

University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

**Multi-modal document
processing**

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Bc. Václav HONZÍK**
Osobní číslo: **A21N0045P**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Softwarové inženýrství**
Téma práce: **Multi-modální zpracování dokumentů**
Zadávací katedra: **Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s datovými kolekcemi vhodnými pro multi-modální zpracování (tj. obsahující obrázky i text).
2. Prostudujte relevantní metody pro multi-modální zpracování dokumentů.
3. Seznamte se s vybranými systémy pro optické rozpoznávání znaků.
4. Na základě předchozí studie navrhnete a implementujete prototyp systému pro multi-modální zpracování dokumentů. U vytvořeného prototypu použijte k převodu obrazu na text vybraný OCR systém.
5. Funkčnost prototypu otestujte na vybrané datové kolekci.
6. Výsledky diskutujte a navrhnete další možná rozšíření.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Pavel Král, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Plzeň, May 18, 2023

Václav Honzík

Acknowledgment

I would like to express gratitude to Doc. Ing. Pavel Král, PhD. for his guidance, valuable advice, and insightful feedback throughout the creation of this thesis.

Abstract

Multi-modal document processing is an area of computer science that focuses on analyzing, understanding, and extracting valuable information from documents that contain multiple types of data. In this work, our main objective is to perform document layout analysis using both visual and textual modalities. Our approach involves the use of instance segmentation models such as Mask R-CNN, YOLOv8, or Cascade R-CNN with a LayoutLMv3 backbone. We employ the outputs of the segmentation models with multi-modal Transformers such as LayoutLMv3 or a fusion model combining German pre-trained BERT with either Vision Transformer or Swin Transformer V2.

Another contribution of this work is a newly created historical "Heimatkunde" dataset, which consists of 4,600 annotations across 329 images and is applicable for multi-modal document layout analysis as well as classification. We train our models on this dataset and are able to achieve excellent results. Therefore, we plan to integrate these models into the Porta Fontium portal.

Abstrakt

Multimodální zpracování dokumentů je oblast informatiky, která se zaměřuje na analýzu, porozumění a získávání cenných informací z dokumentů, které obsahují více typů dat. V této práci je naším hlavním cílem provést analýzu rozložení dokumentů pomocí obrazu i textu. Náš přístup zahrnuje použití modelů pro segmentaci instancí, jako jsou Mask R-CNN, YOLOv8 nebo Cascade R-CNN s páteří LayoutLMv3. Výstupy segmentačních modelů využíváme v multimodálních Transformerech, jako je LayoutLMv3 nebo ve fúzním modelu, který kombinuje německy předtrénovaného BERTa s Vision Transformerem nebo modelem Swin Transformer V2.

Dalším přínosem této práce je také nově vytvořená historická datová sada "Heimatkunde", která se skládá z 4 600 anotací na 329 obrázcích a je použitelná pro multimodální analýzu rozložení dokumentů i pro klasifikaci. Naše modely trénujeme na této datové sadě a jsme schopni dosáhnout výborných výsledků. Tyto modely budou proto reálně využity v historickém portálu Porta Fontium.

Contents

1	Introduction	1
2	Basic Building Blocks	2
2.1	Artificial Neural Networks	2
2.1.1	Multilayer Perceptron	3
2.2	Convolutional Neural Networks	4
2.3	CNNs for Image Segmentation	5
2.3.1	Common Components	6
2.4	Recurrent Neural Networks	7
2.4.1	LSTMs	7
2.5	Word Embeddings	8
2.5.1	Word2Vec	8
2.6	Transformer	9
2.6.1	Transformer Adaptations	11
2.7	Vision Transformer	12
2.7.1	Swin Transformer	12
3	Multi-modal Document Processing	14
3.1	Multi-modal Document Classification	14
3.1.1	Fusion-based Models	14
3.1.2	Transformers in Multi-modal Classification	16
3.2	Multi-modal Document Layout Analysis	16
3.2.1	Multi-modal FCNN	18
3.2.2	Graph Convolution	18
3.3	Multi-modally Pre-trained Transformers	19
3.3.1	LayoutLM	20
3.3.2	LayoutLMv3	21
3.3.3	ERNIE-Layout	23
3.3.4	GPT-4	23
4	Datasets	25
4.1	WIT	25
4.2	RVL-CDIP	25
4.3	FUNSD	26
4.4	CORD	27
4.5	VQA	28

4.6	PubLayNet	29
5	Optical Character Recognition	30
5.1	EasyOCR	30
5.2	Tesseract	31
5.3	PaddleOCR	32
5.4	docTR	33
5.5	Ocropus	33
5.6	Kraken	34
5.7	Calamari	34
5.8	Comparison	34
6	Heimatkunde Dataset	37
6.1	Dataset Classes	38
6.2	Annotation Process	39
6.3	Resulting Dataset	39
6.3.1	OCR Subset	41
6.4	Relevant Metrics	42
6.4.1	Classification	42
6.4.2	OCR	43
6.4.3	Document Layout Analysis	43
6.5	Extraction of the Text	45
6.5.1	Pre-trained Models	45
6.5.2	Fine-tuning	46
6.5.3	Results	46
7	Multi-modal Document Layout Analysis	50
7.1	Instance Segmentation	52
7.1.1	Models	52
7.1.2	Preprocessing	53
7.1.3	Training	54
7.2	Multi-modal Classification	56
7.2.1	Models	56
7.2.2	Preprocessing	59
7.2.3	Implementation	60
7.2.4	Training	61
7.2.5	Configuration and Hyperparameters	62
7.3	Multi-modal System	63
7.3.1	YOLOv8 Compatibility	63

7.3.2	Communication Between Segmentation Model and Classifier	64
7.3.3	Evaluation	64
7.4	Utilization of Textual Modality	65
8	Results	66
8.1	Instance Segmentation Results	66
8.2	Multi-modal Classification Results	67
8.3	Results of Instance Segmentation with Multi-modal Classifier	69
8.4	Baseline Performance of Textual and Visual Modalities	73
8.5	Utilization of Textual Modality	73
8.6	Possible extensions	76
9	Conclusion	77
	List of Abbreviations	80
	Bibliography	81

1 Introduction

In recent years, multi-modal document processing has become a rapidly growing area of research that involves the analysis of complex documents comprising multiple modalities such as text, images, audio, or video. Such documents can range from books and scientific papers to social media posts or medical data. This field closely follows advances in machine learning fields such as natural language processing (NLP) or computer vision (CV), speech recognition, etc.

This work focuses on the utilization of modern multi-modal techniques in order to perform document layout analysis on historical documents containing visual and textual modalities. Such a task is beneficial for further processing of the documents, as it can help to identify important parts of the document to improve text recognition or information retrieval.

The main contribution of this thesis is a model capable of multi-modal layout analysis, as well as a large document layout analysis dataset that can be used by both image-only and multi-modal models. Moreover, for text detection, we create an optical character recognition (OCR) model, which is trained to recognize text written in historical German Fraktur. The outputs of this work will be used in the historical Porta Fontium portal¹ to improve information retrieval from historical documents.

The structure of the thesis is as follows. Chapter 2 introduces essential building blocks of state-of-the-art multi-modal models, mainly related to deep learning. The theoretical background behind multi-modal document processing itself is reviewed in Chapter 3, where we present relevant approaches. Chapter 4 gives an overview of notable multi-modal datasets. This is followed by Chapter 5, which covers several OCR frameworks that can be applied to our data.

The process of annotating our historical dataset, as well as the training of the OCR model responsible for text extraction, is described in Chapter 6. We combine a state-of-the-art instance segmentation model with a multi-modal classifier, in order to perform the document layout analysis itself, which we cover in Chapter 7 and discuss our results as well as potential extensions in Chapter 8.

Finally, we conclude the thesis in Chapter 9, where we give an overview of the achieved results and propose some future directions.

¹<https://www.portafontium.eu/>

2 Basic Building Blocks

Before we introduce multi-modal document processing itself, it is important to understand the key components behind the systems processing such a type of data. Because most state-of-the-art (SOTA) solutions are based on deep learning concepts from NLP and CV, we dedicate this chapter to a brief overview of commonly used model architectures and techniques.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are multivariate statistical models that draw inspiration from biological neural networks and were originally introduced by McCulloch and Pitts. Such a model comprises a set of nodes called neurons that propagate information via their synapses.

The process of a single neuron in this model is described by Eq. 2.1. The neuron receives input as a vector of real-valued numbers \mathbf{x} and returns a scalar a . The components of the input vector are linearly weighted by weights w_1, w_2, \dots, w_n and combined, usually by an addition operation. Along with these inputs, the neuron is equipped with an independent value called bias b , which shifts the value of the scalar to the negative or positive side.

The combined result is used as an input to an activation function σ , which gives the network the ability to model complex decision boundaries. Common functions for this purpose are ReLU (Rectified Linear Unit), sigmoid, or the Swish family of functions, which are visualized in Figure 2.1.

$$a = \sigma\left(\sum_{i=1}^n w_i \cdot x_i + b\right) \quad (2.1)$$

Both synaptic weights and biases are parameters of the model that can be learned, which is typically accomplished by the backpropagation algorithm [61]. The algorithm consists of two steps - the forward pass and the backward pass.

In the forward pass, the input examples are fed through the network to calculate its output. Subsequently, in the backward pass, the output is compared to ground truth to compute an error, typically via a loss function. Such a function is typically differentiable, making it possible to compute the gradient, which can be used to update individual weights and biases of the network.

2.1.1 Multilayer Perceptron

A common type of ANN is the multilayer perceptron (MLP). This architecture consists of a sequence of fully-connected (FC) layers where each neuron in a specific layer is connected to all neurons in the previous layer.

The MLP structure includes an input layer, one (or multiple) hidden layers, and an output layer, as depicted in Fig. 2.2. Nowadays, the primary use of this model is classification tasks, which involve assigning a categorical label y to a given (continuous) input x .

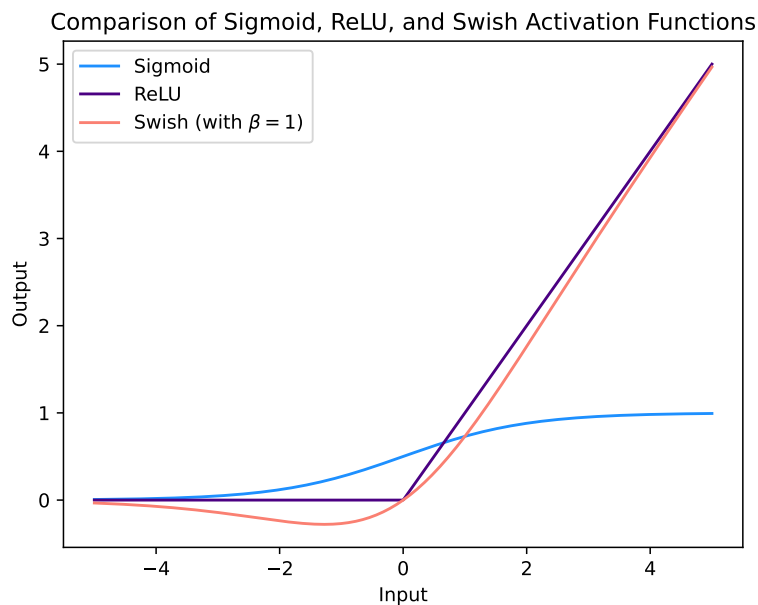


Figure 2.1: Visualization of sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$, ReLU: $f(x) = \max(0, x)$, and Swish: $f(x) = x \cdot \sigma(\beta \cdot x)$.

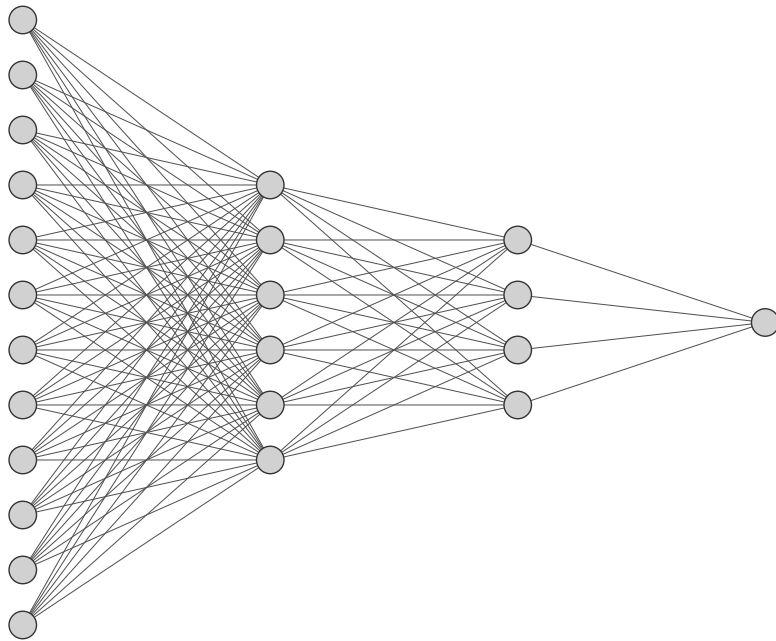


Figure 2.2: Visualization of a simple MLP with two hidden layers. The nodes in the graph represent individual neurons, while the edges constitute weights; generated via NN-SVG tool [36].

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of ANNs that use convolution operation in one or more of its layers. Such an operation applies a learnable filter (kernel) to a given input in the form of a sliding window.

The output of a convolutional layer can be one or more feature maps, based on the number of applied filters. In this context, a feature map contains extracted information such as edges, corners, or significant words in a sentence, depending on the type of data being processed. Similar to the FC layer, the feature map is passed to the activation function to introduce nonlinearity.

Due to the number of filters, the resulting output is typically much larger than the input, and processing it with subsequent layers might not be feasible. Therefore, CNNs typically use an additional type of layer called the pooling layer. The pooling layer is responsible for downsampling the amount of extracted information through operations such as minimum or maximum. This layer behaves similarly to the convolutional layer, except it uses a stride with the size of the entire filter to prevent the layer from seeing the same information twice.

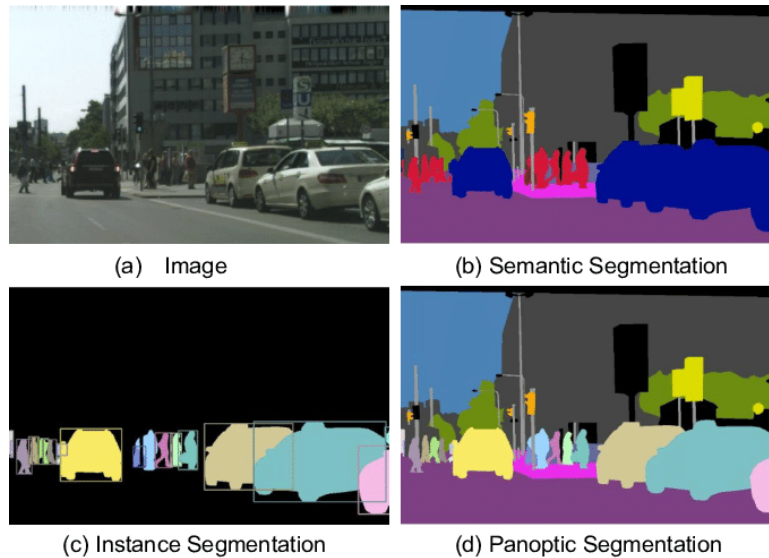


Figure 2.3: Image segmentation tasks. Semantic segmentation (b) produces a segmentation mask, however, the individual instances are indistinguishable - e.g. the vehicles on the right side blend together. On the other hand, instance segmentation (c) produces segmentations and bounding boxes for each instance. Panoptic segmentation (d) combines both approaches [35].

2.3 CNNs for Image Segmentation

A large field that is relevant to multi-modal processing is image segmentation. Due to the convolutional operations, CNNs are crucial components in this area. Image segmentation involves three types of tasks. Firstly, in semantic segmentation, the model is trained to predict labels for each pixel in the image [19]. Such a process produces a segmentation mask, which can be used to separate individual classes in the image.

Secondly, instance segmentation builds on semantic segmentation and assigns labels for separate instances that correspond to the same class [19]. The output from this task is a set of instances.

The third option, panoptic segmentation, is a combination of both semantic and instance segmentation. Essentially, the network is able to generate a segmentation mask and annotate it with instance labels to produce an unified view of segmentation [35]. The difference between each task can be seen in Fig. 2.3.

2.3.1 Common Components

There are several common techniques/layers that are exploited across many image segmentation architectures. The input of the model is a raw image that is fed to a backbone model responsible for extracting useful features for the rest of the network. The backbone can be a common model for image classification with the classification head removed, such as ResNet50 [21], VGG16 [64], etc.

Subsequently, feature maps produced by the backbone are fed to a region proposal network [71] (RPN). The responsibility of RPN is to extract regions of interest (RoIs) that may contain objects in them. The main benefit of this approach is that it is much faster than performing an exhaustive search, which is often computationally unfeasible [71].

The extracted RoIs come in different scales and aspect ratios and need to be transformed into fixed-size vectors for further processing [81]. A popular method is to use the RoIAlign layer [22], which extracts a small feature map (e.g. 7×7) from each RoI. Finally, the resulting set of fixed-size RoIs is then fed to a specific head that is responsible for detecting the object instance or segmentation mask.

These components form the architecture of Mask R-CNN [22], which is one of the most popular instance segmentation models. However, many of these concepts are present in other state-of-the-art image segmentation CNNs such as Cascade R-CNN [8]. The overall architecture of Mask R-CNN is shown in Fig. 2.4.

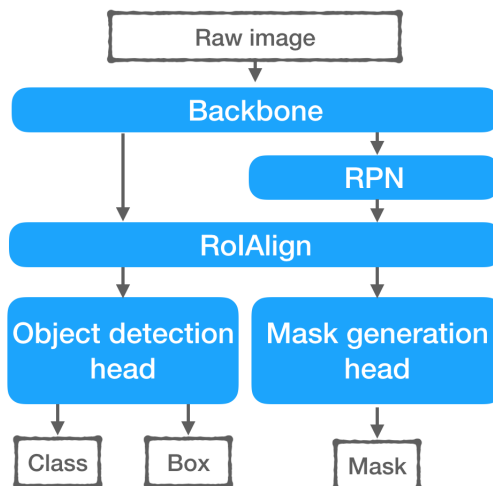


Figure 2.4: Architecture of Mask R-CNN [81].

2.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are an architecture that specializes in processing variable-length sequences of elements. Such sequential data is typically difficult to process by the aforementioned architectures (i.e. CNN and MLP) because their input size is fixed, and feeding the model the entire sequence is unfeasible.

On the other hand, RNNs process one element at a time, using internal context that influences the outputs inferred from the following elements in the sequence. Naturally, this property makes RNNs very suitable for types of data such as text or time series.

2.4.1 LSTMs

Today, Long Short-term Memory (LSTM) networks [24] are one of the most widely used types of recurrent neural networks. The architecture of LSTM tries to solve the issues present in the simpler recurrent versions, such as the Elman RNN [14], which suffered from the vanishing gradient problem. As a consequence, simple RNNs are unable to efficiently remember long-term dependencies, which is crucial for longer sequences.

To store the information about the sequence, LSTMs utilize two different types of states - the hidden state and the cell state. The hidden state is used as the output of the network, while the cell state defines which information from the hidden state is kept or discarded.

The architecture of LSTM is shown in Figure 2.5. Formally, the network accepts a sequence of fixed-size inputs $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$. At a given time step t a triplet of cell state c_{t-1} , hidden state h_{t-1} , and input element x_t is processed. Note that except for the sequence length, the shapes of x_t , c_t , and h_t are fixed-size tensors.

The cell state depends on two gates. The forget gate (F_t) decides which information from the cell state of the previous time step is kept and which is discarded. On the other hand, the input gate (I_t) decides which information is injected into the cell state computed for the current iteration. Finally, the last type of gate - the output gate (O_t) - influences the computed hidden state and thus the output of the network.

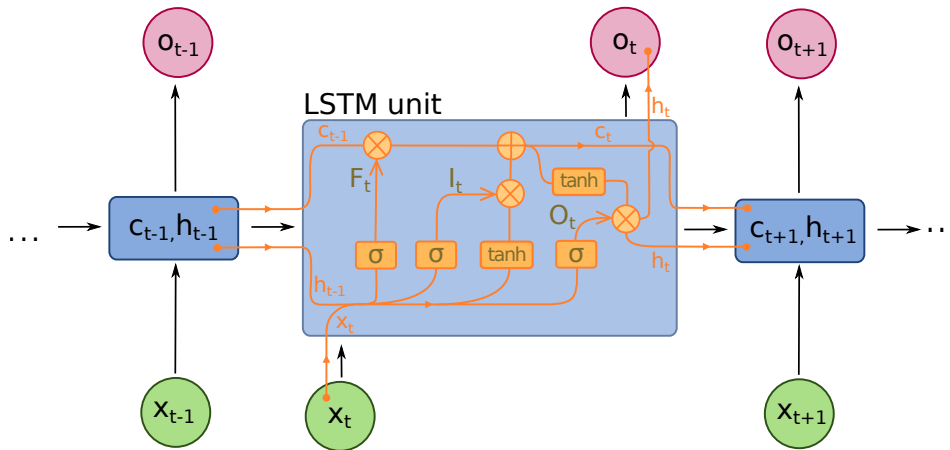


Figure 2.5: LSTM architecture [9].

2.5 Word Embeddings

In the context of NLP, the representation of words in a deep-learning model is difficult. From a human perspective, text is perceived as a sequence of words or characters. However, a neural model requires the input to be a vector of real values, making the conversion between the formats non-trivial.

One way of representing text is to use a t -dimensional vector where each element represents a distinct term, which can be e.g. useful for computing the similarity between two documents [3]. While such an approach is applicable in information retrieval, it is difficult to employ it in a deep learning context as the individual terms are represented by orthogonal vectors.

On the other hand, word embeddings are a technique that enables the representation of words as dense vectors, where similar words have similar encoding. The vectors themselves are low-dimensional compared to the number of unique words in the text (vocabulary), e.g. 300 dimensions vs 60k dimensions.

2.5.1 Word2Vec

Word2vec [47] is probably one of the most efficient word embedding approaches. It presents a simple, yet fast and effective method for extracting embeddings using a single hidden layer perceptron. Two types of models are proposed - the continuous bag of words (CBOW) and the skip-gram.

