Digitize historic architectural plans with OCR and NER transformer models

Kevin Löffler

December 22, 2023

SA

OST - Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Advisor:

Prof. Dr. Mitra Purandare

Abstract

The Swiss Archive for Landscaping Architecture, located at OST in Rapperswil, administers more than 100'000 historic plans. These plans need to be digitzed to make them accessible. This paper proposes a three-model architecture consisting of a layout model to find text on the plans, an optical character recognition model to extract the found words, and finally, a named entity recognition model to label the relevant words like the client, location, or date. K-means clustering is used to group the text blocks from the layout model into related blocks for OCR.

Different deep-learning models are compared and evaluated. The most suitable models are then retrained on the NVIDIA DGX-2 system in a custom-built apptainer image with different training strategies to improve their accuracy. Different pre- and post-processing techniques are employed to improve the accuracy of the pipeline.

The final image pipeline achieves an F1 score of 48% with 35% precision and 77% recall. The chosen NER model "German BERT" scored an F1-score of 86% after re-training and the OCR pipeline extracted 54% of words correctly and 18% close to correct.

The insights from this SA can be applied to future projects to build an application usable by the archive, enabling it to catalog its documents and make them accessible to the world.

Management summary

The Swiss Archive for Landscaping Architecture, located at OST in Rapperswil, administers more than 100'000 historic plans. These plans are currently digitized manually in a very time consuming process. In this paper I propose an AI-pipeline that can help automate this process. Three different Artificial Intelligence models are combined to find all the words on the plan, extract the words and then categorize the words into different entities like client, location or scale.

Different deep-learning models are compared and trained with existing data from the archive to find the best suited for the task. Different techniques are employed to improve the accuracy of the models like removing the archive stamp, remove artifacts from the text and a grouping algorithm to find related words on the plan.

The final pipeline achieves an accuracy score (F1) of 48%. The pipeline acts as a proof of concept for the given task. The insights from this SA can be applied to future projects to build an application usable by the archive, enabling it to catalog its documents and make them accessible to the world.

Contents

List of Figures

List of Tables

1 Introduction

The Swiss Archive for Landscaping Architecture, located at OST in Rapperswil, administers more than 100'000 historic plans. The collection contains estates of renowned Swiss and European architects who worked in the 20th century.

Besides using the plans for teachings at OST, the archive is an important point of reference for today's architects. Having the original plans available is crucial to a lot of projects. The archive started to digitize its plans years ago but has still only managed to digitize a fraction of its documents. This is in part owed to the very manual process that is used to digitize the plans. After photographing the plan all metadata like plan header, scale, creation, etc. has to be recorded by hand.

The goal of this thesis is to explore different machine learning models and approaches that could automate this workflow, enabling the archive to catalog its documents and make them accessible to the world faster.

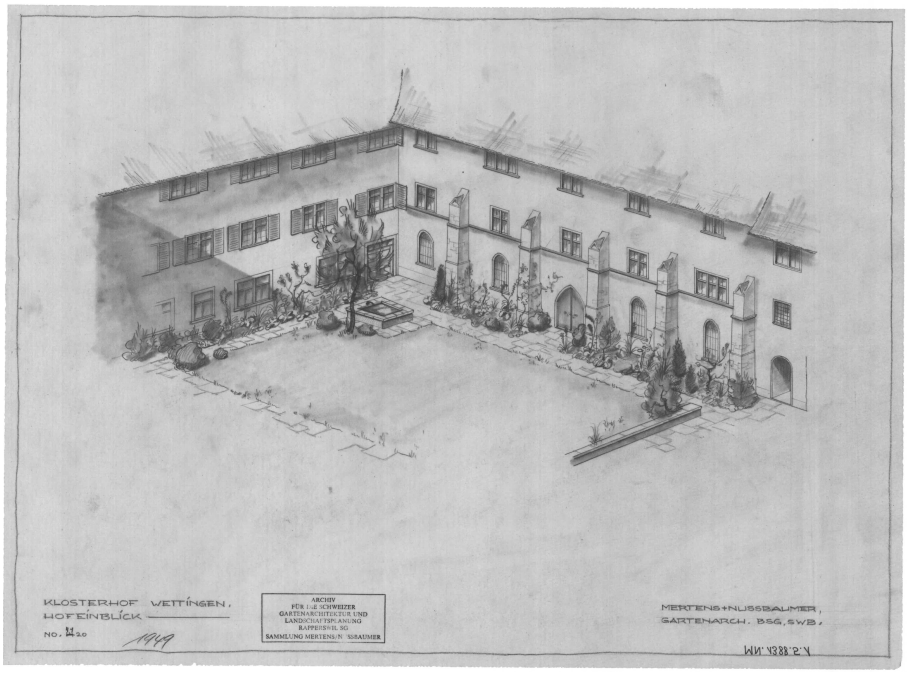


Figure 1: A 1949 architectural plan retrieved from the estate of Mertens and Nussbaumer, *source: "ASLA, Archiv Schweizer für Schweizer Landschaftsarchitektur", n.d.*

2 Conditions

This paper was written as a mandatory part of the bachelor program in computer science at the Eastern Swiss University of Applied Science - OST. The paper was written during one semester and was supervised by Prof. Dr. Mitra Purandare.

3 Abbreviations

NER	-	Named Entity Recognition
OCR	-	Optical Character Recognition
NLP	-	Natural Language Processing
AI	-	Artificial Intelligence

4 Problem analysis

4.1 Current workflow

Between one and two archive employees are working on digitizing the plans. The current workflow consists of the following steps:

1. Photograph the plan and label it
2. Record the metadata:
   (a) Plan head with client and location
   (b) Scale, for example: 1:100, 1:50
   (c) Creation date
   (d) Creation location
3. Edit the image (rotate and crop)
4. Redact personally identifiable information and floor plans of non-public buildings

In this SA I want to focus on automating the recording of the metadata. Firstly, this is a very time-consuming step, as well as a crucial one that is prone to mistakes like typos in names or wrong scales. Secondly, this step suits the capabilities of modern image and language models. More steps can be explored in future works.

4.2 Pretrained OCR models

Text on plans has no clear structure. Landmarks like the title or plan header can be located everywhere on the plan and can have vastly different structures and components, parts of the text like measurements or labels can be rotated and may have different sizes. Additionally, there are a lot of graphical elements on a plan that an OCR model might confuse with characters. Given these constraints, the OCR pipeline needs to segment or localize words before extracting them.

