11474–11481.

[23] LIU, Y., OTT, M., GOYAL, N., DU, J., JOSHI, M., CHEN, D., LEVY, O., LEWIS, M., ZETTLEMOYER, L., AND STOYANOV, V. Roberta: A robustly optimized bert pretraining approach, 2019.

[24] MAMMERI, A., KHIARI, E.-H., AND BOUKERCHE, A. Road-sign text recognition architecture for intelligent transportation systems. In *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)* (2014), pp. 1–5.

[25] MATHEW, M., JAIN, M., AND JAWAHAR, C. Benchmarking scene text recognition in devanagari, telugu and malayalam. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* (2017), vol. 7, IEEE, pp. 42–46.

[26] MOHAMMADI FARSANI, R., AND PAZOUKI, E. A transformer self-attention model for time series forecasting. *Journal of Electrical and Computer Engineering Innovations (JECEI) 9*, 1 (2020), 1–10.

[27] PEPPER, J., SENIN, A., JEBBIA, D., BREEN, D., AND GREENBERG, J. Metadata verification: A workflow for computational archival science. In *2022 IEEE International Conference on Big Data (Big Data)* (2022), pp. 2565–2571.

[28] PINTO, L. G., MARTINS, W. M., RAMOS, A. C., AND PIMENTA, T. C. Analysis and deployment of an ocr—ssd deep learning technique for real-time active car tracking and positioning on a quadrotor. *Data Science: Theory, Algorithms, and Applications* (2021), 121–155.

[29] SATHEESAN, S. P., BHAVYA, DAVIES, A., CRAIG, A. B., ZHANG, Y., AND ZHAI, C. Toward a big data analysis system for historical newspaper collections research. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (New York, NY, USA, 2022), PASC '22, Association for Computing Machinery.

[30] SCHNEIDER, P., AND MAURER, Y. Rerunning OCR: A machine learning approach to quality assessment and enhancement prediction. *Journal of Data Mining &amp Digital Humanities 2022*, Digital humanities in... (nov 2022).

[31] SHIVAKUMARA, P., TANG, D., ASADZADEHKALJAHI, M., LU, T., PAL, U., AND HOSSEIN ANISI, M. Cnn-rnn based method for license plate recognition. *CAAI Transactions on Intelligence Technology 3*, 3 (2018), 169–175.

[32] TOUVRON, H., CORD, M., DOUZE, M., MASSA, F., SABLAYROLLES, A., AND JÉGOU, H. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning* (2021), PMLR, pp. 10347–10357.

[33] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need, 2017.

[34] WANG, C.-Y., BOCHKOVSKIY, A., AND LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[35] WANG, T., ZHU, Y., JIN, L., LUO, C., CHEN, X., WU, Y., WANG, Q., AND CAI, M. Decoupled attention network for text recognition. In *Proceedings of the AAAI conference on artificial intelligence* (2020), vol. 34, pp. 12216–12224.

[36] WU, B., XU, C., DAI, X., WAN, A., ZHANG, P., YAN, Z., TOMIZUKA, M., GONZALEZ, J., KEUTZER, K., AND VAJDA, P. Visual transformers: Token-based image representation and processing for computer vision, 2020.

[37] ZHOU, H., ZHANG, S., PENG, J., ZHANG, S., LI, J., XIONG, H., AND ZHANG, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021), vol. 35, pp. 11106–11115.

[38] ZHU, D., LI, T., HO, D., ZHOU, T., AND MENG, M. Q.-H. A novel ocr-rcnn for elevator button recognition. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 3626–3631.