In the CBOW variant, the model is tasked with predicting a given word w_t based on its neighboring words. The skip-gram model, on the other

hand, uses an inverse task where it is given a specific word and predicts its surrounding context. Both architectures can be seen in Fig. 2.6.

Using these methods and many training iterations, the word vectors can be extracted as the output of the hidden layer in the network. Such generated embeddings can also be used to find semantically similar words using simple algebraic operations [47], which can be useful for example for information retrieval.

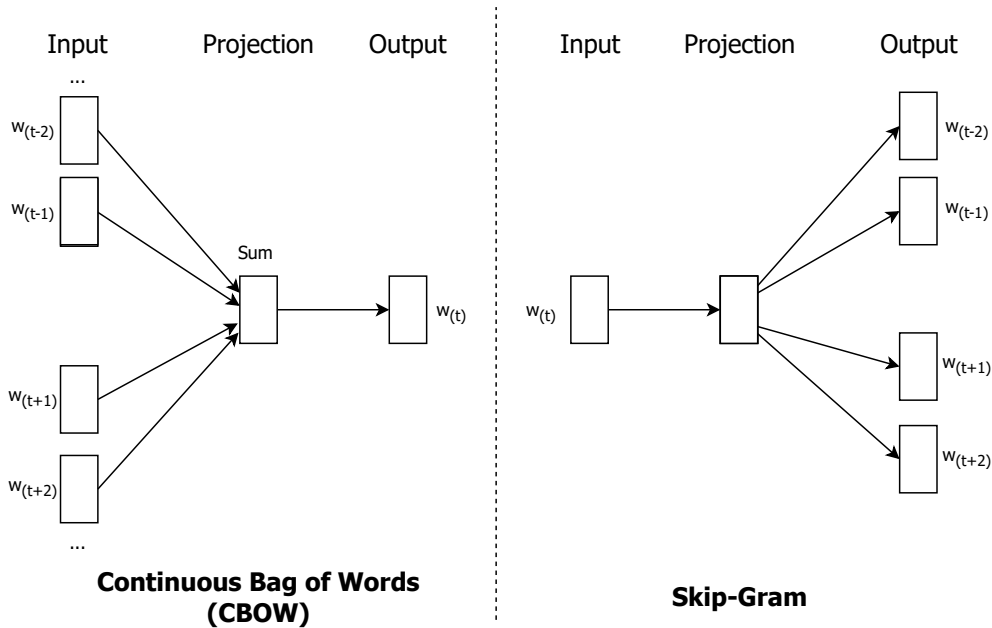


Figure 2.6: Word2Vec - CBOW and Skip-gram model architectures.

2.6 Transformer

The Transformer, introduced in the paper *Attention is all you need* [72], is probably one of the most influential deep learning architectures of the last decade. It is a sequence-to-sequence model¹ originally designed for language translation (English to German) but has since been adapted to many other NLP and CV tasks.

The Transformer consists of two main components - an encoder, which converts the source language into a set of features, and a decoder, which is able to process these features and output a sequence in the target language.

The key concept introduced in both the encoder and decoder blocks is the attention mechanism, which can be used to measure relevance or importance.

¹A model that takes a sequence as input and produces a sequence as output, which may be of different lengths.

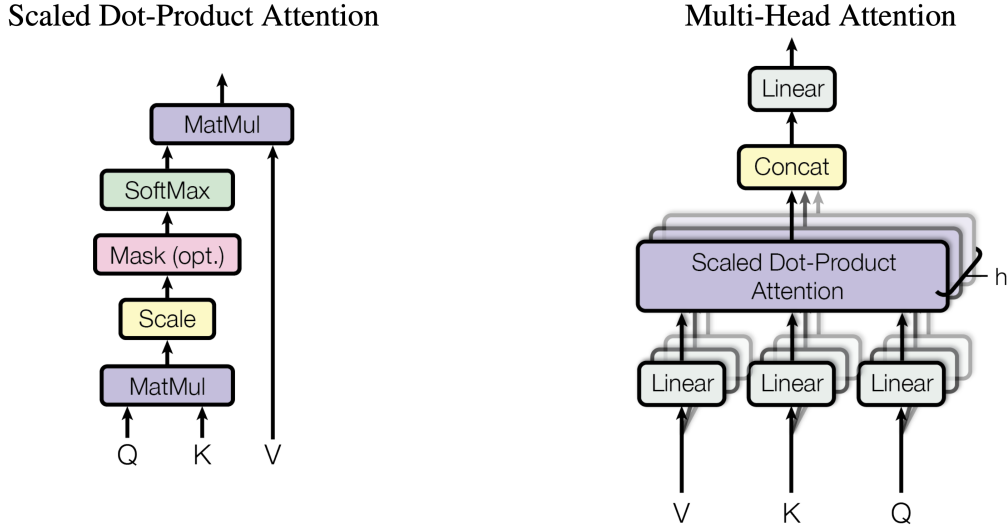


Figure 2.7: Self-attention and multi-head attention mechanism [72].

Specifically, the Transformer uses the scaled dot-product attention, which can be seen in Eq. 2.2 and on the left side in Fig. 2.7.

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}}\right) \cdot \mathbf{V} = \mathbf{A} \cdot \mathbf{V} \quad (2.2)$$

Scaled dot-product attention accepts input in the form of queries (\mathbf{Q}), keys (\mathbf{K}), and values (\mathbf{V}), which can be computed by a set of trainable linear layers. The computed matrix \mathbf{A} is called the attention matrix [40] and it represents a weighted sum of values \mathbf{V} . The most intuitive form of attention is self-attention, where \mathbf{Q} , \mathbf{K} , and \mathbf{V} are computed using the input embeddings of the sequence or the output of the previous attention layer.

The concept of attention can be easily transformed to multi-head attention, as shown in Figure 2.7 on the right. A head, in this context, constitutes an individual application of the attention mechanism and is independent of other heads. The output from each head is concatenated and fed into a linear layer, which transforms it back into the original dimension of the value matrix \mathbf{V} .

Unlike RNNs, which process one element of the sequence at a time, most of the operations in multi-headed attention are trivially parallelizable, leading to a significant speedup in both inference and training. This is because the attention is computed for each word separately as reflected in the $\mathbf{Q}\mathbf{K}$ matrix multiplication.

The overall architecture of the model is shown in Figure 2.8. As mentioned before, the input sequence is mapped to word embeddings (similar

to e.g. Word2Vec), which are positionally encoded to force the model to consider positional information in the sequence. Additionally, to improve the gradient flow, Add & Norm layers are applied after each operation, using residual connections analogous to ResNet [21] and layer normalization analogously to [4]. To introduce non-linear transformations, each block uses position-wise fully connected layers.

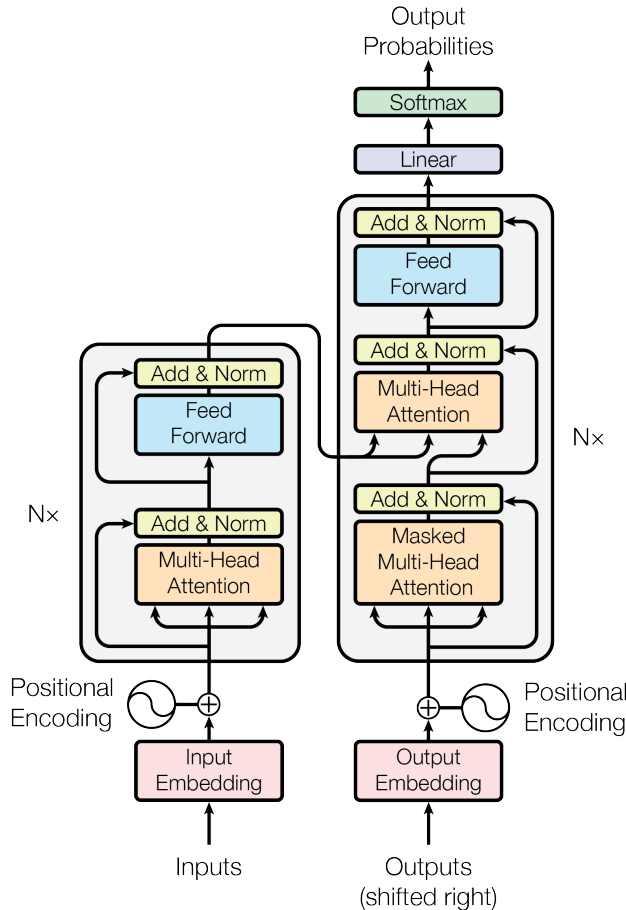


Figure 2.8: Architecture of the Transformer. The encoder (left) and the decoder (right) are repeated up to n times. Part of the output from the final encoder block (values and keys) is fed to the multi-head attention in the decoder, which uses it to generate the translation one token at a time [72].

2.6.1 Transformer Adaptations

It is important to note that both the encoder and the decoder components of the original architecture can be used independently. Among the best-known encoder models is BERT [12] (Bidirectional Encoder Representations from Transformers), which can be used for a variety of classification tasks such as

document classification [1]. On the other hand, GPT [58] (Generative Pre-Training for Transformers) is a decoder-only model that has found immense success in generative tasks such as text generation.

Many of these models are very large (100M - 500B+ parameters) and require massive amounts of data for training (GBs - TBs in size). However, the main advantage is that such pre-trained models can often be fine-tuned for a specific task, requiring only a few training iterations.

2.7 Vision Transformer

Vision Transformer (ViT) [13] is a model inspired by the original Transformer, but instead of text, it focuses on processing visual features. Unlike conventional image processing models, which typically rely on convolutional operations to extract features, ViT replaces their functionality with multi-head attention layers. The model's architecture is largely similar to BERT, consisting of n encoder blocks stacked on top of each other, with the input to the network being embedding vectors.

Analogous to word embeddings, patch embeddings are continuous vector representations of the parts of the image. To obtain them, a patch embedding layer is trained, which encodes a flattened version of the specific image patch into a corresponding embedding vector. The added benefit of this approach is that the model can handle arbitrary sequence lengths [13], which is difficult to achieve in conventional CNNs. The overall architecture is shown in Figure 2.9.

2.7.1 Swin Transformer

Analogous to the original Transformer, ViT has also been an inspiration for newer models, such as the Swin Transformer [43]. The Swin Transformer model introduces hierarchical feature maps by merging image patches in deeper layers [43], which can be seen in Fig. 2.10. Additionally, it only computes self-attention only in the local patch, reducing complexity from quadratic to linear.

Swin Transformer V2 [44], the latest iteration of this architecture, introduces additional improvements, for example, scaled cosine attention instead of conventional self-attention, higher resolution of the feature map, or enhanced pre-training. The model achieves SOTA performance on several image datasets such as MS COCO [41] or ImageNet-V2 [59].

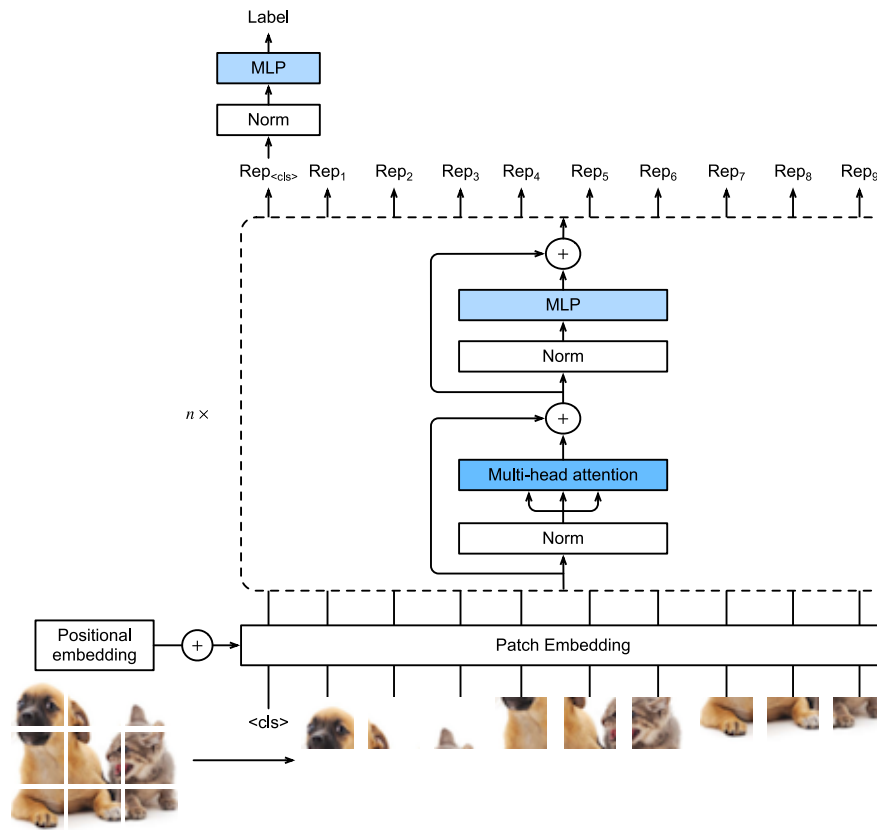


Figure 2.9: Architecture of the ViT model. The model comprises an embedding layer, n Transformer encoder blocks, and a task-specific head (in this case for classification) [79].

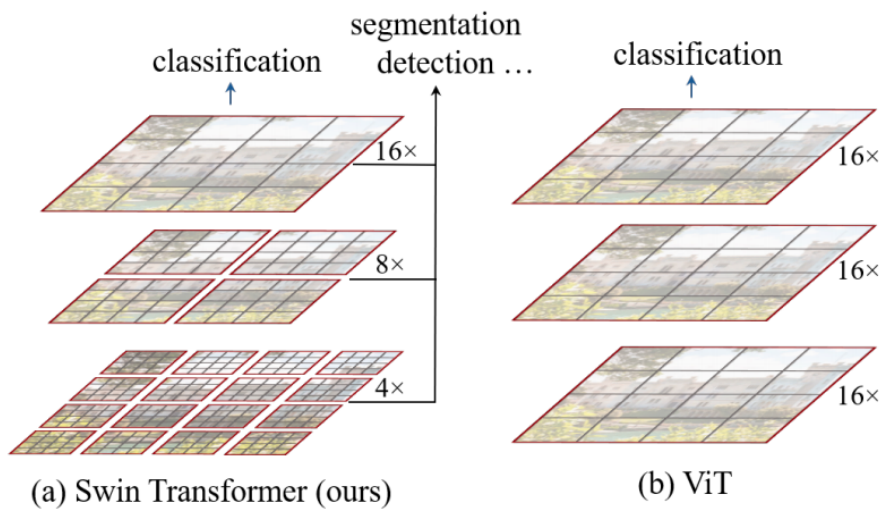


Figure 2.10: Comparison of Swin Transformer and ViT [43].

3 Multi-modal Document Processing

Traditional document processing is mainly concerned with text-only documents and various operations on them, such as digitization, indexing, summarization, etc. However, more and more documents today contain multiple sources of information that can be equally important.

To gain a better understanding of such documents and to extract useful information, it is necessary to process them in a multi-modal manner. Such a process often involves combining state-of-the-art approaches from different areas of machine learning such as computer vision or natural language processing.

The following chapter introduces key concepts used in multi-modal document processing as well as the state-of-the-art architectures applicable to common tasks. We primarily cover models that process images and text, as these are the types of data processed in the thesis, but many of the techniques can be adapted to other modalities such as audio or video.

3.1 Multi-modal Document Classification

Multi-modal document classification is the type of classification task where several modalities are used to predict the target label or a set of labels. Such a task has many applications ranging from analysis of social media posts, classification of news and medical records, fraud detection [55], or deep fake detection [73].

3.1.1 Fusion-based Models

The way in which the given modalities are employed differs from paper to paper as well as from the given data being classified. Arguably, the most intuitive option is to use state-of-the-art networks for each modality and merge their outputs. This technique is commonly referred to as fusion.

For image-text documents, this typically involves the use of deep convolutional networks such as InceptionV3 [70], VGG16 [64], or vision-based Transformers to process the visual part, and a Transformer encoder model such as BERT [12] to process the textual modality. These models are often not trained from scratch, as there are usually not enough samples in the

downstream task dataset, and are instead pre-trained on large datasets and subsequently fine-tuned.

A relatively simple, but in many cases effective, approach is to use a linear combination of the output probabilities from each network. For instance, the following work [17], which uses EfficientNet (for visual information) and BERT (for textual information) on the Small-Tobacco and Big-Tobacco image datasets, defines normalized weights (w_1, w_2) for each modality as shown in Eq. 3.1.

The resulting probability can be used to predict the class number via the argmax function in Eq. 3.2 identically to standard unimodal classification. Note that in this configuration, the weights are a hyperparameter and are not optimized during learning. Analogously, it is also possible to extend this equation with additional weights if we have more than two modalities.

$$P(\text{class}|\text{out}_{\text{image}}, \text{out}_{\text{text}}) = w_1 \cdot P(\text{class}|\text{PredictionText}) + w_2 \cdot P(\text{class}|\text{PredictionImage}) \quad (3.1)$$

$$\text{PredictedClass} = \underset{\text{class}}{\operatorname{argmax}} P(\text{class}|\text{out}_{\text{image}}, \text{out}_{\text{text}}) \quad (3.2)$$

The second approach to processing modality outputs is to use their concatenation. The outputs of individual networks are combined into a single vector, which is then fed into a classifier model. Such a model can be as simple as a perceptron network (e.g. with a single hidden layer) and its only goal is to carry out the classification.

This method is used, for example, in [18]. The paper uses BERT and InceptionV3 to perform classification on the Food101 dataset. The authors use two variants of building the fused vector. Late fusion uses class probabilities for each modality as input to the classifier, while early fusion uses features from the last hidden layer of each network. Schematics for both approaches can be seen in Figures 3.2 and 3.1.

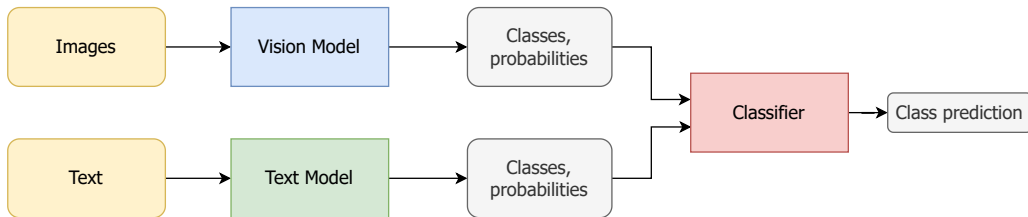


Figure 3.1: Late fusion when using text and image modalities.

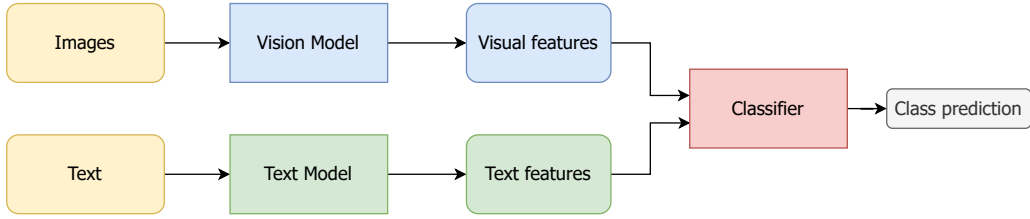


Figure 3.2: Early fusion when using text and image modalities.

Arguably the main advantage of such a fusion approach over the previously mentioned linear combination approach is that we can optimize the weights of all models at once instead of training the networks individually. This should theoretically allow the network to learn from the entire context of the data. The use of early and late fusion differs in each paper using such a technique. The previously mentioned paper [18] reports better results with early fusion. In contrast, other papers such as [11] only use late fusion.

3.1.2 Transformers in Multi-modal Classification

While the original Transformer is only applicable to text processing, since then, similar architectures have been employed to handle other modalities such as the Vision Transformer, which we overview in Section 2.7, the Video Transformer [49], Hubert [26] for audio, or many others.

These models can be treated as black-boxes and used with the aforementioned fusion or weighting methods, and there is also a lot of ongoing research on encoding all modalities into embeddings and passing them directly to the multi-modal Transformer. Classification using the latter approach is performed identically to the Transformer encoders used for text, and we cover this in detail later in Section 3.3.

Note that in this context, the label can either be assigned to the entire multi-modal sequence - i.e. assign a certain class to the document, or assign a class to each of the elements in the sequence. These elements can be, for example, text tokens or parts of the image. The latter is often used in tasks such as named entity recognition, semantic entity labeling, or document layout analysis.

3.2 Multi-modal Document Layout Analysis

Document layout analysis (DLA) is a task concerned with the identification of the regions of interest in a given document as well as their roles [80]. RoIs represent the components of the document such as paragraphs, headings,

images, tables, page numbers, etc. The purpose of this task is mainly auxiliary, as it can provide information that is helpful for further processing of the document - e.g. we may prefer different techniques/models for processing images, body text, footers, and so on. An example output of DLA can be seen in Fig. 3.3.

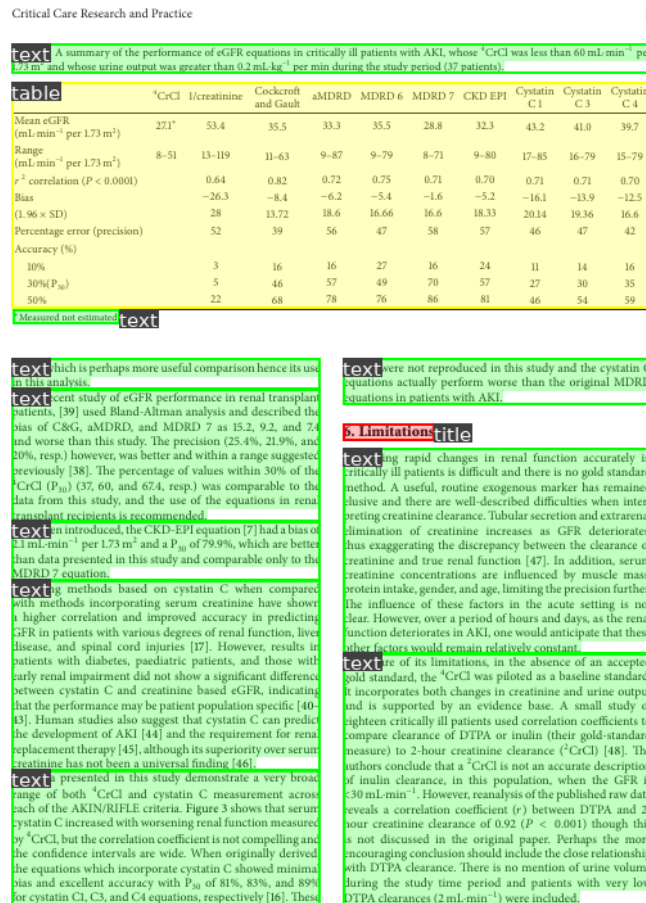


Figure 3.3: Example output of document layout analysis - each component is denoted by a colored bounding box [82].