4.3 Pretrained language models

After the words are extracted, they need to be classified into different entities or discarded if they are not important to the archive. In this paper, several different pre-trained language models and architectures are compared. These models work quite well on current mainstream texts, like an article on Wikipedia or in the news. Two problems inhibit the direct application of these models in the domain of historic, architectural plans:

1. Some textual patterns are very old. For example: names are sometimes written in the following format "surname, title first name" which the models often do not recognize as a complete name.

2. Some entities like scale are specific to the problem domain and there are no pre-trained models.

4.4 Training data

Because the archive has been working on digitization for several years, thousands of documents are already photographed and labeled. Because the people working in the archive often change, the quality of the labeled data varies between projects, and the data needs to be curated and cleaned before it can be used for machine learning. Additionally, the existing data is only suitable for certain training cases.

4.4.1 OCR

There is no directly usable OCR training data because the current records only contain the text strings and no coordinates for the bounding box which would be necessary. Additionally, only a fraction of all the text, the entities mentioned before, are recorded. If the performance of the OCR model cannot be increased to a satisfactory level with model selection or preprocessing, manual training data will need to be created.

4.4.2 NER

The basis for NER training is good with thousands of manually recorded headers, scales, and dates. The data is scattered over different Excel files. The entities 'bauherr' (client) and location are not recorded separately but are usually part of the header and will need to be extracted by hand if the model should be trained with them.

5 Goals and Requirements

This thesis focuses on research, comparing different models and finding an approach that can be built into an application that the archive can use in a later stage. Nonetheless, several requirements were specified.

5.0.1 Functional requirements

- The system should find related groups or blocks of text in scanned images
- The system should be able to extract characters and words from the identified groups of text
- The system should extract a set of predefined entities (client, location, scale, date, creation location) from the text
- The system should support JPEG and TIFF formats
- The system should ignore irrelevant archival data (e.g. stamps)

5.0.2 Non-functional requirements

- Correctly identifying more relevant entities, such as the client and location, is more important than scale or the creation location
- Although speed is not a major concern in this paper, the whole pipeline should respect the constraints of a future application
- Compiling a comprehensive comparison and analysis of different models and architectures
- Based on the comparison, finding a suitable model architecture for OCR and NER

6 Architecture

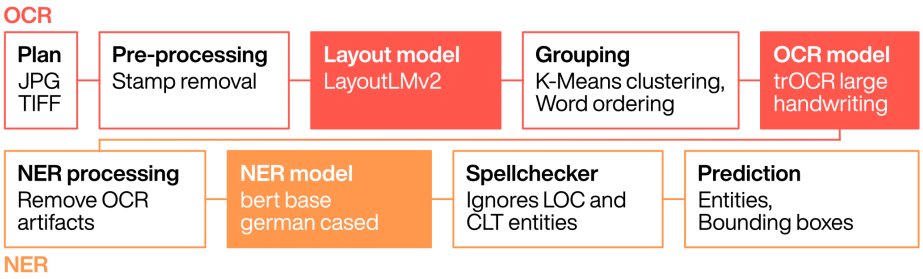


Figure 2: The whole pipeline with its major components, the deep learning models are marked with filled blocks.

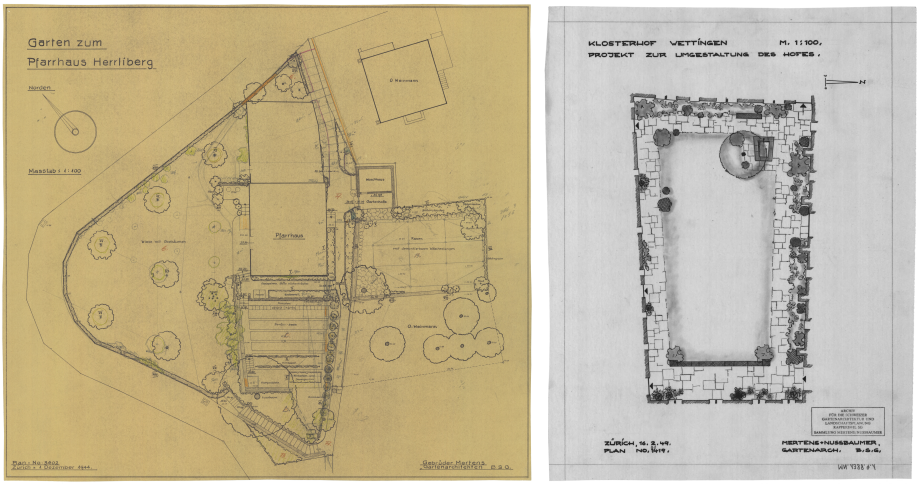
The task is divided into two distinct components: Optical Character Recognition (OCR) and Named Entity Recognition (NER). The OCR pipeline begins by taking the input image and undergoing preprocessing to eliminate archival artifacts, such as the archive's stamp. Subsequently, a deep learning neural network is employed to localize all words within the document. Following this, a clustering algorithm, utilizing k-means, and a word ordering algorithm are applied to group related words into text blocks. The resulting bounding boxes serve as input for the OCR model, which extracts the text. This extracted text is then passed to the NER pipeline.

The initial step in the NER pipeline involves the removal of OCR artifacts, such as trailing periods. Following this, a natural language processing model is deployed to extract predefined entities, such as client or location. Lastly, a spellchecker is utilized to refine the text and correct misidentified characters from the OCR step. The outcome is a set of labeled entities accompanied by their corresponding bounding boxes.

7 ORC - Optical character recognition

7.1 Introduction

Optical Character Recognition describes the process of extracting text from an image, stored as pixel data to a string of characters, thereby making the text searchable and editable. While historically OCR relied on computer vision and image processing techniques, today has witnessed substantial advancements driven by breakthroughs in deep learning and neural network architectures (Shetty & Sharma, 2023).



(a) A top-down view of a property drawn by the brother Mertens in 1944

(b) The courtyard of a monastery in 1949 by Nussbaumer and Mertens

Figure 3: Examples from the archive showing the different types of plans the OCR pipeline needs to handle, *source: "ASLA, Archiv Schweizer für Schweizer Landschaftsarchitektur", n.d.*

7.2 Goal

The goal of the OCR part of the pipeline is to find one or a combination of models that can find and extract text from the plans as well as provide the bounding boxes of each word. The bounding boxes are on one hand important for the application in the future to make it easier for the archiving person to check the predictions. On the other hand, they can be used to gather data to train the model in the future. The extracted text is handed to the NER model.

7.3 Pre-processing

Lombardi and Marinai, 2020 state that after digitization, the first step in a traditional DIAR pipeline is the input images pre-processing. This task is aimed at improving the document quality either for a better human inspection of the work or for improving the automatic recognition of subsequent processing steps. Because of the nature of historical documents this step is particularly relevant in this area as demonstrated by the diverse papers related to pre-processing of historical documents.

Several image processing techniques are discussed in different papers to improve documents for OCR. Because of time constraints, these techniques were not explored in this paper but should be planned for further works on this topic.

7.3.1 Stamp removal

Some plans within the dataset contain a stamp, which was introduced during the archival digitization process.

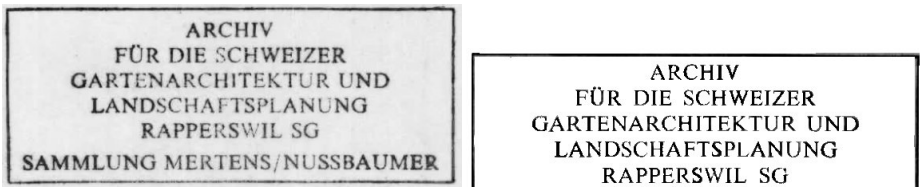


Figure 4: Example of a stamp on the left and edited template on the right with the bottom row containing the estate cut off.

The text present on these stamps is detected and extracted by the layout and OCR model. However, this extraction process can lead to confusion for the NER model and result in longer inference times. To address these issues, a template matching algorithm is employed before the image is processed by any model. If the algorithm identifies coordinates above a specified confidence threshold, it samples and averages the colors of six surrounding pixels to determine the background color of the plan in that region. Subsequently, the entire stamp is replaced with this color, effectively eliminating it from the plan without any sharp lines or artifacts that could influence the text detection. Tests have shown that a modified version of the stamp where the bottom row and line are cut off works the most reliably for the template matching algorithm.

7.4 Preliminary testing

To get a grasp of how well modern off-the-shelf models work I tested different popular models on a small set of images. The result shows that line predictors do not work with the complex layouts that are found on the plans. Before any characters or strings can be recognized, the text needs to be extracted. The LayoutLM model (Xu et al., 2020) jointly models interactions between text and

layout information across documents. An approach like this where words are first found on the page, then grouped into text blocks and after that, a more classic OCR technique is used to digitize the text seems necessary for the given problem.

7.5 Models

The field of OCR does encompass a wide variety of different approaches. The following models where compared:

7.5.1 Tesseract

Tesseract (Smith, 2007) is an open-source Optical Character Recognition (OCR) library developed by HP Research and maintained by Google, incorporating Long Short-Term Memory (LSTM) networks. Its applicability to the given use case is notable as it locates words on a page as well as extracts them. It provides bounding boxes and confidence scores for each prediction. Notably, Tesseract is very fast and there exists a German model. When testing the pretrained German model it reliably finds all relevant words, but the predictions are less accurate compared to other models. This could stem from the fact that Tesseract is not optimized for handwritten text.

7.5.2 TrOCR

Like in many other areas of machine learning transformers have become a popular choice for OCR recently. The TrOCR model (Li et al., 2023) consists of a vision transformer (encoder) and a language transformer (decoder). The encoder divides and flattens the input image into a single row of patches and then generates image embeddings. The decoder takes these embeddings and produces the string output. Both the encoder and decoder consist of multi-head attention and feed-forward blocks. The decoder additionally has a masked multi head attention layer. This architecture lends itself to extensive pretraining, and is therefore ideal for our use case where training data is not available. TrOCR has several pretrained models. The large-handwritten is the most accurate model but also the slowest. One problem is that TrOCR needs a single line input image, meaning it can't find words on a page or do predictions on multiline text blocks. Therefore, this model can only be used in collaboration with another model.

7.5.3 LayoutLM

LayoutLM (Layout Language Model) (Xu et al., 2020) is a model designed for document image understanding. It was developed by Microsoft Research Asia and introduced in 2019. LayoutLM is built upon the BERT (Bidirectional Encoder Representations from Transformers) architecture. However, LayoutLM extends BERT to handle both textual content and layout information in document images.

7.5.4 LayoutLMv2

LayoutLMv2 (Xu et al., 2022) is the improved version of LayoutLM proposed in 2020. It's built on Facebooks Detectron2 detection and segmentation library and can extract layout and style information from a document. In my tests, the model performs identical to the LayoutLM model.

7.5.5 LayoutXLM

LayoutXLM (Xu et al., 2021) is a multilingual extension of the LayoutLMv2 model trained on 53 languages with identical performance in my tests. The language agnostic training does not seem to impact the models performance when it comes to word localization.

7.5.6 EasyOCR

EasyOCR (Jaded-AI, 2020) is a ready-to-use OCR library with support for more than 80 languages, including german. It utilizes CRAFT for text detection and a ResNet, LSTM and CTC architecture. It does not yet support handwritten text which could explain its performance, which does not match that of the TrOCR handwritten model. For text detection on the other hand EasyOCR scores a perfect score with all 90 relevant words detected. EasyOCR could be a faster alternate layout model to LayoutLMv2 if no style or layout information is needed.

7.6 Text detection comparison

The models capable of text detection where tested on different sample images with a total of 90 relevant words. All models perform equally well in terms of

Model	**Found words**	**Accuracy**
Tesseract	88/90	97.8%
LayoutLM	87/90	96.7%
LayoutLMv2	87/90	96.7%
LayoutXLM	87/90	96.7%
EasyOCR	90/90	100%

Table 1: OCR layout model comparison

finding words on the page. The LayoutLM models seem to perform the same on the given data. They predict words very precisely where the other models tend to group close by words into a single bounding box. On the other hand the LayoutLM models tend to predict more false positions for words than the other models.

7.7 Clustering

On architectural plans, words are organized into groups, such as the plan header, creation date and location block and the architect's address and signature, among others. These groupings play a pivotal role in named entity recognition as they provide essential contextual information. Consequently, there is a need to spatially cluster the words to preserve their relationships. To achieve this, a k-means algorithm is employed, given its capability to group related words based on their proximity. As the actual number of groups is unknown, the algorithm is executed with varying k values ranging from 3 to 12. The k-value with the highest silhouette score is then selected as the optimal clustering parameter. Finally, the boxes in each cluster are ordered from top to bottom, left to right based on their coordinates. This approach ensures a robust grouping mechanism for enhancing the accuracy of named entity prediction.

7.8 Post-processing

7.8.1 Removing OCR artifacts

As outlined earlier, the bounding boxes generated by the LayoutLMv2 model undergo padding on all sides to ensure all characters are fully detected. However, this precautionary measure introduces a strange artifact: a period (".") is consistently appended to the end of the majority of predicted words. When the padding is reduced or set to zero, the extraneous dots vanish, but this adjustment adversely impacts prediction quality. To address this issue, a post-processing step is implemented to remove trailing periods from all predictions. This could lead to valid periods being inadvertently excluded, but their absence does not degrade the NER performance.