Unimodal DLA mainly utilizes image features because they contain the most information about the layout. For this purpose, we can use various image segmentation networks such as YOLO [33], Mask R-CNN, and many others.

In a multi-modal approach, it usually makes the most sense to also employ the textual modality, which can be analyzed to extract semantics that

could further help in distinguishing between the type of ROI, for example. According to [80], multi-modal methods can be divided into two categories - CV-based, which treat the task as object detection or instance segmentation, and NLP-based, which works on words or other low-level elements.

3.2.1 Multi-modal FCNN

One of the first works combining modalities [78] for DLA uses a fully convolutional network for both text and image features. The image data is fed through a series of convolutional layers, while the text is converted into a sequence of embeddings and combined with the processed image features at the last convolutional layer. The employed embeddings represent an entire sentence, as an average of all word embeddings, rather than a single word. The architecture of the network is shown in Fig. 3.4.

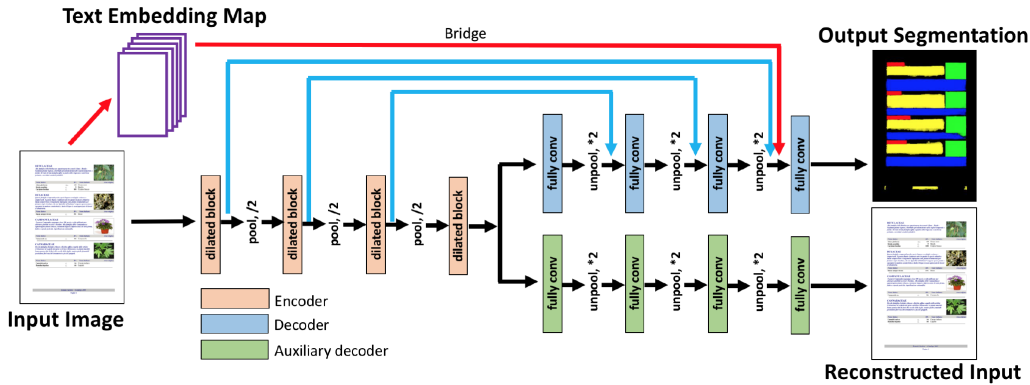


Figure 3.4: Multi-modal FCNN architecture for document layout analysis [78].

3.2.2 Graph Convolution

Another solution is based on the so-called graph convolution [42], which performs NLP-based layout analysis. The general idea is to represent the data in a graph form that can be mapped to embedding vectors. The work constructs the embeddings from visual features of the text, which can be of two types - vertex and edge. Vertex embeddings are computed using a single-layer BiLSTM¹, while edge embeddings are computed from the distance, width-to-width, width-to-height, and height-to-height ratios of the two text boxes of the corresponding vertices.

¹BiLSTM is an LSTM that processes an input sequence from both directions and concatenates the output into a single vector/tensor.

The embeddings are then packed to vertex-edge-vertex triplets and contextualized using the convolution operation - see Fig. 3.5. The result is combined with conventional token embeddings and fed through the network, which is a BiLSTM-CRF model as depicted in Figure 3.6. In this context, CRF stands for conditional random field and it is a block that should improve the accuracy of the classification [28].



Figure 3.5: Contextualization of vertex (t_i) and edge (r_i) embeddings using a convolution operation - in this case, a multi-layer perceptron [42].

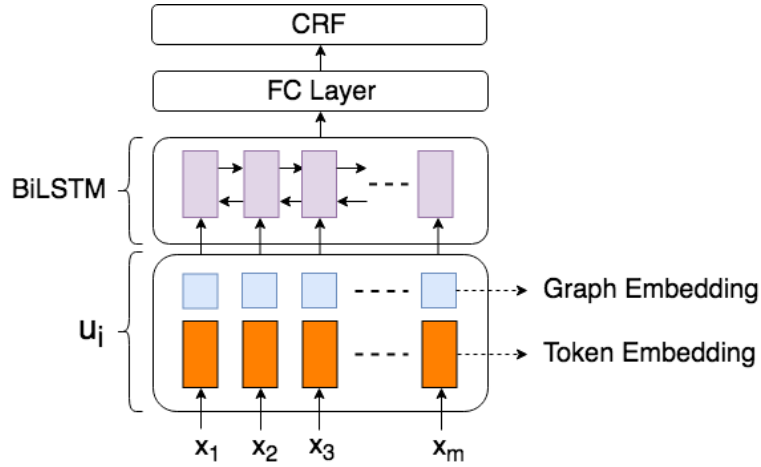


Figure 3.6: Multi-modal BiLSTM-CRF graph-based network architecture [42].

3.3 Multi-modally Pre-trained Transformers

The last section of this chapter is dedicated to multi-modally pre-trained Transformers that were briefly mentioned in Section 3.1.2. There are many advantages to using these models. Most of the multi-modal models mentioned above rely on supervised learning, which usually requires a very large

number of samples to produce good results. However, in many cases, the processed data is either scarce or expensive to generate, which severely limits the application of the model in practice.

On the other hand, (multi-modal) Transformers commonly utilize pre-training on large self-supervised datasets and can be fine-tuned for custom datasets via transfer learning. In addition, such a model is typically capable of performing multiple tasks that require little modifications, whereas conventional deep-learning approaches may involve the use of a completely different architecture.

3.3.1 LayoutLM

LayoutLM [77] is one of the most influential models in the field of multi-modal document processing. The architecture of this network is heavily inspired by BERT, however, it modifies the BERT structure and its pre-training tasks to work better with document layout understanding.

Since a typical Transformer network expects input in the form of token embeddings, the model uses an OCR system for text extraction. The text is mapped to word embeddings and adjusted using 2D position embedding, which aims to help the model better contextualize the location of the word in the sequence. The 2D position embeddings contain the coordinates of the top left and bottom right corners of the bounding box of each token.

The processing of the image part is carried out by the Faster R-CNN [60] object detector. The image is split into segments, one for each bounding box. Subsequently, each segment is fed to the object detection model, which extracts important features that get transformed into embeddings by a sequence of linear layers. Finally, both visual and textual embeddings are summed and passed to the output layer. The whole process can be seen in Fig. 3.7.

The pre-training of LayoutLM comprises two tasks - *Masked Visual-Language Model* (MVLM) and *Multi-label Document Classification* (MDC). Both tasks are performed on the IIT-CDIP dataset [29]. MVLM is analogous to the Masked Language Model in BERT but the goal is to learn to model the language in relation to both context and position. Therefore, the model is given all 2D position embeddings of the sequence with only some of the token embeddings available, while the rest is masked and trained upon.

The second task, MDC, is a classification task that aims to train the network to generate document-level representations. As ground truth, the tags for each document are used and compared with the prediction of the network. Note that this task is inherently a form of supervised learning since

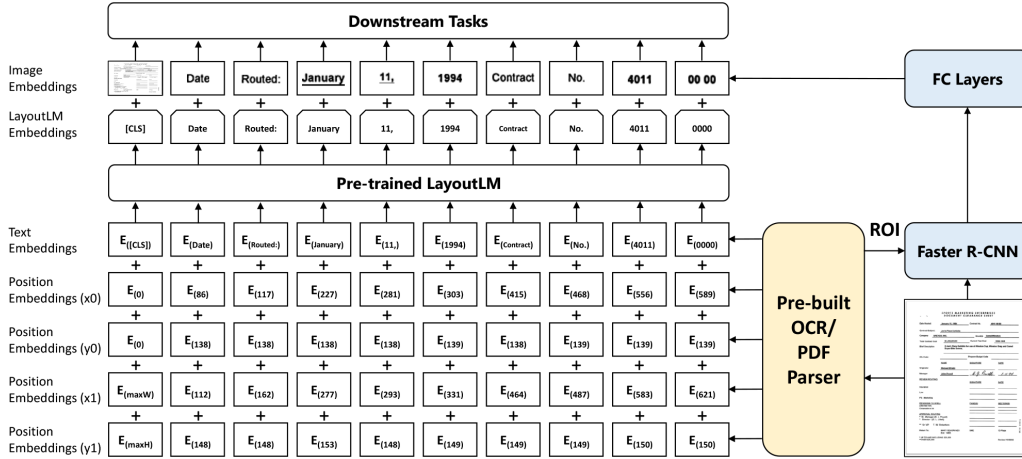


Figure 3.7: Architecture of the LayoutLM model [77].

we need some kind of label to compute the loss. Therefore, the authors mention that it is optional, as it may be unavailable for larger datasets should the model be pre-trained on them.

In general, the model can be fine-tuned for three types of tasks - visual question answering, token classification, and sequence classification.

3.3.2 LayoutLMv3

We are going to skip LayoutLMv2, as most of its features are retained in the latest iteration of the model - LayoutLMv3 [27]. LayoutLMv3 uses a fully multi-modal Transformer network - i.e. no CNN such as Faster R-CNN is used to produce visual features. This helps the model to learn cross-modal information [27], which should result in better performance than its previous versions.

Both textual and visual information is encoded in the form of embeddings. For text, this is the same as the previous versions while the image input uses patch embeddings in a similar way to ViT - i.e. by splitting the image into uniform fixed-size patches and embedding them via pre-trained FC layer.

Regarding position encoding, LayoutLMv3 uses 2D position embeddings and 1D position embeddings. 1D position embeddings are used to denote the index of the token or patch in the sequence, while 2D position embeddings encode the position of the bounding box in the layout.

The main change in the 2D embeddings compared to previous versions is the fact, that we use a bounding box per segment, instead of a bounding box per individual token - since the words in the same segment should have

the same semantic meaning [38]. The architecture of the network is shown in Fig. 3.8.

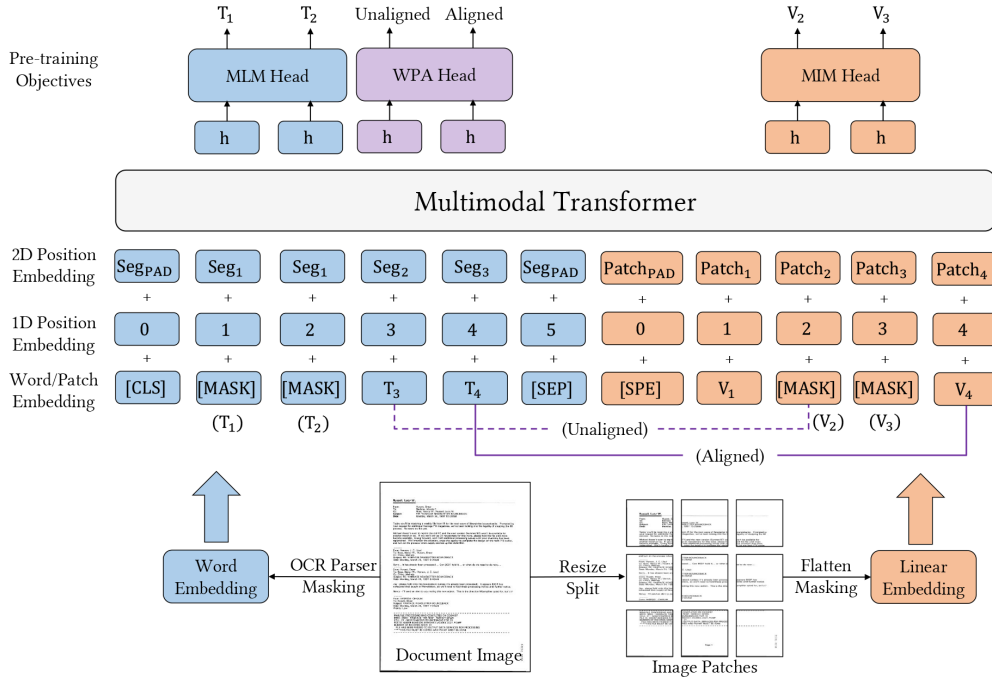


Figure 3.8: Architecture of the LayoutLMv3 model [27].

LayoutLMv3 uses three pre-training tasks - *Masked Language Modeling* (MLM), which is heavily inspired by MVLM in the original LayoutLM, *Masked Image Modeling* (MIM), and *Word-Patch Alignment* (WPA).

MLM masks 30% of the text tokens while the model tries to predict them. MIM, on the other hand, is concerned with image modality and instead masks 40% of the image tokens - forcing the network to predict them. Finally, in the WPA objective, the model is tasked to decide whether the corresponding word and its image patch are aligned or unaligned. All three pre-training tasks use negative log-likelihood loss and the total loss during the training is computed as the sum of the losses from each objective.

Such architecture and pre-training allow LayoutLMv3 to achieve state-of-the-art results in several datasets, such as FUNSD, CORD, DocVQA, or PubLayNet, which we cover in Chapter 4. This makes the model viable for document classification, token classification, visual question answering, and even document layout analysis.

3.3.3 ERNIE-Layout

A viable alternative to LayoutLMv3 is ERNIE-Layout [57]. This model is based on ERNIE [69] or *Enhanced Representation through kNowledge IntE-gration*, which is a Transformer-based encoder similar to BERT.

The principle of the model is analogous to LayoutLM with a few differences. The input to the Transformer is a text-image pair. Specifically, the text is extracted via an OCR tool and arranged in a proper reading order by an advanced document layout analysis toolkit called Document-Parser. The network itself is based on DeBERTa [23] and uses spatial-aware disentangled attention to enable the effect of layout features [57].

The pre-training of the model consists of four tasks. In *Reading Order Prediction*, the model is tasked to predict the boundaries of text segments in a sequence of words that is ordered in the correct reading order. Similar to LayoutLM, ERNIE-Layout also uses the MVLM task. In the third task - *Replaced Region Prediction*, a portion of the image patches are replaced (10%) with patches from another image, and the network learns to recognize them via [CLS] tokens.

The fourth task is called *Text-Image Alignment* and aims to train the model to understand the spatial relationship between image regions and bounding box coordinates. This is achieved by selecting random text lines from the input along with their corresponding regions. Such lines are covered and the model is tasked to predict whether a specific token corresponds to the covered line.

ERNIE-Layout is applicable to the same set of tasks as LayoutLM - i.e. document VQA, sequence classification, and token classification. The performance of the model is comparable to LayoutLMv3 on most datasets such as FUNSD, CORD, RVL-CDIP, and DocVQA.

3.3.4 GPT-4

The last model covered in this section is GPT-4 [52]. Unlike the aforementioned architectures, GPT-4 uses the decoder part of the Transformer architecture and is targeted toward generative tasks.

While its predecessor - GPT-3.5 is an unimodal, text-only model, GPT-4 extends its pre-training process to include visual information. The pre-training comprises a variety of publicly available and proprietary data. Unfortunately, as of right now, the authors have not shared any concrete information about the model’s architecture such as its pre-training tasks or the number of parameters, therefore, these are currently left to speculation.

The model achieves state-of-the-art results on various benchmarks such as HumanEval and is even capable of reaching human-level performance on several academic exams, e.g. LSAT, SAT math, AP tests, etc. Note that the model is likely to improve even further. For instance, an implementation² that utilizes the Reflexion mechanism [63] manages to improve GPT-4’s accuracy on HumanEval from 67% accuracy to 88%. For inference, GPT-4 can be used in a text-only setting, or alternatively with both image and text input for tasks such as visual question answering.

²<https://github.com/GammaTauAI/reflexion-human-eval>

4 Datasets

Currently, there exist many multi-modal datasets that are publicly available and also used as benchmarks for state-of-the-art models. Since the purpose of this work is to utilize documents that mainly consist of text and image modality, our focus in this section is to provide an overview of the most common datasets in this area.

4.1 WIT

WIT¹ - Wikipedia Image-Text Dataset is currently the largest image-text dataset that is available for public use [68]. The dataset is developed by Google and contains around 37.5 million image-text examples in over 108 languages. As the name suggests, the dataset has been created by crawling Wikipedia’s content pages, corresponding to approximately 124 million pages in 279 languages. The content pages are then filtered and preprocessed to extract images with their associated text and useful metadata.

To further improve the quality of the dataset, a small subset of the dataset is validated by human annotators (via crowdsourcing), who determine whether a given text is relevant to the specified image - i.e. they validate the quality of the image-text pair.

Overall, the authors extract three different types of text - reference description (text shown below the image on the content page), attribution description (visible on the page of the image), and alt-text description (mainly used for accessibility purposes). Due to its size, the dataset can be used for both pre-training and for downstream tasks such as image-text retrieval.

4.2 RVL-CDIP

RVL-CDIP² [20] (Ryerson Vision Lab Complex Document Information Processing) is a large image dataset consisting of 400,000 grayscale documents. While this dataset is not technically multi-modal - it contains only images - it is often used as such, since the textual modality can be extracted using OCR software.

¹<https://ai.googleblog.com/2021/09/announcing-wit-wikipedia-based-image.html>

²<https://adamharley.com/rvl-cdip/>

The dataset is primarily used for classification. Each document is assigned one of 16 classes, such as form, resume, specification, scientific report, etc. that are evenly distributed. Although the dataset is relatively large, many of the samples suffer from noise and low resolution, which makes the training more difficult. An example from this dataset can be seen in Fig. 4.1.

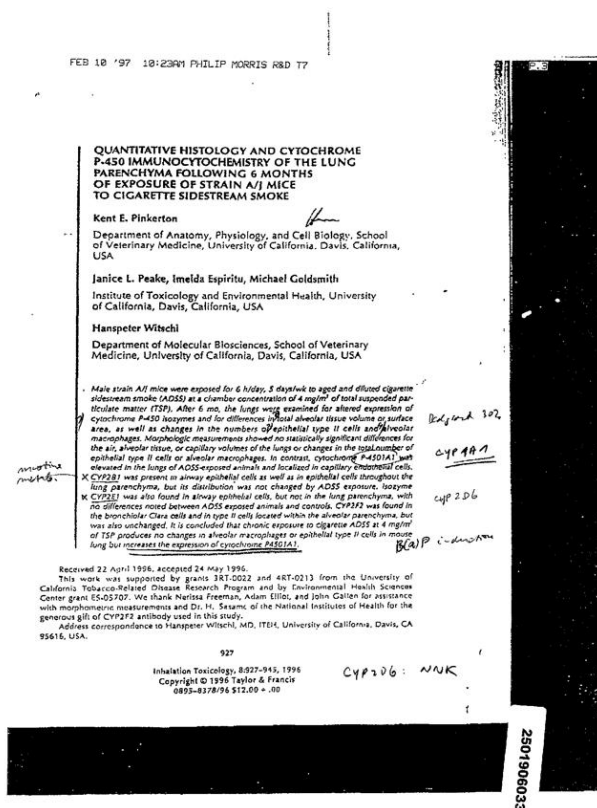


Figure 4.1: Example of a document from the RVL-CDIP dataset (scientific publication label) [20].

4.3 FUNSD

FUNSD³ is another popular document/form understanding dataset that can be used for tasks such as semantic entity labeling, entity linking as well as optical character recognition or spatial layout analysis [32]. The number

³<https://guillaumejaume.github.io/FUNSD/>

of examples is relatively small. It contains 199 annotated scanned forms divided into 149 train and 50 test examples.

The documents are a subset of the RVL-CDIP dataset with the *form* label but have been manually reviewed and annotated to ensure that only the most usable documents are provided. Each example contains a grayscale image and a corresponding list of tokens with bounding boxes and labels saved in JSON format. In total, the dataset captures 9707 semantic entities across 31,485 words. An example of the document with annotations (highlighted in color) is shown in Figure 4.2.

Figure 4.2: Example of a document from the FUNSD dataset [32].

4.4 CORD

CORD⁴ (A Consolidated Receipt Dataset for post-OCR parsing) is a large dataset of 1,000 samples containing receipt data collected from Indonesian restaurants and shops [56].

The ground truth labels are divided into two levels - 8 superclasses (category) and 54 subclasses (tag fields of the superclass). Superclass contains labels such as `menu`, `total`, `void total`, etc., while subclasses are individual fields - e.g. `menu.num`, `menu.price`, `total.price`, etc.

Each example contains a receipt image and ground truth data serialized in JSON format. Similar to FUNSD, CORD is applicable to document

⁴<https://github.com/clovaai/cord>

understanding tasks such as semantic entity labeling. An example of the dataset with corresponding JSON metadata is shown in Fig. 4.3.

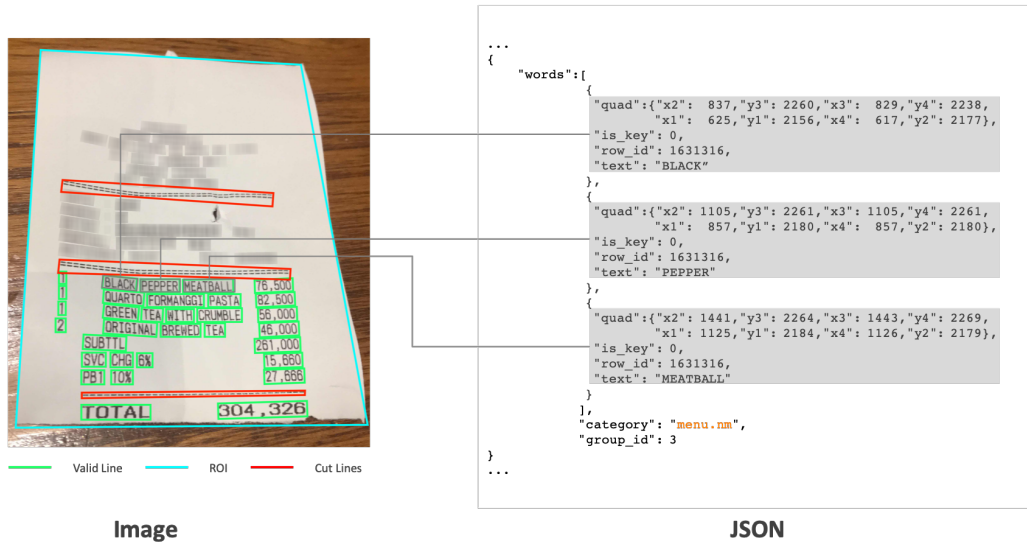


Figure 4.3: Example of a document from the CORD dataset - image and JSON data [56].

4.5 VQA

Visual Question Answering (VQA) is both the name of a dataset⁵ and also the name of a multi-modal task. VQA comprises a set of open-ended questions about images that require the model to have a deep understanding of vision, language, and also common sense [2]. The objective is to provide an understandable and coherent reply in natural language based on the image input and open-ended natural language questions. An example of the input-output pair can be seen in Fig. 4.4.

The VQA dataset uses around 204k images from the MS COCO dataset, which is one of the most widely used datasets for image segmentation and object detection. Each image is associated with several questions (at least 3 per image, i.e. around 614k in total) that have several possible responses (10 per question).

⁵<https://visualqa.org/>



Figure 4.4: Four different VQA examples [2].

4.6 PubLayNet

PubLayNet [82] is a large dataset of documents containing annotations for document layout analysis and understanding. In its current version⁶, it contains around 358k images of research papers/articles that were obtained from PubMed Central. The annotations are of very high quality because they have been automatically matched from the XML variant of the documents.

The dataset consists of three splits - train, validation, and test, which contain 330k, 11.2k, and 11.4k images respectively. As with other image-only datasets, text representations can be obtained by OCR. The example from the dataset can be seen in Section 3.2 in Fig. 3.3.

⁶<https://developer.ibm.com/exchanges/data/all/publaynet/>

5 Optical Character Recognition

So far, we have only discussed how to process multi-modal datasets for specific tasks. However, in many cases, only visual information is available, whereas the textual modality needs to be extracted. For printed text, which is the main type of text we consider for this work, the common approach is to use optical character recognition tools that can fully automate this process. Currently, there are numerous OCR solutions available for a wide range of applications, catering to both non-technical and technical users. These solutions vary not only in the area they specialize in but also in their scope, ranging from locally usable model implementations to fully managed cloud-based OCR services such as Amazon Textract¹ and Google Vision AI².

For the purposes of this thesis, it makes the most sense to use an OCR tool that offers straightforward communication with common machine learning frameworks such as TensorFlow or PyTorch since such frameworks will be used to construct the multi-modal data processing model. For this reason, we do not explore any cloud-based solutions as they usually require a complex setup and do not offer the degree of customization of most traditional open-source OCR frameworks. In some form, the OCR tool should also offer fine-tuning or pluggable custom models, which will let us train on custom samples to improve performance.

In the following Sections 5.1-5.7 we briefly overview several viable candidates for this work and compare them in Section 5.8.

5.1 EasyOCR

EasyOCR [31] is a popular end-to-end OCR solution that is integrated with the Python ecosystem and developed by JaidedAI. As the name suggests, the framework is straightforward to use, and extracting text from an image can be done in a few lines of code. Feature-wise, EasyOCR officially offers support for about 80 languages and is compatible with popular writing scripts such as Latin, Arabic, Chinese, Cyrillic, etc.

Similar to most OCR frameworks, text extraction is done in two phases

¹<https://aws.amazon.com/textract/>

²<https://cloud.google.com/vision>

- text detection and text recognition. By default, for text detection, the framework uses the CRAFT model [5], which is a fully-convolutional neural network based on VGG16 with additional enhancements such as BatchNorm [30] and skip connections. The subsequent text recognition is done by the CRNN model [62], which extracts a feature sequence that is then decoded by a deep BiLSTM into the actual text.

In addition, both text detector and text recognizer components are interchangeable with other models, which makes the framework very flexible as new state-of-the-art models can be easily plugged in. The entire architecture of the framework can be seen in Fig. 5.1. All models in the framework use PyTorch as its backend.

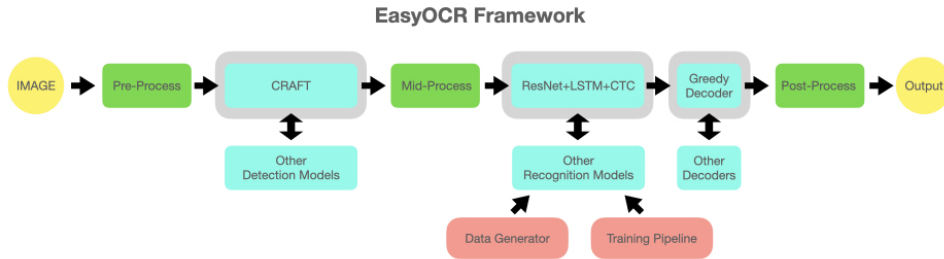


Figure 5.1: Architecture of EasyOCR [31].

5.2 Tesseract

Tesseract [66] is another popular, if not the most popular, OCR engine. It was originally developed by Hewlett-Packard as proprietary software but was since been open-sourced under the Apache license with Google as one of its main developers. Tesseract itself is written in C++ and distributed as a command-line application. However, it can also be easily exploited with many popular languages such as Java (via Tess4J) or Python (via PyTesseract).

To extract text, Tesseract uses two components - a text line recognizer, which is an LSTM adapted from OCRopus [7] and implemented in C++, and a character classifier. The output lines from the text line recognizer are fed to the classifier, which is a BiLSTM. The outputs of the character classifier are then fed to a trained language model, which outputs the final word. The visualization of this can be seen in Fig. 5.2.

In terms of recognized languages, Tesseract officially supports over 120 languages with many historical variants such as 12th-century English or

ancient Greek. Such pre-trained models can also be fine-tuned on custom data to further improve accuracy or to adapt the model to new scripts.

How Tesseract uses LSTMs...

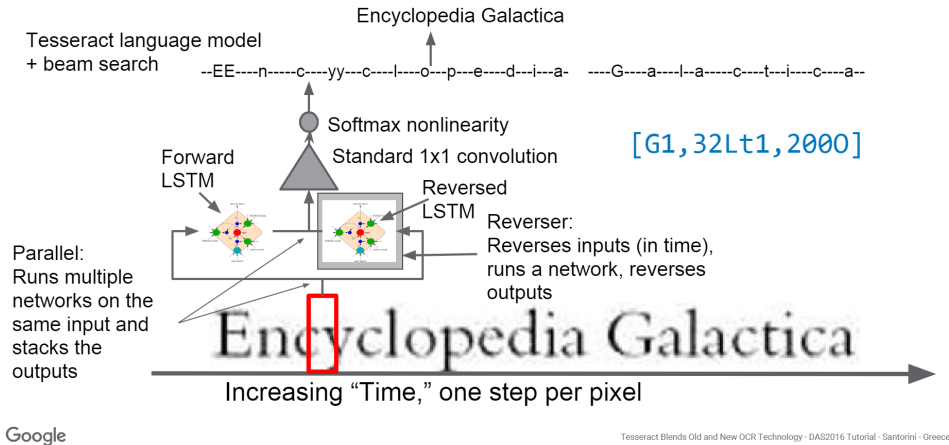


Figure 5.2: BiLSTMs in Tesseract [66].

5.3 PaddleOCR

PaddleOCR [53] is (after Tesseract) the second most popular OCR framework on GitHub. It is developed by Baidu and runs on the PaddlePaddle (PARallel Distributed Deep LEarning) platform. One of its biggest advantages is that it can run on a variety of platforms, ranging from mobile devices to computing clusters.

Depending on the platform, users can choose from various sizes of the PP-OCRv3 model (from around 9.4M parameters to 143.4M parameters). As in EasyOCR, the text is processed in two phases - text detection, which uses differentiable binarization [39] and CRNN for text recognition. The architecture of the OCR model is also customizable with other state-of-the-art algorithms/models. The list of most features of the framework can be seen in Fig. 5.3.

Application	Financial scene <ul style="list-style-type: none"> Forms Bills 	Industrial scene <ul style="list-style-type: none"> Watt hour meter License plate 	Educational scene <ul style="list-style-type: none"> Handwriting Formula 	Medical scene <ul style="list-style-type: none"> laboratory test report 	Community <ul style="list-style-type: none"> Official discussion group Contributor Honor Wall Regular season challenge 	
Deployment	Training mode <ul style="list-style-type: none"> Normal Distributed Mixed Precision 	Training Env. <ul style="list-style-type: none"> Linux GPU/CPU Linux DCU Windows GPU/CPU macOS 	Compression <ul style="list-style-type: none"> Prune Quantization Distillation 	Inference and Deployment <ul style="list-style-type: none"> Python/C++ Inference Python/C++ Serving OpenCL ARM GPU Paddle2ONNX PaddleCloud 	<ul style="list-style-type: none"> ARM CPU Jetson Paddle.js 	
Industrial models and solutions	PP-OCR: Ultra-lightweight OCR System 🔥 <ul style="list-style-type: none"> PP-OCRv3: detection + direction classifier + recognition =17.0M English & numbers model: applicable to scenarios which only contain English and numbers. Multilingual models: support 80 languages including Korean, Japanese, German, French, etc. 		PP-Structure: Structured Document Analysis System 🔥 <ul style="list-style-type: none"> Support layout analysis Support table recognition (support export to Excel) Support KIE (including semantic entity recognition & relation extraction) Support layout recovery Support PDF to Word 			E-book: Dive into OCR <ul style="list-style-type: none"> Comprehensive OCR technology Theory + Practice Notebook interactive learning Teaching video
Algorithms	Text Detection <ul style="list-style-type: none"> EAST DB SAST PSENet FCENet 	Text Recognition <ul style="list-style-type: none"> CRNN SRN NRTR SVTR ABINet 	End-to-end <ul style="list-style-type: none"> PGNet 	Layout Analysis <ul style="list-style-type: none"> Layoutparser PP-Picodet 	Table Recognition <ul style="list-style-type: none"> TableRec-RARE TableMaster SLANet 	Key Information Extraction <ul style="list-style-type: none"> SDMGR LayoutLM LayoutLMv2 LayoutXLM VI-LayoutXLM
					Data tools <ul style="list-style-type: none"> Semi-automatic data annotation tool: PPOCRLabel Data synthesis tool: Style-Text 	

Figure 5.3: Features of PaddleOCR [53].

5.4 docTR

docTR (Document Text Recognition) [48] is a relatively new OCR library that supports both PyTorch and TensorFlow. Similar to EasyOCR and Tesseract, it provides an end-to-end solution for image-to-text conversion. Its inner workings are very similar to EasyOCR, where processing comprises text detection and text recognition, both of which are fully configurable with state-of-the-art networks. Users can employ their own models or fine-tune pre-trained ones on custom datasets. Additionally, all features of the platform are available to PyTorch and Tensorflow respectively.

5.5 Ocropus

Another note-worthy OCR system is OCRopus. Unlike Tesseract, OCRopus tries to create a collection of modular document-analysis methods that can be used together to create a custom processing pipeline. Like all of the mentioned tools, Ocropus offers Python support with its wrapper library called Ocropy, which heavily simplifies the work with the tool. Unfortunately, the project does not seem to be actively developed, as the last version was released in December 2017.

5.6 Kraken

Kraken [34] is a fork of Ocropus that focuses primarily on historical texts such as printed Arabic, Persian, or medieval Latin and French. It provides multi-script text recognition and supports recognition in right-to-left, bi-directional, or top-to-bottom modes. It can be used both in a command line as well as a Python module. By default, Kraken uses a similar model to Ocropus' LSTM (which is also adapted in Tesseract).

5.7 Calamari

As a last entry, Calamari [74] is a fork of Ocropus and Kraken that is also used mainly with historical documents. It is implemented in Python and uses TensorFlow as its computational backend. Its default model is an FCN-LSTM hybrid where FCN (fully convolutional network) is used for text detection and LSTM for subsequent text recognition. Users can use either pre-trained models, training from scratch, or fine-tuning existing models. In addition, for inference, the models can be combined together where the output is determined based on the model votes.

5.8 Comparison

This section serves as a comparison of the seven aforementioned OCR tools. Table 5.1 shows important features of all tools/frameworks such as the number of recognized languages, the ability to use custom models, or supported output formats. Overall, the feature sets of the tools are very similar, and their usage will mainly depend on the type of data that the user needs to process. In practice, the listed tools could be divided into two groups - general purpose OCR - i.e. text extraction from conventional documents and photos, and OCR for historical documents, which targets unconventional fonts and noisy documents.

Each of these frameworks offers some form of Python support - either via an API or because it is written in Python itself, making it easy to integrate with any major ML platform. EasyOCR, PaddleOCR, and docTR also support different models for both detection and recognition phases.

Another important metric to consider is the size of the user base around the framework. While a larger number of users does not necessarily imply a better solution, it usually means that there are more educational materials for newcomers and more third-party packages that can extend the function-

ality of the framework or add missing features such as support for new font or programming language.

While this is difficult to quantify, the size of the user base can be roughly estimated by the project’s popularity on software distribution platforms such as GitHub - which is nowadays one of the most popular ways for developers to share software. Although GitHub does not publicly disclose the number of unique or recent downloads, we can use other metrics such as the number of stars or the number of forks, which are shown in Table 5.2. Using both the number of forks and the number of stars, the most popular project is Tesseract, followed by PaddleOCR, and EasyOCR. Arguably, this result is to be expected as all three tools are used for general-purpose OCR, while tools like Kraken or Calamari target a niche user base interested mainly in historical data.

Ultimately, we decided to use Tesseract for the purposes of this thesis. There are two reasons for this choice. Firstly, Tesseract offers great integration with many libraries and models, such as LayoutLMv3, which is not as common with other OCR frameworks. Secondly, due to its popularity, there are many pre-trained models for our type of data, which is only common with historically focused OCR frameworks. However, most of these frameworks would require much more effort for integration.

Name	# of supported languages	Custom models	API	Architecture
EasyOCR	80	Yes	Python	CRNN
Tesseract	120	No (fine-tuning only)	CLI, Python, C/C++, JavaScript, Java	LSTM
PaddleOCR	80	Yes	Python JavaScript	CRNN
docTR	N/A	Yes	Python	CRNN
Ocropus	N/A	No (fine-tuning only)	CLI, Python	LSTM
Kraken	N/A	No (fine-tuning only)	CLI, Python	LSTM
Calamari	N/A	No (fine-tuning only)	CLI, Python	CRNN

Table 5.1: Comparison of features of each OCR tool/platform.

Name	# of stars	# of forks
EasyOCR	16.2k	2.3k
Tesseract	47.2k	8.2k
PaddleOCR	26.2k	5.4k
docTR	1.4k	173
OCropus	3.2k	585
Kraken	438	93
Calamari	954	159

Table 5.2: GitHub statistics for each project.

6 Heimatkunde Dataset

As a dataset processed in the thesis, we use images from two historical books describing political districts in the Czech Republic - *Heimatkunde des Ascher Bezirkes* (Local History of the Aš District) by J. Tittmann and *Heimatkunde des politischen Bezirkes Plan* (Local History of the Planá District) by Georg Weidl. Due to the name of the books, we name the resulting dataset the *Heimatkunde* dataset. The documents contain information about the geography, agriculture, population, administration, education, and local history of the districts at the end of the 19th century. The text in both books is printed in Fraktur font and written in German.

Most of the book pages have a conventional two-page layout in a landscape format and are grayscale scans at a very high resolution (300 DPI and most of the images are around 3400×2500 pixels in width and height). An unprocessed example from the dataset can be seen in Figure 6.1.

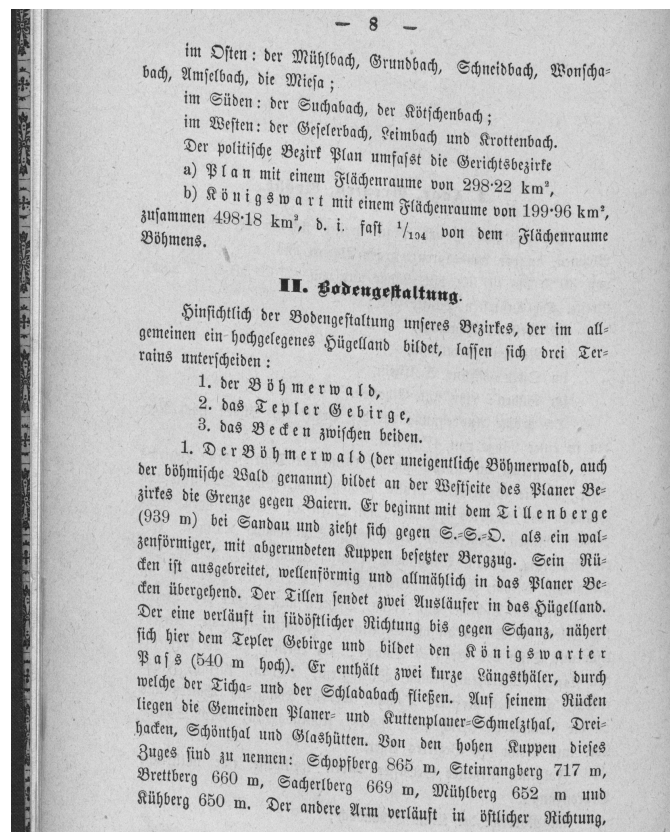


Figure 6.1: Example of an unprocessed page from the dataset.

6.1 Dataset Classes

In total, both books contain around 468 images or approximately 930 pages. For our dataset, we use only a subset - 329 images, which we have manually annotated for the document layout analysis task performed later in Chapter 7. In total, there are 7 types of objects that we identify in the dataset.

Although some of the original documents contain images, we decided not to include them as there are only 10 images in both books and such a sample size is not enough to perform training or validation. Consequently, all of the 7 classes contain some form of text, which should however be advantageous for multi-modal processing since the model can always utilize both sources of information. The classes of the document entities are as follows:

- **Paragraph** - larger block of text, often with an indented first line.
- **Heading** - bold text in a different font style that is one or few lines long.
- **Footnote** - contains miscellaneous information, located at the bottom of the page, separated from paragraphs by a line.
- **Page number** - always at the top of the page.
- **Table** - a collection of rows and columns, often with different formatting. May or may not have borders.
- **List / Listing** - a list of items, e.g. animal species, list of inhabitants, etc.
- **Centered text** - typically a small portion of text containing quotations, smaller font size than a paragraph.

Each of the selected categories should be either semantically or visually distinct. Additionally, some classes such as page numbers or footnotes only appear in certain parts of the layout, which is another source of information that could, in theory, be exploited by the model.

Arguably the two most difficult elements to recognize/classify should be tables and centered text. While centered text appears consistently throughout the data, there are not many samples (see Table 6.1 in the following section), and tables, on the other hand, can have several formats. One solution would be to create a separate class for each type of table but this is likely not feasible here as the number of tables in the text is low as well.

6.2 Annotation Process

To create the annotations we use CVAT¹ [10] (Computer Vision Annotation Tool), which is a widely used computer vision annotation software. Arguably one of the main benefits of this application is that it is open-sourced and can be deployed locally in Docker.

All the images are annotated for the instance segmentation task, where we denote the area of each object by a polygon. An example of the annotation from the editor can be seen in Figure 6.2. Additionally, we also save bounding boxes of each object (simply by using the minimum and maximum from each x,y coordinate), as they are used to extract additional data such as text that is needed for the experiments.

The annotations are converted to the COCO format, which makes the most sense for our use case as this format is directly supported by many image segmentation frameworks. Additionally, it is very straightforward to work with and can be easily transformed into other formats such as YOLO.

6.3 Resulting Dataset

As a result of the annotation process, we obtain a dataset that can be used for layout analysis in historical documents. In total, there are approximately 4.6k annotations across 329 images. The created dataset has a relatively large imbalance between the classes, which is to be expected since some elements such as paragraphs occur much more frequently than elements such as tables or footnotes.

The counts of the individual classes can be seen in Table 6.1. The two most common types of entities are paragraphs and listings. On the other hand, centered text and tables appear infrequently and should be harder for the model to detect.

Finally, we split the dataset for training and evaluation. Approximately 70% of the dataset is used for training while the remaining 30% is kept as evaluation data. The counts for each split can be seen in Table 6.2.

¹<https://github.com/opencv/cvat>

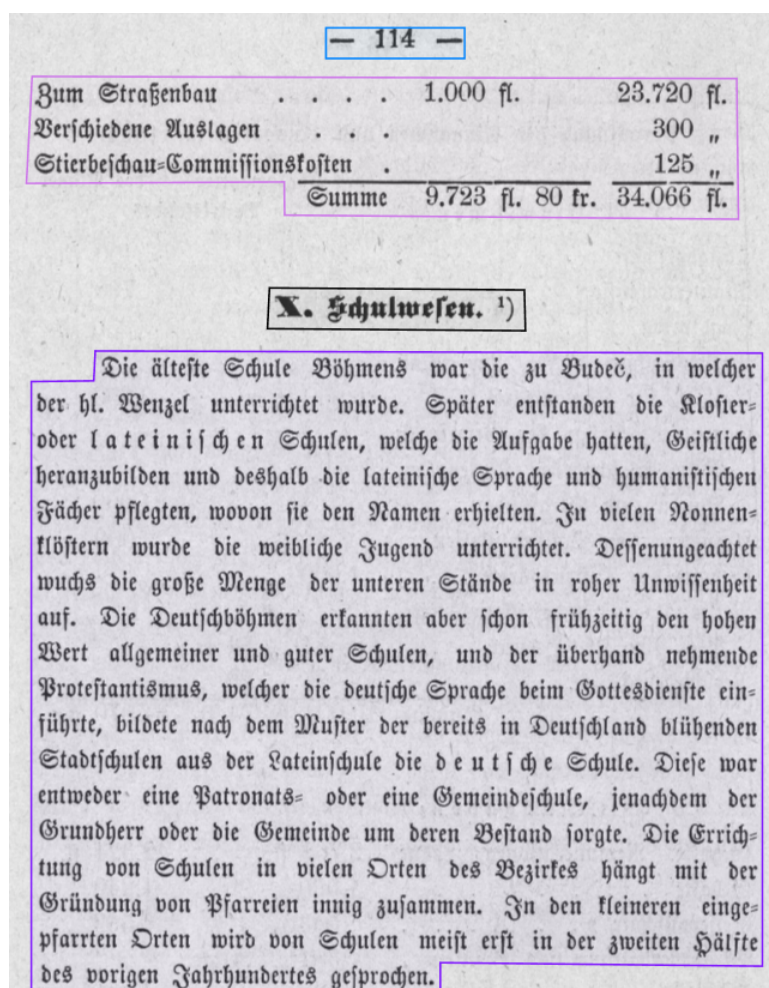


Figure 6.2: An example of the annotation in the CVAT application. There are four classes in the image - page number (blue), table (pink), heading (black), and paragraph (purple).