7.8.2 Spell checker

The output of an OCR model is commonly subjected to spellchecking (Mohapatra et al., 2013) to correct inaccurately predicted characters. (Rosebrock, 2021) In this paper, I experimented with this approach, employing the pyspellchecker (Barrus, 2022) package as the spellchecking tool. The vocabulary was augmented with a compilation of the 10,000 most frequently used words in the German language ("Wortschatz Uni Leipzig - Top 10'000 Wörter", 2001). Despite the spellchecker effectively correcting certain words, it exhibited a tendency to erroneously correct names and locations that should have been exempt from correction. Given the amount of unique names and locations in our dataset that are absent from standard dictionaries, I opted to discontinue this technique. Instead, the spell checker is now applied after the named entity recognition step. This allows names and locations to be filtered before applying the spellchecker.

7.9 Choosen architecture

Following qualitative testing, the TrOCR emerges as the optimal text extraction model. Because it requires single line images, it needs to be paired with a layout model capable of identifying bounding boxes. Despite marginally inferior performance in absolute metrics, the LayoutLM models are deemed most suitable for this project. Their selection is motivated by their potential for future expansion beyond text detection, encompassing elements such as sub-plans, drawings, legends, and more. The minor reduction in accuracy, less than 1%, is considered inconsequential for the current task. The proposed OCR pipeline is as follows:

1. An image is loaded from a path
2. If there is a stamp it is removed with template matching
3. The LayoutLMv2 model predicts the bounding boxes for all words
4. Small boxes (smaller than 20px in one axis) are removed
5. Overlapping boxes are combined using Intersection over Minimum (IOM)
6. K-means clustering is employed with the center coordinates of the boxes to find the word groups
7. The image is cropped to each bounding box and handed to the TrOCR model to extract the text
8. All trailing periods are removed
9. According to the defined order, the predicted words are appended to one string per group

Due to the lack of training data, the selected models were not re-trained. A problem with the LayoutLM model family is that they do not provide a confidence value for the bounding boxes, meaning all boxes need to be processed by the TrOCR model and can then be filtered based on low confidence score, currently -0.1. The prediction time could be shortened if the boxes could be filtered before the OCR step.

7.10 Conclusion

The combination of the two models suits the task the best. The layout model provides reliable bounding boxes for all the relevant words in the document. The TrOCR model works satisfactorily on legible written text but deteriorates quickly on harder-to-read text. One of the main problems identified is that there is no TrOCR model pre-trained on German data. Therefore, the decoder tries to predict English words if they are hard to read which worsens the prediction.

The models were tested on several images and judged on how many words were extracted correctly. They are classified into matches that are extracted

Model	Perfect	Close	Bad
TrOCR	54%	18%	28%
EasyOCR	49%	30%	21%
Tesseract	42%	26%	32%

Table 2: OCR text extraction comparison

perfectly, close matches, where one or two characters are wrong and bad matches where the text is unrecognizable:

Both models could be improved with retraining on custom data which would certainly improve their performance. Gathering such data in sufficient quantities was determined to be outside the scope of this project. Another idea that came up during the project is to use currently available handwriting generators to generate training data in the style of our plans and then use them to fine-tune the models. While this could prove viable, I did not find a generator that generates handwriting close to the style used in the plans and therefore decided not to pursue this approach.

I do believe that accuracy could be significantly improved with different image processing techniques such as denoising, thresholding, distance transforms and opening morphological operations and want to focus on that in further works.

In conclusion the OCR pipeline poses a solid foundation for the following NER pipeline. With its use of two different models, pre- and post-processing techniques as well as a custom written clustering algorithm it presents a solution that is well suited for the given problem. The prediction accuracy is not as good as I initially hoped, but the result is a viable proof of concept.

8 NER - Named entity recognition

8.1 Introduction

Named Entity Recognition is a natural language processing (NLP) technique, focused on identifying and classifying entities within textual data. Classically these entities range from persons and organizations to locations, dates, numbers, and more.

Text is typically tokenized, meaning broken down into smaller units called tokens, which can be words or subwords. These tokens are then classified into predefined categories. This classification is based on the context and characteristics of the token. NER models are often based on transformers or RNN encoders.

8.2 Entities

The archive requested that the model should be able to detect the following entities:

Entity	**Label**	**Description**	**Examples**	**Data**
Client	CLT	The person or organisation owning the property	Hr. Dr. Müller	X
Location	LOC	The location of the project, could be street, municipality, region or a combination	Bahnhofstrasse 12, Rapperswil	X
Scale	MST	The scale of the plans	1:5, 1:50, 1:250	✓
Date	DATE	The creation date of the plan in different formats and styles	1.3.1941, 54	✓
Creation Location	CLOC	The place where the plans where drawn, usually just a city	Zürich, Romanshorn	✓

Table 3: Detailed list of all entites

8.3 Approach

Preliminary tests show again that existing models do not perform very well on the data, with an accuracy of all models in the single-digit percentages. An off-the-shelf model would also not work for the given use case because it requires several custom entities. The client "Bauherr" for example is a compound of a name, with title, and organizations, and the entity's location and creation location do have a very different meaning, although both describing a place. This necessitates training on custom data.

8.3.1 Training data

Several thousand plans are already transcribed by hand. This data is available for training, but there are several problems, prohibiting straightforward usage:

1. The two most important entities, client and location are not labeled independently, they are part of the title
2. Scale data is available but not uniformly formatted
3. Date data is available
4. Creation locations are not well formatted and often seem to be mistaken with object location, rendering them useless

8.4 Process

Initially, emphasis was placed on the extraction of client and location entities, as focusing on the two most important entities instead of all five, would facilitate faster iterations. The initial phase involved the creation of manual training data, with two archive employees extracting client and location entities from a randomly selected set of 500 titles. This process yielded approximately 500 training sentences, supplemented by an additional 100 testing sentences. Subsequently, a set of models was trained and evaluated to initiate the assessment process.

Model	**Accuracy**	**PER**	**LOC**
Spacy	46.36%	54.49%	82.47%
roBERTa	57.56%	39.31%	87.14%
BERT	46.73%	43.75%	88.87%

Table 4: NER base model comparison (client and location only)

8.4.1 Data generation with large language models

The initial results proved promising, however, I sought to improve them further. Recognizing the time-intensive nature of manual data generation, I opted to explore automated methods. My first approach involved leveraging ChatGPT and Huggingchat, two large language models (LLM). Utilizing prompts such as *"I am a data scientist working on an entity recognition model. Can you generate 10 sentences in the style of the example below and use a name and location from this list: [...], example sentences: [...],"* both language models understood the task and generated sentences. Unfortunately, the outcomes were repetitive, with substantial portions of the sentences mirroring the provided examples, rendering them useless for training. Despite introducing prompts emphasizing the importance of uniqueness or prohibiting repetition, these directives were

either disregarded or resulted in sentences too divergent from the original data in structure and content.