Class name	Count	[%]
Paragraph	2079	44.806
Listing	1306	28.147
Page Number	640	13.793
Heading	378	8.147
Footnote	107	2.306
Table	91	1.961
Centered text	39	0.841
Total	4640	100

Table 6.1: Number of occurrences for each class in the dataset, sorted by the most frequent class first.

Split	<i>Train</i>		<i>Test</i>	
Class name	Count	[%]	Count	[%]
Paragraph	1483	45.352	596	43.504
Listing	921	28.165	385	28.102
Page Number	447	13.670	193	14.088
Heading	264	8.073	114	8.321
Footnote	74	2.263	33	2.409
Table	59	1.804	32	2.336
CenteredText	22	0.673	17	1.241
Total	3270	100	1370	100

Table 6.2: Number of occurrences for each class in the train-test split, sorted by the most frequent class first.

6.3.1 OCR Subset

In addition to the document layout analysis variant of our dataset, we also utilize a subset of its images to create examples used for training and evaluation of an OCR model, which is in detail explained later in Section 6.5. Each example comprises an image that contains a line or uniform block of text as well as a corresponding ground truth label. Such examples are shown in Figures 6.3 and 6.4.

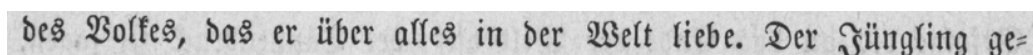


Figure 6.3: Example of the test sample with reference text: "des Volkes, das er über alles in der Welt liebe. Der Jüngling ge-".

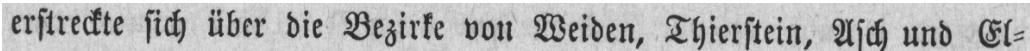


Figure 6.4: Example of the test sample with reference text: "*erstreckte sich über die Bezirke von Weiden, Thierstein, Asch und El-*".

For training, we use two variants of the dataset. The first variant contains only the annotated examples from our dataset, which is around 14 pages, or 782 lines. The other variant is larger and includes our annotations as well as annotations from the Historical German OCR Corpus [46]. This OCR dataset contains very similar data and has 1386 lines. In total, the second variant results in 2168 lines of text.

As for the evaluation, we annotate around 12 pages, resulting in 439 lines or 4430 words. Such a sample size should provide meaningful enough results to estimate the performance of an OCR model.

6.4 Relevant Metrics

This section describes relevant metrics that we use to evaluate the performance of the models on the Heimatkunde dataset.

6.4.1 Classification

In classification, arguably the most popular metric is accuracy. It measures the number of correct predictions compared to the total number of predictions [25]. With respect to a given class, a prediction can be either true positive (TP), true negative (TN), false positive (FP), or false negative (FN). TP and TN predictions denote instances that were correctly labeled, while FP and FN represent misclassifications. Putting all the counts together, accuracy can be defined according to Eq. 6.1.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (6.1)$$

Precision (Eq. 6.2) is defined as the number of correctly predicted positive instances out of all predicted positive instances. This metric is useful for detecting high false positive predictions.

$$Precision = \frac{TP}{TP + FP} \quad (6.2)$$

Recall (Eq. 6.3) is a metric that describes the fraction of positive samples that are correctly classified [25]. It defines how many of the relevant instances the classifier marks as positive.

$$Recall = \frac{TP}{TP + FN} \quad (6.3)$$

Both precision and recall are important and can be captured in a single metric - the F_β score, where β defines the weights of precision and recall as seen in Eq. 6.4. In most cases, both precision and recall are given equal weight, which can be achieved by setting $\beta = 1$. Such a score is commonly referred to as the F1 score.

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \quad (6.4)$$

6.4.2 OCR

To evaluate the quality of a specific OCR model, two metrics are typically used. Character error rate (CER) specifies the percentage of wrong characters from the model output - i.e. it is essentially the inverse of accuracy. As seen in Eq. 6.5, CER can be calculated by counting the number of character insertions (i), substitutions (s), and deletions (d) to transform the OCR output into ground truth text [50], which has a total of n characters.

$$CER = \frac{i + s + d}{n} \quad (6.5)$$

Word error rate (WER) is similar to CER, except that it works at the word level. Therefore, it is calculated by counting the number of words in the reference text (N), word substitutions (S), word insertions (I), and word deletions (D). This is represented by the formula in Eq. 6.6.

$$CER = \frac{I + S + D}{N} \quad (6.6)$$

6.4.3 Document Layout Analysis

In the context of our dataset, document layout analysis is essentially a special case of instance segmentation, hence the utilized metrics are identical. The terminology is similar to classification, except that the concept of TN prediction is not applicable, since we can essentially find an infinite number of them in a given image [54].

Intersection over union (IoU) is typically used to compare the predicted instance to ground truth. The metric is computed using the formula shown in Equation 6.7, where the *Instance* term can be represented by either a corresponding bounding box or segmentation mask. IoU itself is computed as the ratio of the area of the intersection between the predicted instance

($Instance_{pred}$) and the ground truth instance ($Instance_{gt}$) to the area of their union.

$$IoU = \frac{Area(Instance_{gt} \cap Instance_{pred})}{Area(Instance_{gt} \cup Instance_{pred})} \quad (6.7)$$

To determine whether a prediction is positive, we compare the computed IoU to a certain threshold, conventionally ranging from 0.5 to 0.95. Based on these values, we obtain TP, FP, and FN values to compute precision and recall. Note that if there are two TP instances predicting the same ground truth instance, the one with a lower IoU is labeled as FP.

In addition to the predicted label, the model also outputs a confidence score, which can be used to filter out instances for which the model is less certain (below a specific threshold). This makes it possible to control the precision and recall of the model and consequently create a precision-recall (PR) curve, which indicates a trade-off between the metrics. Average precision (AP) can be calculated as the area under the curve (AUC) of the PR plot. However, computing AUC directly is often impractical as the plot is a zigzag-like curve and the AUC will be skewed [54], therefore it is common to interpolate the values.

Another important term is mean AP, commonly referred to as mAP, which measures the AP across all classes in the dataset. The mAP for N classes is computed simply as the mean of the class-specific APs as shown in Eq. 6.8.

$$mAP = \frac{1}{N} \cdot \sum_{i=1}^N AP_{class=i} \quad (6.8)$$

In the case of our dataset, we use the COCO evaluation metrics², which are commonly used to evaluate the image segmentation of state-of-the-art models such as the one proposed in the LayoutLMv3 paper. In the context of the COCO evaluation, average precision is computed across all categories - thus it behaves like mAP.

The COCO evaluation provides three variants of the AP metric that are relevant to our dataset. The most important metric, AP or AP@[0.50:0.95] is calculated as an average over 10 IoU thresholds ranging from 0.50 to 0.95 with 0.05 increments. The formula for calculating the COCO variant of AP is shown in Eq. 6.9. The second metric is AP50, which defines (mean) average precision at IoU = 0.5 - this is identical to the Pascal VOC metric [15], and analogously the last metric - AP75 is (m)AP at IoU = 0.75.

²<https://cocodataset.org/#detection-eval>

$$AP@[0.50:0.95] = \frac{AP50 + AP55 + \dots + AP90 + AP95}{10} \quad (6.9)$$

6.5 Extraction of the Text

Since the Heimatkunde dataset is captured purely in image form, it is necessary to have a mechanism capable of extracting the text, as we utilize this modality later in our document layout analysis experiment. Fortunately, all of the text is printed (i.e. it contains no handwritten portions) and uses the same font style. Therefore, we can easily utilize many of the OCR frameworks we covered in Chapter 5.

As we mentioned in Section 5.8, we use Tesseract due to its large user base and integration with many ML libraries. Since our data contains relatively non-standard text, it is necessary to train a new Tesseract model to recognize the text in order to reduce the error rate.

We can either train the model from scratch, which requires a large amount of data for satisfactory performance, or we can use an existing model and fine-tune it. Since the dataset is relatively small, we choose the fine-tuning approach because it requires fewer training iterations and the resulting model is likely to achieve better performance due to a larger corpus.

There are two viable candidates that we consider as a starting point. The first one is the GT4HistOCR dataset [67], which contains a large number of training examples - around 313k lines of text in German Fraktur and Early Modern Latin. In addition, there exist pre-trained models on this dataset which are publicly available as a part of the OCR-D project [51] in the form of Tesseract weights³. The other option is to use Tesseract’s Fraktur model from the official GitHub repository⁴.

6.5.1 Pre-trained Models

Both Tesseract and GT4HistOCR pre-trained models are tested on the evaluation subset of our dataset, which we describe in Section 6.3.1. Based on the CER and WER scores of the models, the better-performing one is selected as the pre-training base. In our benchmark, Tesseract’s Fraktur model performs significantly better than the model trained on GT4HistOCR. The GT4HistOCR model achieves a CER of 3.494% and a WER of 17.223%,

³<https://ub-backup.bib.uni-mannheim.de/~stweil/ocrd-train/data/>

⁴<https://github.com/tesseract-ocr/tessdata.git>

while Tesseract’s Fraktur (FRK) model achieves a CER of 1.667% and a WER of 8.481%. Therefore, we use the FRK model for subsequent fine-tuning.

6.5.2 Fine-tuning

To fine-tune the FRK model, we use the training subset of our dataset, which is described in Section 6.3.1. Note that the total number of samples used for training is slightly smaller because 10% of the samples are used for validation. Therefore, the actual sizes are 1950 and 703 for annotations with and without the Historical German OCR Corpus, respectively. The training itself is done by Tesseract’s `Tesstrain` script⁵.

We train the model on both variants of the dataset for 50k iterations and checkpoint every 5k iterations. At each checkpoint, we take the model with the best training CER and WER since the previous checkpoint and test it on the evaluation dataset.

6.5.3 Results

We record both CER and WER values on the training dataset, which is shown in Figures 6.5 and 6.6. In both plots, it is visible that CER and WER gradually decrease over time. However, the same trend does not appear when the networks are tested on unseen data - see Fig. 6.7 and Fig. 6.8, indicating that the model becomes overfitted after approximately 10k iterations.

The best CER and WER scores on the evaluation dataset are achieved by the model trained on the larger training set (with the additional Historical OCR Corpus data) after approximately 10k training iterations. It achieves a character error rate of **1.229%** and a word error rate of **7.178%**. On the other hand, also after 10k iterations, the configuration trained only on our annotations is relatively close as well with 1.237% CER and 7.223% WER. Both models achieve performance comparable to other state-of-the-art models trained on similar datasets. The results of both pre-trained and fine-tuned models can be seen in Table 6.3.

⁵<https://github.com/tesseract-ocr/tesstrain>

Model	CER [%]	WER [%]	Iteration
Tesseract FRK	1.667	8.481	-
GT4HistOCR model ⁶	3.494	17.223	-
Our dataset	1.237	7.223	10k
Our dataset + Historical OCR Corpus	1.229	7.178	10k

Table 6.3: CER and WER of pre-trained models (from Section 6.5.1) and fine-tuned models (from Section 6.5.2). The best metrics are denoted in bold.

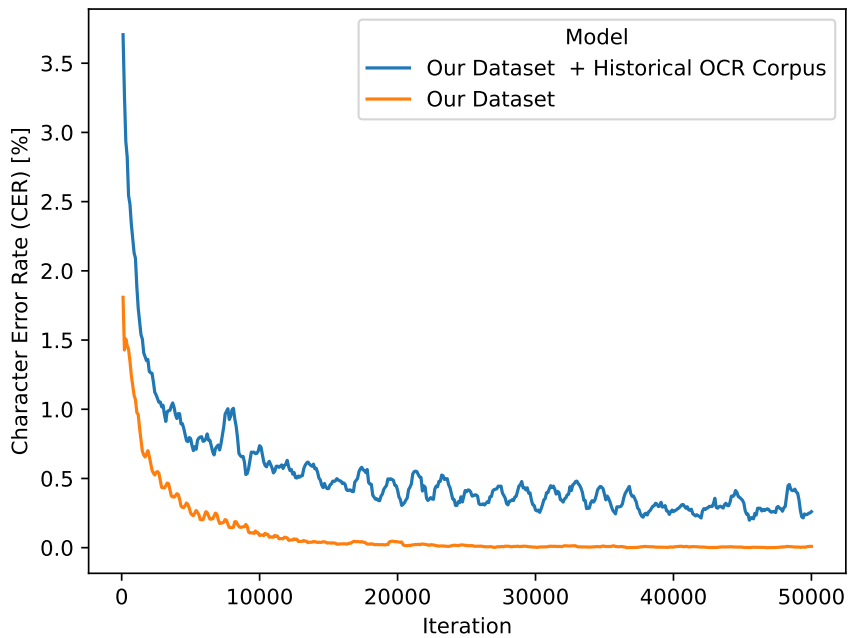


Figure 6.5: CER for both variants of the training dataset.

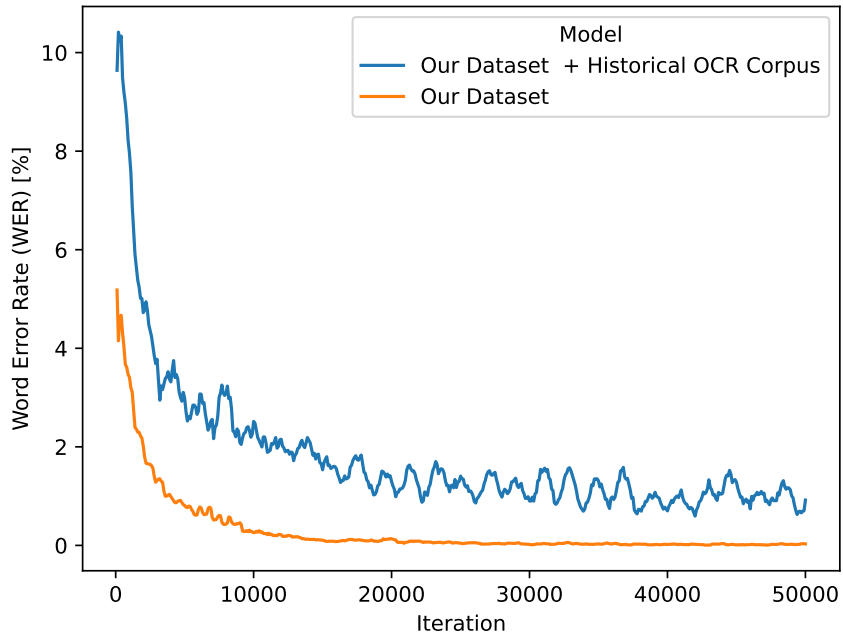


Figure 6.6: WER for both variants of the training dataset.

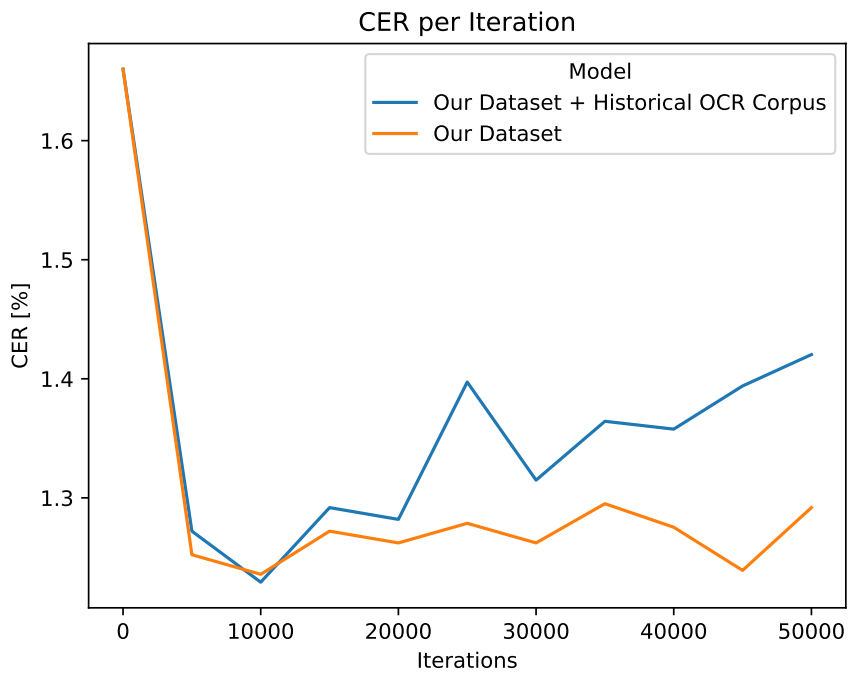


Figure 6.7: CER on the testing part of the corpus.

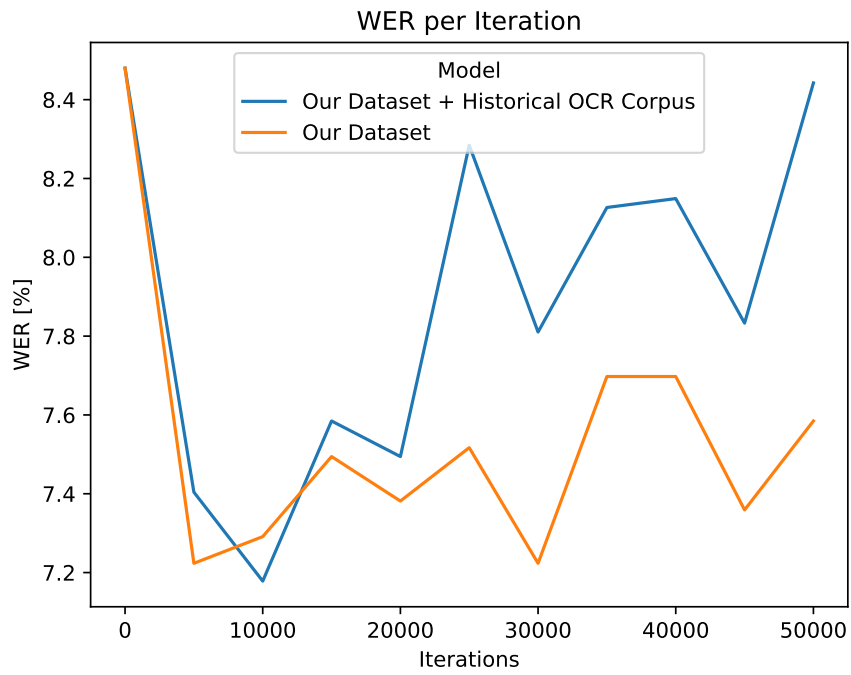


Figure 6.8: WER on the testing part of the corpus.

7 Multi-modal Document Layout Analysis

This chapter describes our solution of implementing a multi-modal document layout analysis system to process the data from the Heimatkunde dataset. In this context, the task is equivalent to instance segmentation because all recognized entities are themselves document components.

Such a system should utilize both textual and visual information to recognize each entity in the document, which can be beneficial for the further processing of multi-modal documents. For example, applying OCR at the component level can yield higher-quality text than recognizing characters in the entire document, where some tokens may be skipped or misrecognized. Another application could be in information retrieval systems, where it is possible to use the classes of the document entities to determine the relevance/importance of the search results.

The entire chapter consists of several parts. Firstly, we present the essential components of our solution, which is depicted in Figure 7.1. Its architecture comprises two different models - an instance segmentation model and a multi-modal classifier.

The instance segmentation model is described in Section 7.1. This model processes only the image features and extracts individual instances that are subsequently passed to the multi-modal classifier. The multi-modal classifier is described in Section 7.2. It operates on the output of the instance segmentation model as well as on the text extracted by the OCR model, which was trained in Section 6.5, and produces labels of the instances.

Secondly, in Section 7.3, we describe the architecture of our solution as a whole - i.e. the interaction between the classifier and the segmentation model, as well as implementation details. Thirdly, Section 7.4 is a small experiment where we test how the textual modality affects the overall classification.

Finally, we discuss the results as well as potential improvements in Chapter 8.

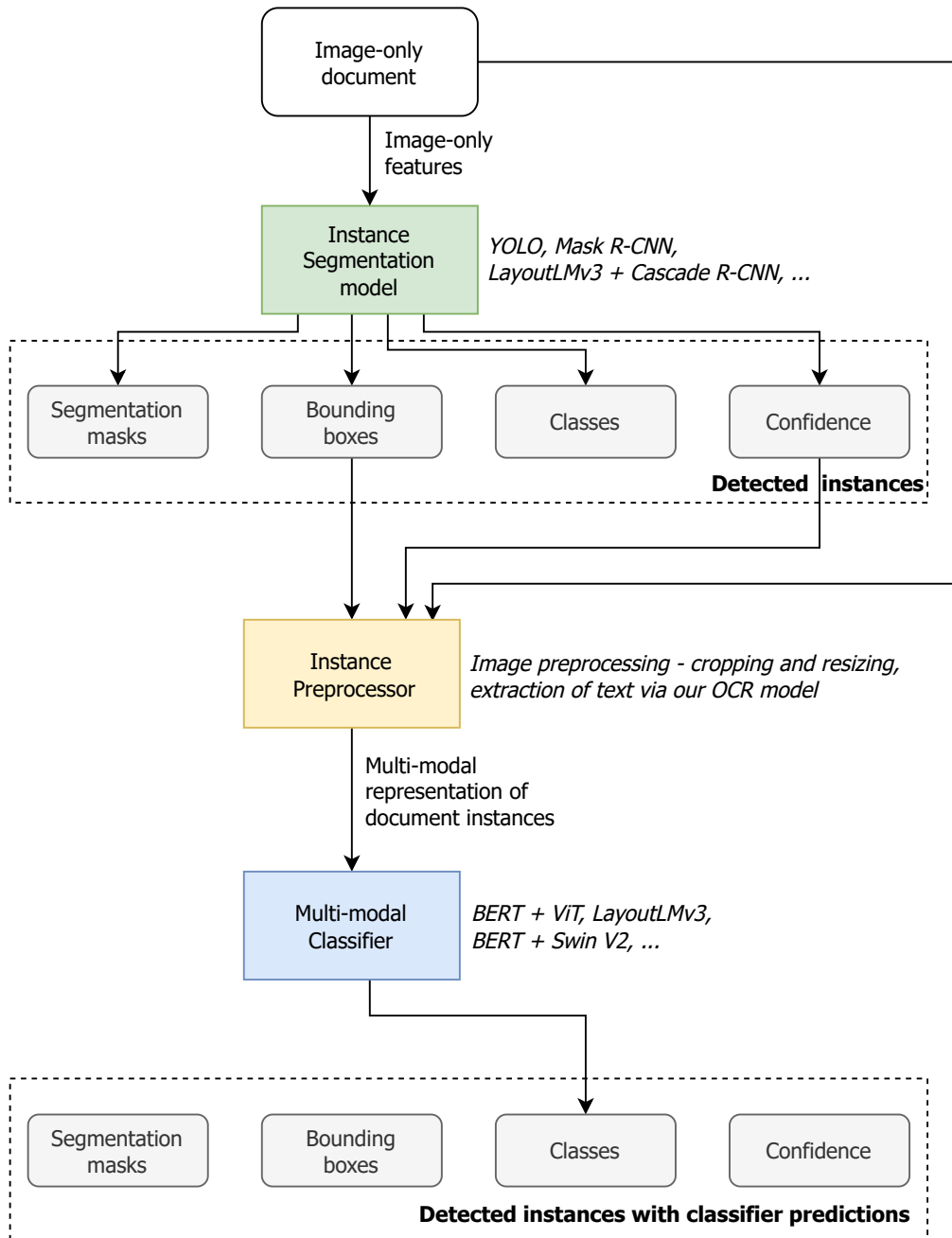


Figure 7.1: Architecture of our solution. It consists of a unimodal image-only instance segmentation model and a multi-modal classifier that predicts the class of each instance. Segmentation masks, bounding boxes, and confidences are retained from the segmentation. The extraction of the text is performed in the instance preprocessor, which maps the data to be digestible by the classifier - this also includes cropping the image accordingly to the detected instance.