Realizing the impracticality of this approach, I opted for a prompt modification. The models were now directed to generate training sentences containing tokens that could later be replaceable by a Python script. For instance, the prompt *"Garden of $$$ in @@@"* where *$$$* would be replaced with a name and *@@@* with a location. Historic municipality names from Switzerland in the year 1960 (*Historisiertes Gemeindeverzeichnis der Schweiz, TXT Format*, n.d.) were utilized for location data, while names were generated by ChatGPT. A sentence was generated for each location, yielding 6,000 training sentences. Subsequently, the models were retrained and evaluated:

Training	**Model**	**Accuracy**	**PER**	**LOC**
Trained on generated data (6000)	Spacy	27.14%	33.64%	38.51%
	roBERTa	26.72%	31.41%	55.25%
	BERT	18.90%	15.83%	39.56%
Trained on manual then on generated (250 + 6000)	Spacy	40.20%	36.89%	50.04%
	roBERTa	42.41%	34.12%	79.33%
	BERT	25.70%	19.73%	51.02%
Trained on generated, then on manual (6000 + 250)	Spacy	44.12%	42.41%	59.87%
	roBERTa	48.83%	39.21%	82.94%
	BERT	31.62%	23.66%	58.06%

Table 5: NER base model comparison with different training strategies (client and location only)

The performance of all models exhibited a decline upon training with the generated sentences. This deterioration is likely attributed to the lack of diversity in the generated data, leading the models to learn specific words rather than capturing general patterns. The introduction of the 250 manually crafted sentences to the training set resulted in an improvement in accuracy, yet it remained below the levels achieved in the initial iteration. Intriguingly, the sequence of training steps was found to have a measurable impact on model performance, accounting for approximately a 6% increase. In conclusion, it can be said that a substantial part of the given entities are out of vocabulary (OOV), and trying to get the model to memorize lists like the municipalities or names does not lead to higher accuracy.

8.4.2 Training strategy

Equipped with insights gained from working with two entities, my objective was to extend the model training to encompass all five entities. The approach involved incorporating entities available in the existing data into manually crafted sentences, thereby providing the required contextual information. I formulated 150 example sentences, similar to before, featuring placeholders for the entities. Each entity is associated with a generator function capable of producing

an infinite number of examples. For client and location entities, the existing manual data from archive employees was utilized. The scales were derived from the archival data and enriched with various prefixes such as *"M: 1:10"*, *"Mst. 1:10"*, *"Masstab 1:10"*, and the like. Dates were randomly generated and formatted in diverse styles, including *"1.1.55"*, *"01-01-55"*, *"1/55"*, *"Jan. 55"* and *"55"* among others. The creation location dates were drawn from a manually curated list of Swiss city and street names. With this training data generator, arbitrary amounts of training sentences can be generated. Of course, generating way more sentences than the 150 templates will lead to the same effect that we observed before with ChatGPT-generated examples. To test the optimal amount of training data the same models were trained on different training set sizes:

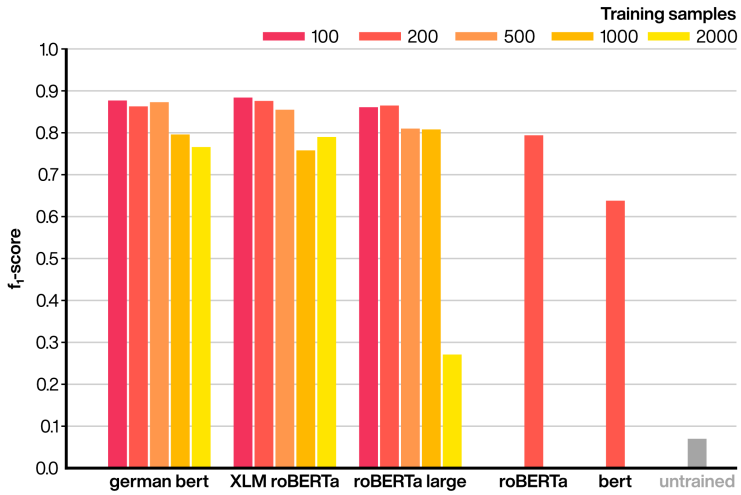


Figure 5: NER model comparison with different training set sizes

Depending on the model the best amount of training data is somewhere between 100 and 500 sentences. Because certain labels are more important to the archive than others the per-label comparison between different models may be more meaningful:

Here we can see that the german-bert model outperforms the XLM roBERTa model on the client and location entities substantially despite having a lower total score.

8.5 Models

8.5.1 BERT

BERT, or Bidirectional Encoder Representations from Transformers (Devlin et al., 2018), is a transformative model in natural language processing (NLP) introduced by Google in 2018. It addresses the limitations of traditional language

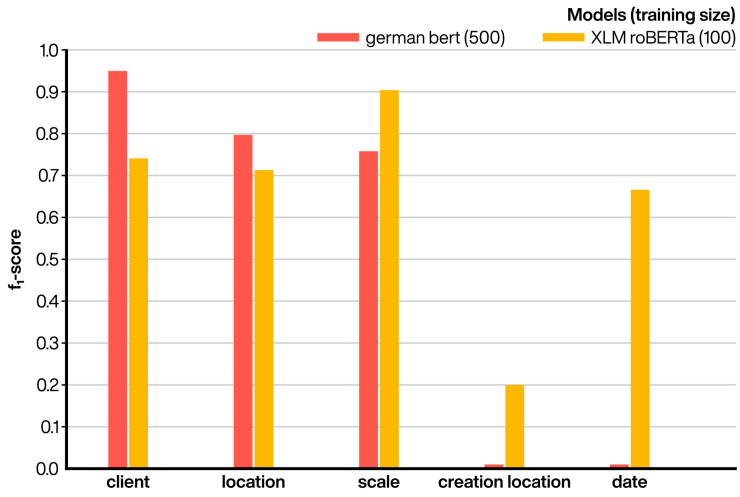


Figure 6: NER model comparison by entities

models by employing a bidirectional context understanding approach within a transformer architecture. Unlike traditional models, BERT processes words in both directions simultaneously, enhancing its ability to capture contextual nuances.

The architecture comprises multiple layers of encoders with self-attention mechanisms, facilitating parallelized training. BERT's pre-training involves predicting masked words in a given context, leveraging a large dictionary. In contrast to traditional recurrent neural networks (RNN), BERT and the other transformer models are trained with Masked language modeling (MLM). Taking a sentence, the model randomly masks 15% of the words in the input then runs the entire masked sentence through the model and predicts the masked words. This allows the model to learn a bidirectional representation of the sentence. Pre-training is followed by fine-tuning on specific tasks, enables BERT to achieve state-of-the-art performance in various NLP benchmarks. Its contextual understanding, transfer learning capabilities, multilingual support, and consistent top-tier performance make BERT a cornerstone in natural language understanding and uniquely suited for the given use case. (Liu et al., 2019b)

For this paper, the BERT base model is used as a baseline benchmark. In my tests, the model consistently ranks at the lower end with a score of 65%. Like the other tested transformer models, BERT requires retraining to be usable.

8.5.2 roBERTa

roBERTa, or robustly optimized BERT approach (Liu et al., 2019a), represents a refinement of the original BERT architecture, introduced by Facebook AI in 2019. Built upon the success of BERT, roBERTa incorporates key modifications to enhance its performance on various natural language processing (NLP) tasks.

Notable adjustments include removing the next sentence prediction objective, training with larger mini-batches and learning rates, and dynamically masking out spans of text during pre-training. These alterations contribute to roBERTa's robustness and improved generalization across diverse tasks and domains.

The roBERTa base model scores well in my tests after retraining, scoring a f1 score of close to 80%.

8.5.3 roBERTa large

RoBERTa Large (Liu et al., 2019b) is an extension of the RoBERTa model, representing a high-capacity transformer-based architecture designed to excel in natural language processing tasks. Introduced by Facebook AI, RoBERTa Large is characterized by a substantial increase in model parameters, enabling it to capture intricate contextual relationships within vast amounts of data. The larger variant of the roBERTa model does bring an increase in prediction accuracy of about 7% over the base model.

8.5.4 XLM roBERTa

XLM roBERTa is an extension of the RoBERTa model and represents a cross-lingual variant designed to address multilingual challenges in natural language processing. Developed by Facebook AI, XLM roBERTa inherits the robust pre-training strategies of RoBERTa while incorporating cross-lingual training objectives. This enables the model to effectively capture language-agnostic features, making it well-suited for tasks across multiple languages. XLM roBERTa should perform better in the test, given its focus on language-independent features, but the improvements in accuracy are negligible compared to the roBERTa large model.

8.5.5 German Bert

German BERT (Chan et al., 2019) is a BERT variant built by deepset. It is trained on more than 10GB of German text data from Wikipedia, OpenLegalData, and news articles. German BERT significantly outperforms Google's multilingual BERT models. My test confirm the accuracy of German Bert with a score of 86% and more importantly a score of 82% on the client entity and a score of 81% on the location entity, outperforming all other tested models on these crucial entities.

8.6 Conclusion

Overall the performance of all NER models is good on the precondition that the input data is perfect, the performance on the OCR predicted text will be discussed in the next chapter. The German BERT model trained on 500 sentences is chosen as the best model for the given task. It performs quite well with client, location and scale entities as these example sentences show:

- Überbauung **Dübendorf (LOC)** Projekt zur Gestaltung der Gärten
- **Herrn A. Wick-Isler Ing. (CLT)** *Schnitte (CLT) Bassin (LOC)* **M 1:50 (MST)**
- **Herrn Dr. Zuellig (CLT)** Schloss **Meienberg (LOC)** Masstab **1:200 (MST)**
- Giesswannerbrunnen Friedhof **Gränichen (LOC)** M. **1:10 (MST)**, Plan nr. *3138 (DATE)*

Legend: **Correctly labeled**, unlabeled, *Incorrectly labeled*

9 Software

This chapter highlights some interesting problems encountered during the project as well as the software architecture used.

9.1 Architecture

All code in this project is implemented in Python, chosen for its widespread adoption in AI research. To enhance code readability and streamline development, type hints are extensively incorporated. Leveraging the benefits of type hints, such as autocomplete in combination with an IDE, facilitates smoother handling of diverse data types and formats. Given the unique format requirements for input data across various models, I made the decision to design specific classes. These include classes for input data (Fragment), entities, prediction results, and an enum encapsulating all entity labels. This modular approach ensures a more organized and adaptable codebase, tailored to the specific demands of different machine learning models.

```
@dataclass
class Entity:
    text: str
    label: EntityLabel
    start_index: int
    end_index: int

    def __init__(self, text: str, label: EntityLabel, title: str):
        self.text = text
        self.label = label
        self.start_index = self.__find_start_index(text, title)
        self.end_index = self.start_index + len(text)

    @staticmethod
    def __find_start_index(text, title):
        if len(text) == 0:
            raise ValueError(f'entitiy is empty, title: "{title}"')

        start_index = title.find(text)
        if start_index == -1:
            raise IndexError(f'Start index of text "{text}" could
    not be found in title "{title}"')
        return start_index

    def __repr__(self):
        return f'Entity("{self.text}", {self.label.name}, ({self.
    start_index},{self.end_index}))'

@dataclass
class Fragment:
    text: str
    entities: list[Entity]

    def entities_overlap(self) -> bool:
        local_entities = [entity for entity in self.entities]
        entity_to_check = None
```

```
        while len(local_entities) > 0:
            entity_to_check = local_entities.pop(0)
            for entity in local_entities:
                if (entity.start_index < entity_to_check.
    start_index < entity.end_index
                        or entity_to_check.start_index < entity.
    start_index < entity_to_check.end_index
                        or entity.start_index < entity_to_check.
    end_index < entity.end_index
                        or entity_to_check.start_index < entity.
    end_index < entity_to_check.end_index):
                    return True
        return False

@dataclass
class PredictionResult:
    accuracy: float | None
    precision: float | None
    recall: float | None

    entity_accuracy: dict[str, float]

    text: str
    entities: list[Entity]
    predictions: list[Entity]

class EntityLabel(Enum):
    CLT = 'Client',
    LOC = 'Location',
    MST = 'Scale',
    CLOC = 'Creation-place'
    DATE = 'Date'
```

Listing 1: Classes used to make the models interopable

In addition, an abstract model class has been developed to provide model-independent functionalities, such as prediction evaluation based on an input fragment and a prediction result. Furthermore, a dedicated data loader has been implemented to convert the CSV file containing training and testing data into fragments. The streamlined design ensures that the coding effort required for a specific model is minimized. Only the abstract methods for loading a model, training, predicting, and testing need to be implemented. Complementing this, a small amount of 'glue code' is necessary to translate the fragment into the expected data format of the respective model. This modular approach promotes code reusability and safes time testing different models.