7.1 Instance Segmentation

Firstly, we need to train an instance segmentation model that performs the detection of the individual components in the document. Currently, there are many viable models that can be used for this type of task. They can either be built in a machine learning framework like PyTorch (from scratch) or are preferably already implemented in a specialized image segmentation/object detection library.

Apart from the computational environment, segmentation models also differ in the number of trainable parameters and the type of data they work well with - e.g. some models are optimized for inference speed (which is important when dealing with real-time data such as video streams), while others aim for very high precision. As far as we are concerned the model should satisfy the following criteria:

- Runnable in the Python ecosystem and preferably built on top of PyTorch or TensorFlow frameworks
- Deployable on common consumer-grade hardware with a dedicated GPU
- Favor high precision over low response time as our task is not time critical

7.1.1 Models

In total, we make use of three different models - Mask R-CNN, YOLO, and LayoutLMv3 with Cascade R-CNN, which we discuss in the following sections. We utilize YOLO and Mask R-CNN since they are commonly used for instance segmentation, while the LayoutLMv3-based model achieves state-of-the-art results on various DLA datasets such as PubLayNet.

Mask R-CNN

Mask R-CNN is one of the most popular solutions for instance segmentation. We use Mask R-CNN implemented in Detectron2¹ [76], which is an image segmentation/object detection framework developed by the authors of PyTorch. Specifically, we use the `mask_rcnn_R_50_FPN_3x` configuration, which utilizes ResNet50 as its backbone.

¹<https://github.com/facebookresearch/detectron2>

YOLO

As a second model, the latest iteration of YOLO is used - YOLOv8², which is developed by Ultralytics [33]. The advantage of this model is its scalability, as it can even be deployed on mobile devices or e.g. Raspberry Pi because the smallest model `n` has only 3.4M parameters. On the other hand, larger variants `l` and `x` should match or exceed the performance of Mask R-CNN.

In the context of our implementation, the main drawback of the model is that it is not directly implemented in Detectron2, which requires additional effort to incorporate it into our multi-modal system. We choose to use the `l` variant of the model, as it has a similar number of parameters to Mask R-CNN.

LayoutLMv3 with Cascade R-CNN

The third model is based on LayoutLMv3, which is described in detail in Section 3.3.2. The implementation used here is adapted from the official repository³ and also utilizes Detectron2 for training and inference.

The main advantage of this model over YOLOv8 and Mask R-CNN should be its multi-modal pre-training. We expect this model to perform the best since all our classes contain textual features. The network is used as a backbone, while the segmentation is performed by Cascade R-CNN.

Note that while LayoutLMv3 itself is multi-modally pre-trained, the actual document layout analysis is unimodal because the authors only leverage the Vision Transformer part of the model.

Number of parameters

The number of parameters of each instance segmentation model can be seen in Table 7.1. Both YOLOv8 and Mask R-CNN use a similar number of parameters - around 40M, which makes them easily comparable. On the other hand, the LayoutLMv3-based model has around 140M parameters, accounting for more than three times the size of the CNN-only counterparts.

7.1.2 Preprocessing

Before we train the models on the dataset, it is necessary to preprocess the annotated examples and convert them to a digestible format. The CVAT

²<https://github.com/ultralytics/ultralytics>

³<https://github.com/microsoft/unilm/tree/master/layoutlmv3>

Model name	# of parameters
Mask R-CNN with ResNet50 backbone	41M
YOLOv8 (l-seg variant)	46M
Cascade R-CNN with LayoutLMv3 backbone	140M

Table 7.1: Number of parameters in each image segmentation model.

application (which we used to annotate the dataset) provides output in both COCO and YOLO formats, but the YOLO format is outdated (YOLO v1.1) and the annotated data is not split for training and validation.

Therefore, we use only the COCO output and manually map it to the YOLO format later in the preprocessing step. We divide the dataset into two splits - one for training and one for evaluation - which were already described in Section 6.3. To perform the train-test split, we use code from the `cocosplit` GitHub repository⁴.

The images are scanned at a very high resolution, which is not feasible for training or evaluation. Therefore, we downscale each image to a width of 1280 pixels or less. This keeps the text perfectly readable but significantly reduces the amount of memory needed to train the model. The downscaling also requires modifying the segmentations, bounding boxes, and their pre-computed areas in the COCO annotations, as none of them are normalized. Finally, we convert the preprocessed COCO train and test datasets to the YOLO format via scripts from the `JSON2YOLO` repository⁵.

7.1.3 Training

The training loop of each segmentation model is very similar, and the main differences are in the hyperparameters for each network. Both Mask R-CNN and LayoutLMv3 are trained via Detectron2, while YOLOv8 is trained using the Ultralytics library. Each model uses slightly different values for the learning rate (LR) and a different number of iterations to converge. All the important hyperparameters are included in Table 7.2. During training, the models are periodically evaluated on the test data, and their best weights are selected based on the COCO AP@[0.50:0.95] metric.

In the case of Mask R-CNN, we train with a batch size of 4 using the SGD optimizer and 1×10^{-4} learning rate. Additionally, we use weights from the pre-trained variant of the model on the COCO dataset, which we found

⁴<https://github.com/akarazniewicz/cocosplit>

⁵<https://github.com/ultralytics/JSON2YOLO>

to converge much faster than training from scratch. The network is trained for 20k iterations and evaluated on the testing data after every 1000 steps.

Regarding instance segmentation with LayoutLMv3, we try two different variants of the model. The first one uses only the default weights from pre-training, while the other one includes weights that were fine-tuned on the PublayNet dataset⁶. The network is trained with the AdamW optimizer [45], which uses a linear learning rate warmup with cosine decay. In total, the process runs for 20k iterations with evaluation after every 1000 steps and uses a batch size of 3.

The third utilized model, YOLOv8, offers similar hyperparameters to the models in Detectron2, except that the length of training is specified in epochs. The model is trained for 100 epochs using the SGD optimizer and is evaluated after each epoch. The learning rate of the optimizer is controlled by the One Cycle LR scheduler [65], which uses an initial learning rate of 1×10^{-2} and a final learning rate of 1×10^{-4} .

Based on the input resolution, two variants of the YOLOv8 are trained. The first variant uses an input size of 640p (i.e. 640 pixels in width), which is the default, and the second variant accepts 1280p input, which is used in both the LayoutLMv3-based model and Mask R-CNN since it is the maximum width of the images.

In both cases, the training uses weights from a model pre-trained on the COCO dataset. Due to the large memory consumption (over 16GB of VRAM for a 4-item batch), the 1280p variant uses a batch size of 2, while the 640p one uses a batch size of 4.

Model	Input	Initial Weights	LR	Optimizer	Scheduler	Batch
Mask R-CNN	1280	COCO	1×10^{-4}	SGD	None	4
LayoutLMv3	1280	Default	2×10^{-4}	AdamW	CosineLR	3
LayoutLMv3	1280	PubLayNet	2×10^{-4}	AdamW	CosineLR	3
YOLOv8	1280	COCO	1×10^{-2}	SGD	OneCycleLR	2
YOLOv8	640	COCO	1×10^{-2}	SGD	OneCycleLR	4

Table 7.2: Hyperparameters and variants of the models used for training - model, input size, initial weights, learning rate, optimizer, scheduler, and batch size.

⁶<https://huggingface.co/HYPJUDY/layoutlmv3-base-finetuned-publaynet>

7.2 Multi-modal Classification

The second type of model in our solution is a multi-modal classifier. The main issue this model tries to solve is that in some cases, the segmentation model extracts bounding boxes/segmentations correctly but the final class predictions are incorrect.

This is mainly the case if the visual features are similar, for instance, footnotes, paragraphs, and listings all use an identical font and can only differ in subtle details that may not get picked up by the segmentation model. On the other hand, the text in each type of element is often different - e.g. listings contain personal names, names of animal species, or enumerations, while paragraphs often contain common words.

Therefore, using both text and image features in a multi-modal way might improve the number of correct predictions, especially in cases where semantics are important. Alternatively, should the model not yield better results, it can still be useful, e.g. for further validation of the document layout analysis results, where we can be more certain if both classifier and segmentation predictions match.

7.2.1 Models

In general, there are two types of models that we employ. The first one is a fusion model that uses early fusion to generate the prediction. Such an architecture comprises a pair of vision and text models working in parallel and outputting a set of modality-specific features. These features are concatenated and fed to a perceptron that generates the final prediction. In addition, it is also possible to easily introduce other modalities such as layout information - the size of the bounding box, position, etc., which could theoretically further improve classification accuracy.

The second approach is based on the use of a multi-modal Transformer. Here, our main candidate is LayoutLMv3, since it is state-of-the-art in sequence classification datasets such as RVL-CDIP. Because the Transformer is multi-modally pre-trained and accepts input in the form of embeddings, we are constrained with only image and text modalities.

Fusion-based Model

We provide and analyze many configurations of the fusion-based model. The architecture of the network is depicted in Fig. 7.2. To process the textual modality, we use a German pre-trained variant of BERT. The visual stream is handled by a vision Transformer - either ViT or Swin Transformer V2.

The architecture of the model follows early fusion and thus, all of the variants of the Transformers are used without a classification head on top and serve as feature extractors. In our configuration, BERT produces a matrix with the shape of (512, 768), corresponding to 512 768-dimensional word embeddings. Similarly, ViT and Swin output either 197 or 49 of identically long patch embeddings.

Subsequently, the feature vectors extracted by BERT and Vision Transformer are fused into a single vector and fed into a perceptron, which performs the multi-modal classification. Theoretically, we could concatenate the word and patch embeddings directly but there are three main issues with this approach.

Firstly, such a concatenated vector would require a large number of weights in the perceptron’s input layer and, as a consequence, would increase memory consumption significantly. Secondly, compared to the number of trainable weights in the fully connected layers, the number of training samples is relatively small and the network could easily underfit or require many more training iterations. Finally, the features extracted from each modality should reflect the sequence as a whole, for which a single pass through a fully-connected layer is not optimal.

Therefore, we introduce an additional layer on top of each Transformer output, which is an LSTM, similarly to [18]. In our case, the LSTM operates bidirectionally. The outputs from both directions are concatenated, resulting in much more compact 128 or 256-dimensional vectors, depending on the hyperparameter configuration. To reduce the chance of overfitting during training, the LSTM output is passed through a dropout layer with 30% probability of being zeroed. Finally, the vector is modified by the ReLU activation and concatenated.

Depending on the hyperparameters, the fusion model can also employ information from the bounding box of the annotation. The data is passed via perceptron with a single 64-neuron hidden layer, that outputs a 16-dimensional vector. Subsequently, such a vector is concatenated with text and image features and fed to the fusion MLP. Note that we do not use the additional LSTM layer for the bounding box features because the data is not sequential and already has very low dimensionality.

Multi-modal Transformer Model

In addition to the fusion-based architecture, we also use the base version of LayoutLMv3. The architecture of this model is identical to the one from the original paper, which can be seen in Figure 3.8.

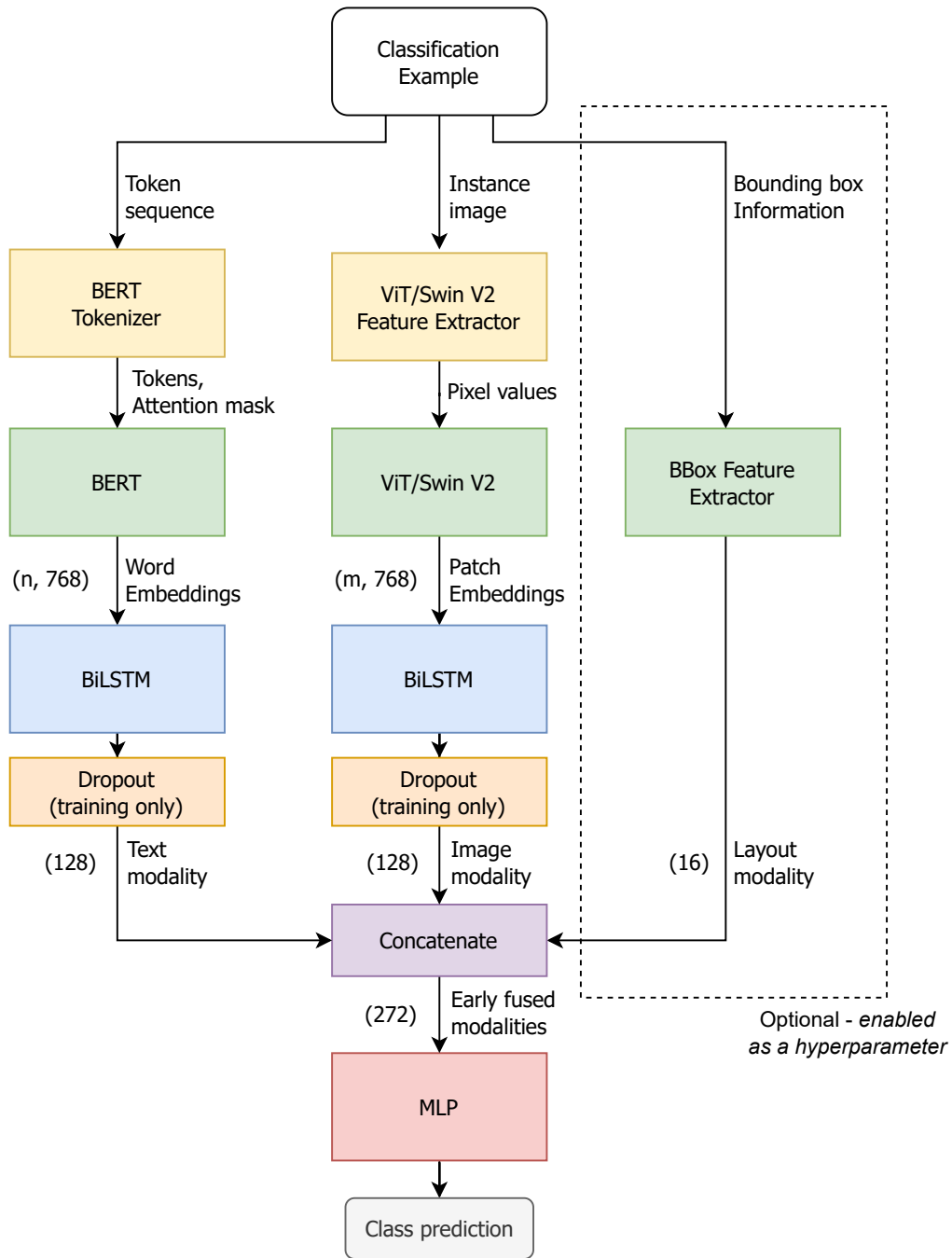


Figure 7.2: Architecture of the early-fusion model. The right side of the network (Bounding Box Feature Extractor) is optional and is enabled only in certain hyperparameter configurations of the network. The shape of the image/text features is either a 128 or 256-dimensional vector.

7.2.2 Preprocessing

Since the input to the network is different than in the case of instance segmentation, and the model must be fed with an image of a document component and its text, it is necessary to apply additional preprocessing to make our dataset (or the output from the segmentation model) digestible for classification.

For training, we can either employ images directly from the ground truth dataset or we can use the outputs of the segmentation model. The segmentation outputs could theoretically provide better results, since the classifier is trained on real-world inputs, but are likely to be erroneous. Therefore, we only train on ground truth samples.

The image preprocessing is very straightforward since we need to pass the image located in the extracted bounding box to the classifier, and thus only cropping is necessary. The text of the document component is extracted using the Tesseract model described in Section 6.5. The OCR model is run through the PyTesseract library with the page segmentation mode set to a uniform block of text, which should be the most appropriate mode for our type of data.

It is worth noting that these operations are relatively expensive and do not change over the course of training. Therefore, we preprocess the dataset once and save the results for reuse.

The implementation of the preprocessing itself is depicted in Figure 7.3. To map the document layout analysis examples into ones suitable for classification, we create a `ClassificationDatasetMapper` component. This component takes the images as well as the COCO-formatted annotations and produces examples that can be directly digested by the multi-modal classifier. Additionally, the resulting classification examples can be loaded using Huggingface `Datasets` library⁷ [37], which provides formats for various frameworks, including PyTorch and TensorFlow, so in theory, the dataset can be easily used with other frameworks.

Internally, `ClassificationDatasetMapper` calls another custom component - `InstancePreprocessor`, which performs the actual preprocessing of each annotation. As input, the preprocessor receives a COCO-annotated instance and the path to the image of the annotation. Based on the bounding box of the instance, the original image is cropped and passed to the OCR model to extract tokens with their bounding boxes (these are needed as input to LayoutLMv3).

In addition to the text and image features, we also store layout informa-

⁷<https://huggingface.co/docs/datasets/index>

tion about the instance’s bounding box - specifically, its area and coordinates in the original image. The values are normalized by the size of the image to make the features consistent across all instances. Finally, the tokens, their bounding boxes, annotation labels, and position-related features are stored in a JSON array that can be loaded via a custom Huggingface Datasets script.

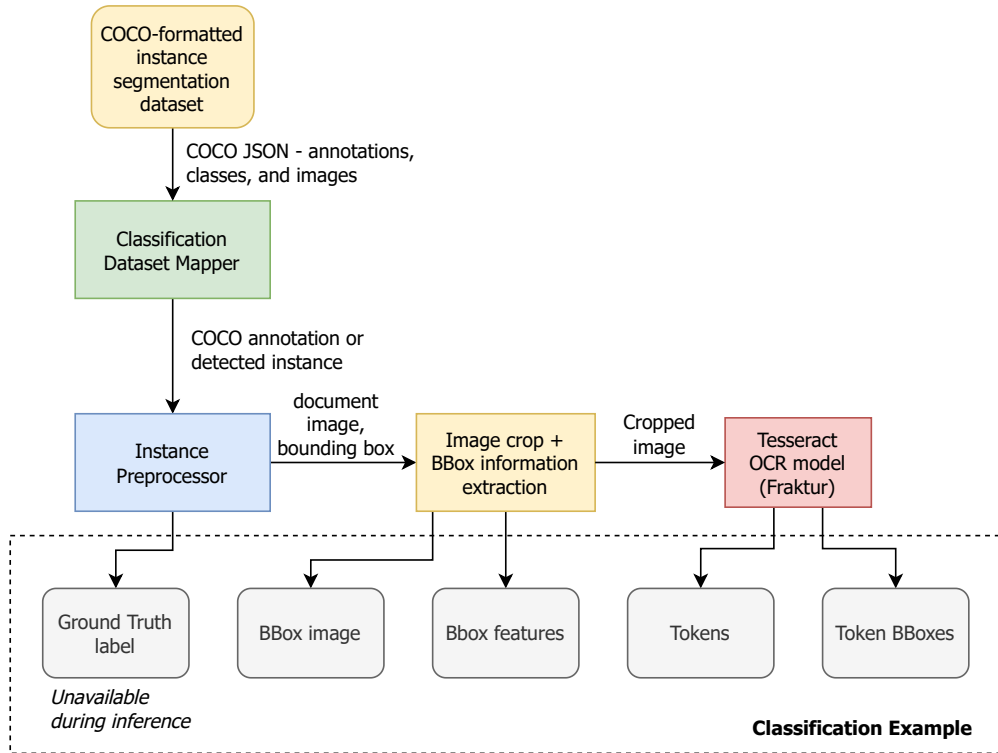


Figure 7.3: Visualization of preprocessing for a single annotation in the COCO format. Annotations are loaded by the dataset mapper, which calls the instance preprocessor to preprocess features for classification.

7.2.3 Implementation

Each of the classifier models is built using the PyTorch framework. To implement BERT, ViT, Swin Transformer V2, and LayoutLMv3, we use the Huggingface Transformers library⁸ [75]. Training, evaluation, and inference are handled by PyTorch Lightning⁹ [16], which also performs the serialization of the models for later use.

Note that LayoutLMv3 uses OCR internally by default, but we disable

⁸<https://huggingface.co/docs/transformers/index>

⁹<https://www.pytorchlightning.ai/index.html>

this feature because the extracted text is already available before training as a result of the preprocessing. This behavior is virtually identical, except that the OCR does not need to be performed each time the model is trained and the text is instead loaded directly with the dataset.

7.2.4 Training

Training the classifiers is relatively straightforward, however, our dataset suffers from severe class imbalance. If such an issue is not addressed, the trained models are very likely to pay attention to the most common classes and fail to recognize ones that occur infrequently, such as tables or centered text.

In our case, an appropriate technique to avoid this issue is oversampling. We use PyTorch’s `WeightedRandomSampler` class, which is able to sample the training examples based on the weight of their classes. The weights are computed as an inverse of their frequency, making rare classes appear more often.