9.2 Data

All data is stored in a csv with tab characters as the delimiter, because sentences often contain commas. The training data has the following columns:

1. sentence
2. CLT (client)

3. LOC (location)
4. MST (scale)
5. CLOC (creation location)
6. DATE (creation date)

The model evaluation file has the following columns:

1. title (string)
2. accuracy (float)
3. entity_accuracy (dictionary with entities as the key and the accuracy as value)
4. predictions (list of entities)
5. targets (list of entities)

9.3 Stamp removal

The stamp removal is implemented with template matching from the openCV python package (OpenCV-Team, 2023). The images are loaded and converted to grayscale. A margin of 60 pixels is added to the bottom of the mask to blank out the cut off part of the stamp, additionally a margin of 10 pixels is added around the stamp to avoid black lines from the stamp showing up. The template matching algorithm prived by openCV returns a score for every coordinate of the image, therefore the biggest is chosen as the possible location of the stamp. If this value is bigger than the certainty_threshold (default is 0.32) then the stamp is removed, if the value is lower it is assumed that no stamp exists on the image and the input image is returned unmodified. To remove the stamp six pixels around the stamp are sampeld and the average of these pixels is used to paint over the stamps location. This averaging is used because not all plans do have a very light background and this avoids sharp differences in contrast which the OCR model could mistake for a character. Finally the edited image is returned.

9.4 K-Means clustering

The clustering algorithm is dependent on the number of bounding boxes identified. If the number of boxes is below three, they are all treated as one group and returned. If the number of boxes is between 3 and 5, k-means is calculated with 2 and 3 as the value k, if there are more boxes, k-means is calculated with all numbers between 5 and 12 as the value k. After that, the shiluette score for all values of k is calculated and the groupes with the biggest score are returned. The k-means algorithm is provided by the scikit-learn pyton package ("A set of python modules for machine learning and data mining", 2023)

9.5 Box ordering

After the clustering finds related boxes they need to be ordered correctly. This is achieved by first finding all boxes in the same row: All boxes are compared and they are determined to be in the same row if the y-axis center of the first box is between the highest and lowest point of the other box. After the rows of boxes are created, they are ordered left to right by their left most x coordinate.

9.6 Data generation

The method generate_ner_training_data can generate an infinite amount of training sentences. The manually written strings are stored in a dictionary with the corresponding entities and stored in a list.

```
[{'template': 'Garten $CLT in $LOC', 'tokens': ['$CLT', '$LOC']}]
```

Listing 2: Template dictionary

The method calls the corresponding generator method for each entitiy in the template. Depending on the entity this method just returns a string from a list or formats it in a certain way like the scale generator does:

```
def mst_generator():
    buffer = []
    templates = [
        'Masstab $MST',
        'Masstab: $MST',
        'Mst: $MST',
        'M. $MST',
        'M: $MST',
        'M. = $MST',
        'M = $MST',
    ]
    while True:
        if len(buffer) > 0:
            mst = buffer.pop()
            if mst:
                yield random.choice(templates).replace('$MST', mst)
    , mst
            continue
        buffer = get_list_from_csv('../../data/ner/training_data.
    csv', 'MST', delimiter='\t')
```

Listing 3: scale generator method

9.7 Testing

A set of unit tests were written for the evaluation method and the transformer base model. These two components are crucial for the whole project and changed several times. These tests made refactoring easier by avoiding regressions.

9.8 Repository

All code is publicly available on GitHub:
https://github.com/kevinloeffler/ASLA-AI

10 Hardware

10.1 DGX-2

Fortunately, the OST granted me access to their high-performance computing hardware. The DGX-2 is a cutting-edge, high-performance computing system designed by NVIDIA, specifically engineered to accelerate AI and deep learning workloads. It is equipped with 16 Tesla V100 Tensor Core GPUs interconnected with NVLink technology, providing a combined 2 petaflops of processing power. The DGX-2 proved invaluable for rapid iteration and testing of various models, training sets, and parameters. At OST, the DGX-2 operates with Apptainer, a container environment akin to Docker but built for HPC. Notably, Apptainer images do not demand root access, distinguishing them from Docker images that typically run with elevated privileges. To accommodate my specific needs, I crafted a custom Apptainer image for Python 3.10. This was necessary as all prebuilt images utilized Python 3.8, which lacked support for the type hinting syntax I did want to use.

10.2 Training

All models where trained using a single GPU but often I trained multiple models in parallel. Depending on the training data and model size, training took between 5 and 45 minutes per model. I trained between 100 and 700 steps depending on the training size and recorded all runs with help of the Weights & Biases framework.

11 Conclusion

11.1 Requirement evaluation

The main goal of this paper was to find suitable OCR and NER models and build a proof of concept that could extract five different entities from an image of a historic plan. This was achieved. The defined requirements in chapter 3 are evaluated as following:

- **The system should find related groups or blocks of text in scanned images**
  The system can find all relevant words reliably and the clustering works well.
- **The system should be able to extract characters and words from the identified groups of text**
  The system can extract the text groups into strings, the prediction accuracy depends heavily on the legibility of the text. The performance can and should be improved with fine training on custom data and preprocessing of the images.
- **The system should extract a set of predefined entities (client, location, scale, date, creation location) from the text**
  The NER model works great on simulated (clean) data and is passable on OCR output. The model struggles with sentences that are random text on plans where it tries to predict nonexistent entities. This tendency to overpredict (false positives) should be corrected in the future, on the one hand with a better OCR pipeline and on the other with a more robust NER model that is also trained on sentences without any entities.
- **The system should support JPEG and TIFF formats**
  The system accepts all commonly used image formats including JPEG and TIFF.
- **The system should ignore irrelevant archival data (e.g. stamps)**
  The stamps are reliably filtered out with template matching, resulting in better performance for the whole pipeline.
- **Correctly identifying more relevant entities, such as the client and location, is more important than scale or the creation location**
  The system achieves the best accuracies on the client entity, followed by location and scale, prioritizing the important entities.
- **Although speed is not a major concern in this paper, the whole pipeline should respect the constraints of a future application**
  The system does not achieve this requirements, on images with more than 10 text boxes (most), the whole pipeline does take more than 20 seconds. The most time intensive part is the text extraction with the trOCR model.

This could be improved by filtering the bounding boxes based on confidence befor handing them to the OCR model, or using a faster OCR model like EasyOCR which would sacrifice accuracy.

- **Compiling a comprehensive comparison and analysis of different models and architectures**
  The paper thouroughly tests several different models. The code is written very modular so that more and new released models could be trained and tested with little extra code.