To measure the quality of the model, we cannot fully rely on accuracy either, since the result is biased towards the most common classes. Therefore, three additional metrics are employed - precision, recall, and the F1 score. All three metrics are macro-averaged - that is, we compute them separately for each class and then take their arithmetic mean. This effectively treats each class with the same weight and does not skew the results towards the most common classes.

Arguably the most important is the F1 measure as it is essentially a function of both precision and recall (more precisely, their harmonic mean). A high F1 (e.g. above 0.95) implies that both precision and recall are high as well.

The training loop is standard and most of it can be handled internally by the PyTorch Lightning framework. The networks are trained for 20 epochs, where in each epoch the sampler returns a total number of examples equal to the size of the training dataset. Note that this may result in the model seeing only a subset of the training examples in each epoch, however, all examples are seen eventually in the subsequent epochs.

As an optimizer, we use AdamW, which is known to work very well with many Transformer networks. Depending on the model configuration, the learning rate of the optimizer can be scheduled by linear warmup and linear decay. To compute the gradient, the standard negative log-likelihood loss is employed with outputs from the log softmax function. Additionally, as most of our models run on CUDA, we utilize 16-bit floating point precision,

which speeds up the entire training process and significantly reduces memory consumption.

At the end of each epoch, the classifier is evaluated on the test data, and if it achieves the best F1 score, both its weights and configuration are checkpointed and used for comparison with other hyperparameter variants. For easier tracking of the models, we log all training and evaluation values via Weights and Biases¹⁰ [6].

7.2.5 Configuration and Hyperparameters

In order to find the best model, we test different configurations of hyperparameters. While there are not many hyperparameters that we can change in LayoutLMv3 without changing its structure, the fusion model is very flexible.

All the hyperparameters we evaluate and compare are included in Table 7.3. For LayoutLMv3, this involves two different learning rates that are quite common for Transformers - 1×10^{-5} and 5×10^{-5} , three different learning rate scheduler configurations - disabled, 1000 iterations, and 1500 iterations. All models use a batch size of 4.

The fusion model uses the same set of hyperparameters as LayoutLMv3 and introduces additional ones. Here, we also test the size of each modality-specific feature vector that is extracted from the BiLSTM layer. This can be either a 128 or 256-element vector for both image and text feature extractors. As for the feature extractor processing the bounding box modality, it can either be turned on or off, and it returns a lower dimensional vector compared to image/text, as the number of input features is very limited.

The number of parameters in the LayoutLMv3 is surprisingly small, and both fusion models - BERT + ViT and BERT + Swin V2 are almost 60% larger. The sizes are visible in Table 7.4. All three models use pre-trained weights from Huggingface, which are the following:

- `google/vit-base-patch16-224-in21k` for ViT
- `microsoft/swin-base-patch4-window7-224-in22k` for Swin Transformer V2
- `bert-base-german-cased` for BERT
- `microsoft/layoutlmv3-base` for LayoutLMv3

¹⁰<https://wandb.ai/site>

	LayoutLMv3	BERT + ViT	BERT + Swin V2
Optimizer	AdamW	AdamW	AdamW
Learning rate	$[1 \times 10^{-5}, 5 \times 10^{-5}]$	$[1 \times 10^{-5}, 5 \times 10^{-5}]$	$[1 \times 10^{-5}, 5 \times 10^{-5}]$
LR sched. steps	[0, 1000, 1500]	[0, 1000, 1500]	[0, 1000, 1500]
Text vector size	-	[64, 128]	[64, 128]
Image vector size	-	[64, 128]	[64, 128]
BBox features	-	[True, False]	[True, False]
# Variants	6	48	48

Table 7.3: Hyperparameters - optimizer, learning rate, learning rate scheduler steps, size of the extracted text vector, size of the extracted image vector, and whether to include bounding box feature extractor. Columns two, three, and four denote each individual model. Square brackets indicate variants of the hyperparameter.

Model Name	# Params
LayoutLMv3	125M
BERT + ViT	197M
BERT + Swin V2	197M

Table 7.4: Approximate number of parameters in each classifier model.

7.3 Multi-modal System

The training of both segmentation and classifier networks produces models that can be combined to create a system capable of multi-modal document layout analysis, as shown in Figure 7.1 at the beginning of this chapter. To implement such a system, we primarily use Detectron2 because it includes the COCO evaluation, which we use to measure performance, as well as the implementation of both the Mask R-CNN and LayoutLMv3-based instance segmentation models.

7.3.1 YOLOv8 Compatibility

Arguably the only issue with Detectron2 is the fact that YOLOv8 is not compatible with it and runs on the Ultralytics platform instead. Fortunately, since we only need each model for inference, it is possible to create a compatibility layer between the frameworks.

For inference, Detectron2 defines an `inference_on_dataset` function that takes a model object as its argument and invokes its `__call__` method.

Such a method must return an `Instances` object, which contains standard object detection/instance segmentation outputs such as bounding boxes, segmentation masks, predicted classes, confidences, etc. Thus, the implementation itself is a simple wrapper class over the YOLOv8 model that overrides the `__call__` method and maps the YOLOv8 output to an `Instances` object.

7.3.2 Communication Between Segmentation Model and Classifier

The results of the segmentation model must be passed to the multi-modal classifier to predict instances of each component in the document. Analogous to training, the classifier requires the detected instances to be preprocessed.

The preprocessing is almost identical to the one applied during the classifier training, which was previously shown in Figure 7.3 and discussed in Section 7.2.2. The only difference is that we use the instance preprocessor component directly - i.e. it is called by the Detectron2 framework instead of the dataset mapper.

Note that the classifier should only predict the instances detected with reasonably high confidence as those with low confidence are likely to be FP. Therefore, detections below a certain threshold - in our case 0.5, are not fed into the classifier and we use the labels predicted by the instance segmentation model instead.

For convenience, the communication between the classifier and the segmentation model is facilitated by a wrapper called `InstanceClassifier`. Internally, the instance classifier contains the instance preprocessor as well as the multi-modal classifier and its interface is compatible with the rest of the Detectron2 ecosystem.

The component accepts a specific `Instances` object generated from the input image by the segmentation model, extracts and preprocesses visual and textual features, and passes them to the classifier. Finally, the predicted class from the classifier is mapped back to the original `Instances` object and returned so that it can be used for evaluation or inference.

7.3.3 Evaluation

The evaluation of the system is relatively straightforward and mostly performed by Detectron2. As mentioned before, we use the COCO evaluation metrics to determine the quality of each model, which is already a built-in part of Detectron2 in the `COCOEvaluator` class.

To use our classifier during the evaluation, we create a class that inherits from `COCOEvaluator` and overrides its `process` method where it calls the `InstanceClassifier` to ensure that the prediction is performed multimodally.

7.4 Utilization of Textual Modality

In Chapter 6 we have trained an OCR model with satisfactory CER and WER, however, the extracted text is still erroneous. This section attempts to test the effect of these errors on the overall classification. The idea is that a lower-quality text should also produce a classifier with worse results, as it essentially has incorrect information, and only the image part of the output is reliable.

Furthermore, it is important to note that the quality of the produced text depends not only on the OCR model itself but also on the quality of the segmentation model since it provides the input to the OCR. If the segmentation model performs poorly and the detected instances are of low quality - for example, only a part of the instance is detected and the text is cut off, the output from the OCR is likely to be poor as well.

Therefore, for the purpose of this experiment, we assume that the segmentation model is optimal and that the error in the text is only caused by the OCR system itself. To do so, we use ground truth annotations in the same way as we did in Section 7.2.4. The modification of this task is that we introduce an artificial noise in the form of randomly changed characters in each token.

We generate the noise using a uniform random generator - that is, each character has the same probability of being replaced by a randomly generated one. The main interest is in low probabilities as they are more representative of real-world scenarios. Overall, the set of low probabilities ranges from 0 to 30% with 5% increments, and we additionally use 50%, 75%, and 100% probabilities to test extreme values.

Overall, we obtain 10 variants of the dataset, one for each probability. Such a modified dataset is then used to train a classifier. During the training, we measure standard performance metrics such as F1, precision, recall, and accuracy. The employed classifier is a BERT + ViT model that is trained for 20 epochs with a learning rate of 1×10^{-5} and equally large 128-element modality feature vectors.

8 Results

This chapter discusses the results of the models trained in Chapter 7 as well as potential future extensions to improve the performance. Note that for most of the used metrics, we use 0–100 range normalization instead of 0–1. This includes accuracy, precision, recall, F1, and all (m)AP metrics.

8.1 Instance Segmentation Results

In Section 7.1 we utilize 3 types of models for instance segmentation - Mask R-CNN, LayoutLMv3 with Cascade R-CNN, and YOLOv8. As mentioned in Section 6.4, we evaluate the task performance using COCO evaluation metrics, where we use $AP@[0.50:0.95]$ (mean of mAP at IoU levels from 50% to 95% with 5% increments), AP50 (mAP at IoU = 50%), and AP75 (mAP at IoU = 75%). These metrics can be calculated either from the predicted bounding boxes or from the predicted segmentation masks.

The results regarding bounding boxes are shown in Table 8.1, while the results for segmentation masks are shown in Table 8.2. The most important metric is $AP@[0.50:0.95]$ because it is a combination of 10 different mAP values.

In terms of bounding box average precision, the two best models are variants of YOLOv8 that process either 640p or 1280p input. The 1280p variant achieves an $AP@[0.50:0.95]$ of **83.64**, while the 640p one attains an $AP@[0.50:0.95]$ of 81.34. These results are surprising because both variants outperform the much larger LayoutLMv3-based model, which is only competitive when trained with the PubLayNet weights and achieves an $AP@[0.50:0.95]$ of 79.45.

On the other hand, when using evaluation metrics from segmentation masks of each instance, the best variant is the LayoutLMv3-based model with the PubLayNet weights, achieving an $AP@[0.50:0.95]$ of **79.77**. The second best model is the 1280p variant of YOLOv8, closely followed by the LayoutLMv3-based model with the default pre-training weights. These models score an $AP@[0.50:0.95]$ of 76.34 and 75.22 respectively. The least competitive model is the 640p variant of YOLOv8 with 55.20 $AP@[0.50:0.95]$.

Model	Initial weights	Input size	AP@[0.50:0.95]	AP50	AP75
Mask R-CNN	COCO	1280	73.55	94.75	88.08
LayoutLMv3	PubLayNet	1280	79.45	95.46	91.76
LayoutLMv3	Default	1280	73.59	91.64	82.38
YOLOv8	COCO	1280	83.64	95.68	94.37
YOLOv8	COCO	640	81.34	93.46	91.96

Table 8.1: Bounding box COCO metrics of each model - Mask R-CNN, LayoutLMv3 with Cascade R-CNN, and YOLOv8. The best values are denoted in bold.

Model	Initial weights	Input size	AP@[0.50:0.95]	AP50	AP75
Mask R-CNN	COCO	1280	75.12	93.84	89.07
LayoutLMv3	PubLayNet	1280	79.77	95.60	90.99
LayoutLMv3	Default	1280	75.22	91.80	85.77
YOLOv8	COCO	1280	76.34	95.81	86.54
YOLOv8	COCO	640	55.20	86.34	54.89

Table 8.2: Segmentation COCO metrics of each model - Mask R-CNN, LayoutLMv3 with Cascade R-CNN, and YOLOv8. The best values are denoted in bold.

8.2 Multi-modal Classification Results

Section 7.2 is concerned with training a multi-modal classifier that takes a detected instance from the segmentation model and predicts its class. Here, we evaluate many configurations of three different models - LayoutLMv3 (its sequence classification variant) and two fusion-based models - BERT + ViT and BERT + Swin V2. In total, we run 6 different configurations of LayoutLMv3 and 48 different configurations each for BERT + ViT and BERT + Swin V2, for a total of 102 different runs.

We select the best model based on its macro-averaged F1, as it combines both precision and recall and has no bias towards the most frequent classes. From all configurations, we collect the three best ones for each model, which are shown in Table 8.3. The hyperparameters used for these models are shown in Table 8.4.

The best F1 score of **97.38** is achieved by the fusion model comprising BERT and ViT. As shown in Table 8.4, this configuration (*Fusion-34*) uses

a learning rate of 1×10^{-5} , 1500 steps in the learning rate scheduler, disables the bounding box modality, and its extracted feature vectors for text and image are 128 and 256 elements long.

The fusion model using BERT and Swin Transformer V2 with the same hyperparameters and an F1 of 97.24 is very close to the best variant. On the other hand, none of the LayoutLMv3 configurations comes close in terms of F1, and the best variant scores only 95.52. However, this can be explained by the fact that the model is not pre-trained for German, and its number of parameters is almost 50% less than that of the two fusion models.

In all cases presented here, the learning rate used is 1×10^{-5} , which probably means that the other option of 5×10^{-5} is too high. As for LayoutLMv3, configurations using the 5×10^{-5} learning rate even diverge in some cases and fail to learn the dependencies altogether.

Model	Configuration	F1	Precision	Recall	Accuracy	Loss
LayoutLMv3	LayoutLMv3-2	95.52	95.93	95.28	94.45	0.1651
LayoutLMv3	LayoutLMv3-1	95.27	96.57	94.19	94.01	0.1705
LayoutLMv3	LayoutLMv3-3	94.43	95.67	93.55	94.23	0.1932
BERT + ViT	Fusion-34	97.38	98.16	96.72	96.28	0.1550
BERT + ViT	Fusion-2	97.21	98.02	96.46	96.20	0.1937
BERT + ViT	Fusion-1	96.95	97.34	96.59	95.69	0.1853
BERT + Swin V2	Fusion-34	97.24	97.92	96.62	96.13	0.1477
BERT + Swin V2	Fusion-7	96.85	97.80	95.98	95.69	0.1984
BERT + Swin V2	Fusion-26	96.54	96.99	96.14	95.18	0.1843

Table 8.3: F1, precision, recall, accuracy, and loss of the top 3 variants of each model. The best values are denoted in bold. F1, precision, and recall are macro averaged. Accuracy is micro-averaged.

Configuration	Learning rate	Scheduler steps	BBox features	Text vector size	Image vector size
LayoutLMv3-1	1×10^{-5}	None	-	-	-
LayoutLMv3-2	1×10^{-5}	1000	-	-	-
LayoutLMv3-3	1×10^{-5}	1500	-	-	-
Fusion-1	1×10^{-5}	None	True	128	128
Fusion-2	1×10^{-5}	None	True	128	256
Fusion-7	1×10^{-5}	1000	True	256	128
Fusion-26	1×10^{-5}	None	False	128	256
Fusion-34	1×10^{-5}	1500	False	128	256

Table 8.4: Hyperparameter configurations used in the top 3 best models in Table 8.3. Learning rate, number of steps in the learning rate scheduler, whether to use bounding box features, size of the extracted text vector from BERT, and size of the extracted image vector from ViT/Swin Transformer V2.

8.3 Results of Instance Segmentation with Multi-modal Classifier

Combining the trained instance segmentation models with multi-modal classifiers makes it possible to create and evaluate our multi-modal system. Due to the lower performance of the LayoutLMv3 classifier, we only present results from the two best variants of the fusion models - i.e. BERT + ViT and BERT + Swin V2.

Both classifiers are evaluated in combination with each instance segmentation model to observe potential improvements in their performance. Note that only instances with confidence greater than 0.5 are processed by the classifier to prevent it from seeing potential false positives or ill-recognized instances.

Analogous to Section 8.1, we compute COCO AP scores based on the predicted bounding boxes and segmentation masks of instances. The results regarding bounding boxes are shown in Table 8.5, while the results obtained from segmentation masks are shown in Table 8.6.

Although we are able to improve the performance of the 640p YOLOv8 variant, the rest of the results do not exceed the image-only baseline. The combination of 640p YOLOv8 with the best configuration of BERT + Swin Transformer V2 results in the second-best model overall, achieving **82.23** bounding box AP@[0.50:0.95], which is around **0.89** better than the uni-modal variant of the 640p YOLOv8.

Similarly, if we consider segmentation masks of the instances, we obtain

an AP@[0.50:0.95] of 55.78, which is 0.58 more than the baseline. Note that since the baseline is low, the improvement from the classifier does not justify its use as it is essentially the second-worst model. On the other hand, bounding boxes are likely more suitable for our use case than the segmentation masks. Using the best configuration of the BERT + ViT model, we can see slightly lower improvements where the 640p YOLOv8 score is 82.18 AP@[0.50:0.95] on bounding boxes and 55.78 AP@[0.50:0.95] on segmentation masks.

Example output of the 640p and 1280p variants of YOLOv8 combined with the BERT + Swin Transformer V2 classifier can be seen in Figures 8.1 and 8.2. Both variants of YOLOv8 detect bounding boxes very well, however, the YOLOv8 640p model does not produce satisfactory segmentation masks in some cases.

Segmentation model	Input size	Initial weights	Classifier	AP@[0.50:0.95]	AP50	AP75
Mask R-CNN	1280	COCO	BERT + ViT	70.25	89.62	84.20
Mask R-CNN	1280	COCO	BERT + Swin V2	72.06	91.96	86.21
LayoutLMv3	1280	PubLayNet	BERT + ViT	78.01	93.41	89.71
LayoutLMv3	1280	PubLayNet	BERT + Swin V2	78.44	93.83	90.21
LayoutLMv3	1280	Default	BERT + ViT	72.48	89.74	81.11
LayoutLMv3	1280	Default	BERT + Swin V2	72.12	89.19	80.65
YOLOv8	1280	COCO	BERT + ViT	82.35	94.22	93.06
YOLOv8	1280	COCO	BERT + Swin V2	81.42	93.02	91.99
YOLOv8	640	COCO	BERT + ViT	82.18	94.47	93.08
YOLOv8	640	COCO	BERT + Swin V2	82.23	94.55	93.16

Table 8.5: Bounding box COCO metrics of each segmentation model - Mask R-CNN, LayoutLMv3 with Cascade R-CNN, and YOLOv8, in combination with a multi-modal classifier - BERT + ViT or BERT + Swin V2. Improved values over the original baseline in Table 8.1 are denoted in bold.

Segmentation model	Input size	Initial weights	Classifier	AP@[0.50:0.95]	AP50	AP75
Mask R-CNN	1280	COCO	BERT + ViT	71.55	88.80	85.15
Mask R-CNN	1280	COCO	BERT + Swin V2	73.42	91.15	87.10
LayoutLMv3	1280	PubLayNet	BERT + ViT	78.17	93.57	89.44
LayoutLMv3	1280	PubLayNet	BERT + Swin V2	78.72	94.00	89.93
LayoutLMv3	1280	Default	BERT + ViT	73.51	89.89	83.64
LayoutLMv3	1280	Default	BERT + Swin V2	73.39	89.33	83.86
YOLOv8	1280	COCO	BERT + ViT	75.05	94.35	84.69
YOLOv8	1280	COCO	BERT + Swin V2	74.28	93.15	84.35
YOLOv8	640	COCO	BERT + ViT	55.78	87.57	55.07
YOLOv8	640	COCO	BERT + Swin V2	55.79	87.50	55.33

Table 8.6: Segmentation box COCO metrics of each segmentation model - Mask R-CNN, LayoutLMv3 with Cascade R-CNN, and YOLOv8, in combination with a multi-modal classifier - BERT + ViT or BERT + Swin V2. Improved values over the original baseline in Table 8.2 are denoted in bold.

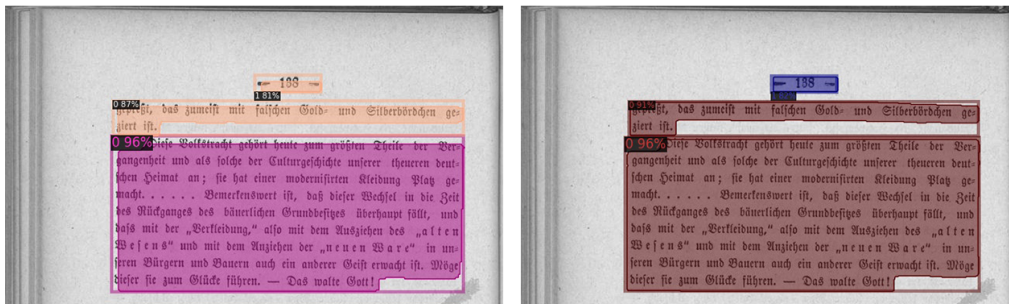


Figure 8.1: Example output of the 640p YOLOv8 with the BERT + Swin Transformer V2 classifier (left) and output of the 1280p YOLOv8 with an identical classifier (right). The classes of the individual instances and their confidences are shown in the upper left corner of the bounding box. This example contains a page number (label 1) and a paragraph (label 0). Generated via Detectron2.

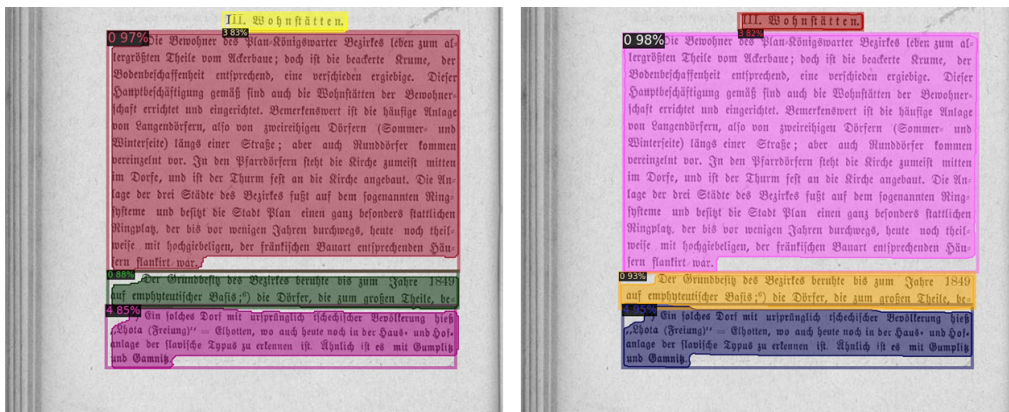


Figure 8.2: Example output of the 640p YOLOv8 with the BERT + Swin Transformer V2 classifier (left) and output of the 1280p YOLOv8 with an identical classifier (right). The classes of the individual instances and their confidences are shown in the upper left corner of the bounding box. This example contains a heading (label 3), a paragraph (label 0), and a footnote (label 4). Generated via Detectron2.

8.4 Baseline Performance of Textual and Visual Modalities

In addition to the performance of the multi-modal classifiers, we also evaluate the performance of BERT, ViT, and Swin Transformer V2 which, as mentioned before, are all unimodal. Therefore, these measurements should provide a good estimate of the importance of both image and textual modality.

The results of each model are shown in Table 8.7. We use a similar configuration for all models. Since there is no need to use a BiLSTM to extract features, a conventional classification head is used instead. Note that due to the overall architecture of the multi-modal fusion network, the models are not one-to-one comparable. Each model is trained with 1500 warmup steps, the AdamW optimizer with a learning rate of 1×10^{-5} , and a batch size of 6.

Surprisingly, both ViT and Swin Transformer V2, which are image models, are able to achieve relatively high F1 compared to our best multi-modal classifier. ViT achieves the best result with an F1 of **96.52**, which is 0.86 less than the best multi-modal variant. This likely indicates that most of the multi-modal models use mainly visual features, while textual features are rarely used. On the other hand, BERT - a text-only model- can also achieve a surprisingly high F1.

Model	Modality	F1	Precision	Recall	Accuracy	Loss
BERT	Text	83.89	91.39	86.69	81.63	0.4318
ViT	Image	96.52	97.23	95.84	94.67	0.1888
Swin V2	Image	96.31	96.62	96.11	94.43	0.2095

Table 8.7: Classification performance of BERT, ViT, and Swin Transformer V2 on the dataset. Best values are denoted in bold.

8.5 Utilization of Textual Modality

In Section 7.4 we perform an experiment to test the importance of the textual modality on the classification. In addition to measuring the performance, we also try to estimate how much error is introduced when the characters are randomly replaced with a given probability.

To estimate this, we can use standard CER and WER metrics and compare each variant of the modified dataset to the reference text. Note that we have not annotated the entire dataset for OCR, and thus such metrics

cannot be computed directly. However, we can use the evaluation dataset created in Section 6.3.1, which should provide a relatively good estimate.

We run our fine-tuned OCR model on the examples and then modify its output according to the probabilities - i.e. for each prediction we replace each character with a given probability p . The results of all variants are shown in Table 8.8. As expected, higher probabilities significantly increase CER and WER.

Random character probability	CER [%]	WER [%]
0%	1.23	7.18
5%	5.47	29.91
10%	9.86	46.34
15%	13.83	58.35
20%	18.10	66.98
25%	22.26	75.12
30%	26.34	79.87
50%	43.15	93.00
75%	64.70	99.00
100%	85.16	100.00

Table 8.8: Approximate character error rate and word error rate for each probability. Best values are denoted in bold.

Subsequently, we measure the performance of the model on each variant of the modified dataset. Analogous to the training procedure of the multi-modal classifier in Section 7.2.4, the best F1 on the evaluation data is extracted. The general idea is that more randomly replaced characters should lead to an overall worse F1.

Interestingly, except for 100% random text input, we do not observe any significant changes in F1. The best metrics for each variant of the dataset are shown in Table 8.9. Note that we can also be concerned with F1 over time, which is shown in Figure 8.3. To get a better estimate of the metric, we apply exponential moving average smoothing with a factor of 1 to all runs.

While the text containing 100% of random characters is the worst, most of the variants perform similarly on average. The model trained on the original dataset still achieves the highest F1 of **96.80**, however, comparable results can be achieved by the dataset with 50% or even 75% noise probability.

As noted in the previous section, this likely suggests that the text does not play as crucial a role as we initially expected, and that most of the information about the instance can be predicted from the visual modality alone.

Random character probability	F1	Precision	Recall	Accuracy	Loss
0%	96.80	98.05	95.70	96.06	0.1540
50%	96.78	97.47	96.18	95.33	0.1790
75%	96.74	98.03	95.59	95.40	0.2174
25%	96.67	97.17	96.25	95.11	0.1760
20%	96.50	97.64	95.49	95.26	0.1780
15%	96.41	97.34	95.53	95.11	0.1730
30%	96.40	97.12	95.78	95.11	0.1913
10%	96.40	97.44	95.47	95.47	0.1677
5%	96.25	97.05	95.57	95.26	0.1840
100%	95.69	97.28	94.38	94.96	0.1874

Table 8.9: F1, precision, recall, accuracy, and loss of the BERT + ViT model with *Fusion-1* configuration from Table 8.4 except that bounding box features are turned off. Best values are denoted in bold.

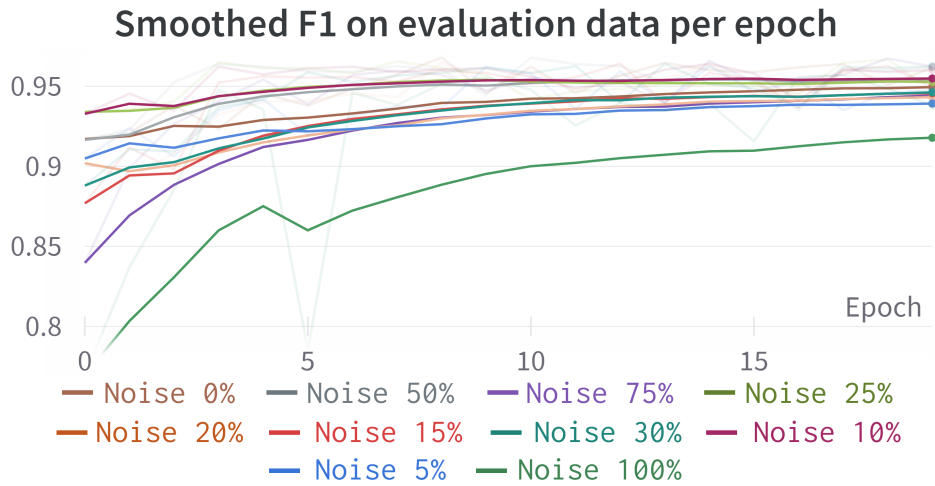


Figure 8.3: Validation F1 per epoch smoothed with an exponential moving average with a smoothing factor of 1. The non-smoothed version is displayed with lower opacity.

8.6 Possible extensions

There are several extensions or adjustments that could be applied to our solution to improve the multi-modal results.

One idea would be to use a late fusion instead of the early fusion, where the classification is performed by both the instance segmentation model and the classifier. The classifier could be either multi-modal (e.g., the current models we use) or purely text-based (BERT-like).

However, implementing this approach would likely require a large number of modifications to the instance segmentation models since neither YOLOv8 nor the models implemented in Detectron2 provide a simple solution for obtaining class probabilities for detected instances. Such functionality is crucial because these probabilities are necessary to build the late-fusion vector to perform the final classification.

Another extension could be to introduce additional modalities, either to the instance segmentation model itself - e.g. employ the detected tokens alongside visual data, or add more sources of information to the classifier, such as a sequence of all detected instances.

Finally, our solution could also be tested on more complex data, where more information is encoded in the text, and the multi-modality could be better exploited.

9 Conclusion

The aim of this thesis was to employ multi-modality in order to improve the performance of models designed for document processing using solely one modality.

We created a historical "Heimatkunde" dataset, which in its current form is applicable to document layout analysis as well as for classification. To extract the textual modality from the data, we used an OCR approach, for which we reviewed many frameworks and trained a Tesseract model on a subset of our dataset. Overall, the resulting OCR model achieves excellent CER and WER values of **1.23%** and **7.18%** respectively, which is comparable to state-of-the-art models on similar datasets.

Subsequently, using our dataset, we trained and evaluated three different instance segmentation models - Mask R-CNN, YOLOv8, and LayoutLMv3 with Cascade R-CNN, which were combined with a multi-modal classifier. The best configurations of the classifier are early-fusion models consisting of the BERT model for text analysis and ViT or Swin Transformer V2 for image processing. The BERT + ViT variant is able to attain an F1 of **97.38**, while the BERT + Swin V2 achieves an F1 of 97.24.

For document layout analysis, the second-best configuration is the combination of BERT + Swin Transformer V2 with the 640p variant of YOLOv8, which improves the bounding box AP@[0.50:0.95] from 81.34 to 82.30. On the other hand, our best result is obtained by YOLOv8 using 1280p input, which achieves an AP@[0.50:0.95] of **83.64** using only image modality.

However, we find that adding noise to the textual modality does not significantly affect the classifier's results. The image-based models - ViT and Swin Transformer V2 are able to achieve an F1 score that is close to the multi-modal models, suggesting that most of the information is already encoded in the image, which can be efficiently processed by the segmentation model alone.

Another contribution of this thesis is our created dataset and the source code for the experiments, which are available on our GitHub repository¹.

In future work, we will primarily focus on improving our solution. It might be interesting to implement multi-modality directly into the segmentation model - for example, to make it process tokens alongside visual information, which could potentially provide better results. Furthermore, our approach could also be extended to other modalities, such as video.

¹<https://github.com/honzikv/multimodal-document-processing-thesis>

List of Abbreviations

ANN Artificial neural network.

AP Average precision.

API Application programming interface.

AUC Area under curve.

BBox Bounding box.

BERT Bidirectional Encoder Representations from Transformer.

BiLSTM Bidirectional LSTM.

CBOW Continous bag of words.

CER Character error rate.

CNN Convolutional neural network.

COCO Common Objects in Context.

CORD Consolidated Receipt Dataset for Post-OCR Parsing.

CV Computer vision.

CVAT Computer Vision Annotation Tool.

DLA Document Layout Analysis.

FC Fully-connected.

FCNN Fully convolutional neural network.

FN False negative.

FP False positive.

FRK Tesseract Fraktur model.

FUNSD Form Understanding in Noisy Scanned Documents.

GPT Generative Pre-Training for Transformers.

IoU Intersection over union.

LM Language Model.

LR Learning rate.

LSTM Long short-term memory network.

mAP Mean average precision.

MDC Multi-label Document Classification.

MIM Masked Image Modeling.

ML Machine learning.

MLM Masked Language Modeling.

MLP Multi-layer Perceptron.

MVLM Masked Visual Language Model.

NLP Natural language processing.

OCR Optical character recognition.

R-CNN Region-based convolutional neural network.

ReLU Rectified Linear Unit.

RNN Recurrent neural network.

ROI Region of Interest.

RPN Region proposal network.

RVL-CDIP Ryerson Vision Lab Complex Document Information Processing.

SGD Stochastic gradient descent.

SOTA State-of-the-art.

TN True negative.

TP True positive.

ViT Vision Transformer.

VQA Visual Question Answering.

VTN Video Transformer.

WER Word error rate.

WIT Wikipedia Image-Text Dataset.

WPA Word-Patch Alignment.

YOLO You only look once.

Bibliography

- [1] Ashutosh Adhikari et al. *DocBERT: BERT for Document Classification*. 2019. arXiv: 1904.08398 [cs.CL].
- [2] Aishwarya Agrawal et al. *VQA: Visual Question Answering*. 2016. arXiv: 1505.00468 [cs.CL].
- [3] Felipe Almeida and Geraldo Xexéo. “Word Embeddings: A Survey”. In: *CoRR* abs/1901.09069 (2019). arXiv: 1901.09069. URL: <http://arxiv.org/abs/1901.09069>.
- [4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].
- [5] Youngmin Baek et al. *Character Region Awareness for Text Detection*. 2019. DOI: 10.48550/ARXIV.1904.01941. URL: <https://arxiv.org/abs/1904.01941>.
- [6] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: <https://www.wandb.com/>.
- [7] Thomas M. Breuel. “The OCRopus open source OCR system”. In: *Electronic imaging*. 2008.
- [8] Zhaowei Cai and Nuno Vasconcelos. *Cascade R-CNN: Delving into High Quality Object Detection*. 2017. arXiv: 1712.00726 [cs.CV].
- [9] Wikimedia Commons. *File:Long Short-Term Memory.svg — Wikimedia Commons, the free media repository*. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:Long_Short-Term_Memory.svg&oldid=488381453.
- [10] CVAT.ai Corporation. *Computer Vision Annotation Tool (CVAT)*. Version 2.2.0. Sept. 2022. URL: <https://github.com/opencv/cvat>.
- [11] Tyler Dauphinee, Nikunj Patel, and Mohammad Rashidi. *Modular Multimodal Architecture for Document Classification*. 2019. DOI: 10.48550/ARXIV.1912.04376. URL: <https://arxiv.org/abs/1912.04376>.
- [12] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: <https://arxiv.org/abs/1810.04805>.

- [13] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: <https://arxiv.org/abs/2010.11929>.
- [14] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (Mar. 1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1. URL: https://doi.org/10.1207/s15516709cog1402_1.
- [15] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. URL: <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.
- [16] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: <https://github.com/Lightning-AI/lightning>.
- [17] Javier Ferrando et al. “Improving Accuracy and Speeding Up Document Image Classification Through Parallel Systems”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pp. 387–400. DOI: 10.1007/978-3-030-50417-5_29. URL: https://doi.org/10.1007%2F978-3-030-50417-5_29.
- [18] Ignazio Gallo et al. “Image and Text fusion for UPMC Food-101 using BERT and CNNs”. In: *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. 2020, pp. 1–6. DOI: 10.1109/IVCNZ51579.2020.9290622.
- [19] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. “A survey on instance segmentation: state of the art”. In: *International Journal of Multimedia Information Retrieval* 9.3 (July 2020), pp. 171–189. DOI: 10.1007/s13735-020-00195-x. URL: <https://doi.org/10.1007%2Fs13735-020-00195-x>.
- [20] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. “Evaluation of Deep Convolutional Nets for Document Image Classification and Retrieval”. In: *International Conference on Document Analysis and Recognition (ICDAR)*.
- [21] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].
- [22] Kaiming He et al. *Mask R-CNN*. 2017. DOI: 10.48550/ARXIV.1703.06870. URL: <https://arxiv.org/abs/1703.06870>.
- [23] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL].

- [24] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.
- [25] Mohammad Hossin and Sulaiman M.N. “A Review on Evaluation Metrics for Data Classification Evaluations”. In: *International Journal of Data Mining & Knowledge Management Process* 5 (Mar. 2015), pp. 01–11. DOI: 10.5121/ijdkp.2015.5201.
- [26] Wei-Ning Hsu et al. *HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units*. 2021. arXiv: 2106.07447 [cs.CL].
- [27] Yupan Huang et al. *LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking*. 2022. DOI: 10.48550/ARXIV.2204.08387. URL: <https://arxiv.org/abs/2204.08387>.
- [28] Zhiheng Huang, Wei Xu, and Kai Yu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. 2015. arXiv: 1508.01991 [cs.CL].
- [29] Ian Soboroff. *Complex Document Information Processing (CDIP) dataset*. en. 1996. DOI: 10.18434/MDS2-2531. URL: <https://data.nist.gov/od/id/mds2-2531>.
- [30] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].
- [31] JaiedAI. *EasyOCR: Ready-to-use OCR with 80+ supported languages and all popular writing scripts*. <https://github.com/JaiedAI/EasyOCR>. 2021.
- [32] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. “FUNSD: A Dataset for Form Understanding in Noisy Scanned Documents”. In: *Accepted to ICDAR-OST*. 2019.
- [33] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *YOLO by Ultralytics*. Version 8.0.0. Jan. 2023. URL: <https://github.com/ultralytics/ultralytics>.
- [34] Benjamin Kiessling. *The Kraken OCR system*. Version 4.1.2. Apr. 2022. URL: <https://kraken.re>.
- [35] Alexander Kirillov et al. *Panoptic Segmentation*. 2019. arXiv: 1801.00868 [cs.CV].

- [36] Alexander LeNail. “NN-SVG: Publication-Ready Neural Network Architecture Schematics”. In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: 10.21105/joss.00747. URL: <https://doi.org/10.21105/joss.00747>.
- [37] Quentin Lhoest et al. “Datasets: A Community Library for Natural Language Processing”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. arXiv: 2109.02846 [cs.CL]. URL: <https://aclanthology.org/2021.emnlp-demo.21>.
- [38] Chenliang Li et al. “StructuralLM: Structural Pre-training for Form Understanding”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 6309–6318. DOI: 10.18653/v1/2021.acl-long.493. URL: <https://aclanthology.org/2021.acl-long.493>.
- [39] Minghui Liao et al. *Real-time Scene Text Detection with Differentiable Binarization*. 2019. DOI: 10.48550/ARXIV.1911.08947. URL: <https://arxiv.org/abs/1911.08947>.
- [40] Tianyang Lin et al. *A Survey of Transformers*. 2021. arXiv: 2106.04554 [cs.LG].
- [41] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
- [42] Xiaojing Liu et al. “Graph Convolution for Multimodal Information Extraction from Visually Rich Documents”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 32–39. DOI: 10.18653/v1/N19-2005. URL: <https://aclanthology.org/N19-2005>.
- [43] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].
- [44] Ze Liu et al. *Swin Transformer V2: Scaling Up Capacity and Resolution*. 2022. arXiv: 2111.09883 [cs.CV].

- [45] Ilya Loshchilov and Frank Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: <http://arxiv.org/abs/1711.05101>.
- [46] J. Martínek et al. “Hybrid Training Data for Historical Text OCR”. In: *15th International Conference on Document Analysis and Recognition (ICDAR 2019)*. Sydney, Australia, Sept. 2019, pp. 565–570. ISBN: 978-1-7281-2861-0. DOI: 10.1109/ICDAR.2019.00096.
- [47] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].
- [48] Mindee. *docTR: Document Text Recognition*. <https://github.com/mindee/doctr>. 2021.
- [49] Daniel Neimark et al. *Video Transformer Network*. 2021. DOI: 10.48550/ARXIV.2102.00719. URL: <https://arxiv.org/abs/2102.00719>.
- [50] Clemens Neudecker et al. “A Survey of OCR Evaluation Tools and Metrics”. In: *The 6th International Workshop on Historical Document Imaging and Processing*. HIP ’21. Lausanne, Switzerland: Association for Computing Machinery, 2021, 13–18. ISBN: 9781450386906. DOI: 10.1145/3476887.3476888. URL: <https://doi.org/10.1145/3476887.3476888>.
- [51] OCR-D. *OCR-D*. <https://ocr-d.de/>. 2023.
- [52] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].
- [53] PaddlePaddle. *PaddleOCR*. <https://github.com/PaddlePaddle/PaddleOCR>. 2021.
- [54] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. “A Survey on Performance Metrics for Object-Detection Algorithms”. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.
- [55] Jongchan Park et al. “Fraud Detection with Multi-Modal Attention and Correspondence Learning”. In: *2019 International Conference on Electronics, Information, and Communication (ICEIC)*. 2019, pp. 1–7. DOI: 10.23919/ELINFOCOM.2019.8706354.
- [56] Seunghyun Park et al. “CORD: A Consolidated Receipt Dataset for Post-OCR Parsing”. In: (2019).
- [57] Qiming Peng et al. *ERNIE-Layout: Layout Knowledge Enhanced Pre-training for Visually-rich Document Understanding*. 2022. arXiv: 2210.06155 [cs.CL].

- [58] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: 2018.
- [59] Benjamin Recht et al. *Do ImageNet Classifiers Generalize to ImageNet?* 2019. arXiv: 1902.10811 [cs.CV].
- [60] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: <https://arxiv.org/abs/1506.01497>.
- [61] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.
- [62] Baoguang Shi, Xiang Bai, and Cong Yao. *An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition*. 2015. DOI: 10.48550/ARXIV.1507.05717. URL: <https://arxiv.org/abs/1507.05717>.
- [63] Noah Shinn, Beck Labash, and Ashwin Gopinath. *Reflexion: an autonomous agent with dynamic memory and self-reflection*. 2023. arXiv: 2303.11366 [cs.AI].
- [64] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: <https://arxiv.org/abs/1409.1556>.
- [65] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG].
- [66] Ray Smith. *tesseract-ocr/tesseract*. <https://github.com/tesseract-ocr/tesseract>. 2021.
- [67] Uwe Springmann et al. *Ground Truth for training OCR engines on historical documents in German Fraktur and Early Modern Latin*. 2018. arXiv: 1809.05501 [cs.CL].
- [68] Krishna Srinivasan et al. “WIT: Wikipedia-based Image Text Dataset for Multimodal Multilingual Machine Learning”. In: *arXiv preprint arXiv:2103.01913* (2021).
- [69] Yu Sun et al. *ERNIE: Enhanced Representation through Knowledge Integration*. 2019. arXiv: 1904.09223 [cs.CL].
- [70] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. DOI: 10.48550/ARXIV.1512.00567. URL: <https://arxiv.org/abs/1512.00567>.

- [71] Jasper Uijlings et al. “Selective Search for Object Recognition”. In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.
- [72] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].
- [73] Junke Wang et al. *M2TR: Multi-modal Multi-scale Transformers for Deepfake Detection*. 2022. arXiv: 2104.09770 [cs.CV].
- [74] Christoph Wick, Christian Reul, and Frank Puppe. “Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition”. In: *Digital Humanities Quarterly* 14.1 (2020).
- [75] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- [76] Yuxin Wu et al. *Detectron2*. <https://github.com/facebookresearch/detectron2>. 2019.
- [77] Yiheng Xu et al. “LayoutLM: Pre-Training of Text and Layout for Document Image Understanding”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, 1192–1200. ISBN: 9781450379984. DOI: 10.1145/3394486.3403172. URL: <https://doi.org/10.1145/3394486.3403172>.
- [78] Xiao Yang et al. *Learning to Extract Semantic Structure from Documents Using Multimodal Fully Convolutional Neural Network*. 2017. DOI: 10.48550/ARXIV.1706.02337. URL: <https://arxiv.org/abs/1706.02337>.
- [79] Aston Zhang et al. “Dive into Deep Learning”. In: *CoRR* abs/2106.11342 (2021). arXiv: 2106.11342. URL: <https://arxiv.org/abs/2106.11342>.
- [80] Peng Zhang et al. *VSR: A Unified Framework for Document Layout Analysis combining Vision, Semantics and Relations*. 2021. DOI: 10.48550/ARXIV.2105.06220. URL: <https://arxiv.org/abs/2105.06220>.

- [81] Xiang Zhang. *Understanding mask R-CNN*. Nov. 2021. URL: https://www.shuffleai.blog/blog/Understanding_Mask_R-CNN_Basic_Architecture.html.
- [82] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. “PubLayNet: largest dataset ever for document layout analysis”. In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. Sept. 2019, pp. 1015–1022. DOI: 10.1109/ICDAR.2019.00166.