- **Based on the comparison, finding a suitable model architecture for OCR and NER**
  The paper proposes an architecture that is a proof of concept for the given task.

11.2 Pipeline performance

During the comprehensive testing of the entire pipeline on various test images, an **F1-score of 0.4791** was attained when evaluating the predicted entities. The precision of the pipeline is calculated as 0.3484, with a recall of 0.7666. These scores fall notably below the NER model's performance on clean data. The primary challenge for the pipeline lies in the NER model excelling in sentences containing entities (high recall) but struggling with those without any entities (low precision). Depending on the specific plan, these non-entity sentences can vary significantly in frequency. Enhancing precision in this context could be achieved through additional NER training sentences that lack entities, thereby training the model to identify them. However, this poses a more intricate problem, as non-entity sentences lack a discernible pattern compared to their entity-containing counterparts. An alternative strategy involves acquiring training data for the layout model to enable the identification of landmarks such as the plan header or architect/date block. This would enable the OCR model to selectively predict sentences with these landmarks and disregard others, potentially offering a more effective solution.

A less significant yet more straightforward improvement involves filtering the architect's name and office. The NER model, particularly German BERT, has a tendency to misclassify the architect's name as the client. Given that all plans within a specific estate originate from the same architect, an approach would be to manually input these names at the start of a project and subsequently exclude them from the NER prediction result. If these predictions were ignored the F1-score would increase by 6% from 0.47 to 0.53.

11.3 Spellchecker

The implemented spellchecker exhibits limited utility. Because many crucial entities are not found in standard dictionaries, it cannot be effectively applied after the OCR model, as this would risk erroneously replacing names and locations. In response to this challenge, I chose to deploy the spellchecker after the NER

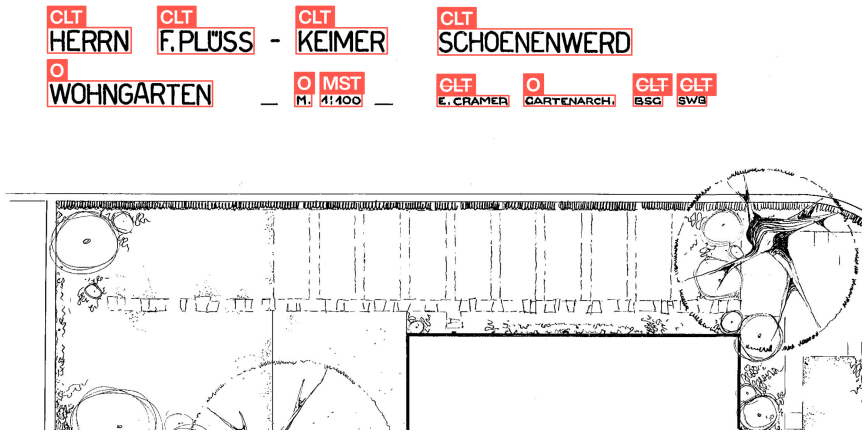


Figure 7: Visualization of the pipeline output

model predicts the labels. By doing so, I am able to selectively ignore words labeled as "client" and "location." The exploration of a spellchecker optimized for the use with OCR outputs could be an interesting research project.

11.4 Future work

In the future I want to improve the AI pipeline explored in this paper and implement it into a app that the archive can use to digizite the plans. By working with the app the archive can gather the currently missing training data over the next years, allowing further retraining in the future.

References

Asla, archiv schweizer für schweizer landschaftsarchitektur. (n.d.).

Barrus, T. (2022). *Pure python spell checker based on work by peter norvig* [Python pip package]. https://pypi.org/project/pyspellchecker/

Chan, B., Möller, T., Pietsch, M., & Soni, T. (2019). *Open sourcing german bert model*. https://www.deepset.ai/german-bert

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Historisiertes gemeindeverzeichnis der schweiz, txt format. (n.d.). https://dam-api.bfs.admin.ch/hub/api/dam/assets/23886071/master

Jaded-AI. (2020). *Easyocr: Ready-to-use ocr with 80+ supported languages and all popular writing scripts.* [GitHub]. https://github.com/JaidedAI/EasyOCR

Li, M., Lv, T., Chen, J., Cui, L., Lu, Y., Florencio, D., Zhang, C., Li, Z., & Wei, F. (2023). Trocr: Transformer-based optical character recognition with pre-trained models [Beihang University and Microsoft Corporation]. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(11). https://doi.org/10.1609/aaai.v37i11.26538

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019a). Roberta: A robustly optimized bert pretraining approach.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019b). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, *abs/1907.11692*. http://arxiv.org/abs/1907.11692

Lombardi, F., & Marinai, S. (2020). Deep learning for historical document analysis and recognition—a survey. *Journal of Imaging*, *6*(10). https://doi.org/10.3390/jimaging6100110

Mohapatra, Y., Mishra, A. K., & Mishra, A. K. (2013). Spell checker for ocr [Department of Computer Science & Engineering, Orissa Engineering College, Bhubaneswar, Odisha, Pin-752050, India]. *International Journal of Computer Science and Information Technologies*, *4*(1).

OpenCV-Team. (2023). *Wrapper package for opencv python bindings* [Python pip package]. https://pypi.org/project/opencv-python/

Rosebrock, A. (2021). Using spellchecking to improve Tesseract OCR accuracy [https://pyimagesearch.com/2021/11/29/using-spellchecking-to-improve-tesseract-ocr-accuracy/]. *PyImageSearch*.

A set of python modules for machine learning and data mining [Python pip package]. (2023). https://pypi.org/project/scikit-learn/

Shetty, A., & Sharma, S. (2023). Ensemble deep learning model for optical character recognition. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-023-16018-0

Smith, R. (2007). An overview of the tesseract ocr engine. *2*, 629–633. https://doi.org/10.1109/ICDAR.2007.4376991

Wortschatz uni leipzig - top 10'000 wörter [Archiv]. (2001). https://web.archive.org/web/20160305005801/http://wortschatz.uni-leipzig.de/Papers/top10000de.txt

Xu, Y., Xu, Y., Lv, T., Cui, L., Wei, F., Wang, G., Lu, Y., Florencio, D., Zhang, C., Che, W., Zhang, M., & Zhou, L. (2022). Layoutlmv2: Multi-modal pre-training for visually-rich document understanding.

Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., & Zhou, M. (2020). Layoutlm: Pre-training of text and layout for document image understanding. http://dx.doi.org/10.1145/3394486.3403172

Xu, Y., Lv, T., Cui, L., Wang, G., Lu, Y., Florencio, D., Zhang, C., & Wei, F. (2021). Layoutxlm: Multimodal pre-training for multilingual visually-rich document understanding.