



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

SEGMENTATION OF LOGICAL UNITS IN TEXT

DĚLENÍ TEXTU DO LOGICKÝCH CELKŮ

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. MARTIN KOSTELNÍK

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. KAREL BENEŠ

BRNO 2024

Master's Thesis Assignment



155569

Institut: Department of Computer Graphics and Multimedia (DCGM)
Student: **Kostelník Martin, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: Machine Learning
Title: **Segmentation of logical units in text**
Category: Speech and Natural Language Processing
Academic year: 2023/24

Assignment:

Literature:

- Glavaš, Goran, and Swapna Somasundaran. "Two-Level Transformer and Auxiliary Coherence Modeling for Improved Text Segmentation." AAI 2020 NYC (2020).

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Beneš Karel, Ing.**
Head of Department: Černocký Jan, prof. Dr. Ing.
Beginning of work: 1.11.2023
Submission deadline: 17.5.2024
Approval date: 9.11.2023

Abstract

The goal of this project is the topic segmentation of text into coherent units. It builds on the PERO-OCR software, aiming to improve the processing of Czech historical documents and information retrieval for librarians and scientists. This included the creation and annotation of a custom dataset comprised of 4044 pages from books, dictionaries, and periodicals. I propose an innovative approach treating segmentation as a line clustering problem. The method involves a two-stage process: initial detection of regions of interest containing text lines using the YOLOv8 model, followed by joining them using a graph neural network. This method achieves a V-measure of 77.93 %, 95.79 % and 90.23 % for books, dictionaries and periodicals, respectively.

Abstrakt

Cílem projektu bylo vytvořit systém pro automatickou segmentaci textu do logických celků. Práce staví na systému PERO-OCR a cílí na zlepšení zpracovávání českých historických dokumentů a jejich vyhledávačů používaných knihovníky a vědci. Práce zahrnovala vytvoření a anotace vlastní datové sady složené celkem z 4044 stránek z knih, slovníků a novin. K problému segmentaci textu je přistoupeno inovativním přístupem, kdy je brán jako shlukovací problém jednotlivých řádků textu. Metoda je dvoufázová: nejprve probíhá detekce regionů textu pomocí modelu YOLOv8 a následuje jejich spojení grafovou neuronovou sítí. Vyhodnocení je provedeno pomocí shlukovací metriky V-measure a na testovacím datasetu dosahuje hodnot 77.93 % pro knihy, 95.79 % pro slovníky a 90.23 % pro noviny.

Keywords

text segmentation, machine learning, optical character recognition, OCR, language models, graph neural networks, object detection, BERT, YOLOv8, historical documents

Klíčová slova

segmentace textu, strojové učení, optické rozpoznávání znaků, OCR, jazykové modely, grafové neuronové sítě, detekce objektů, BERT, YOLOv8, historické dokumenty

Reference

KOSTELNÍK, Martin. *Segmentation of logical units in text*. Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Karel Beneš

Rozšířený abstrakt

Tato práce se zaměřuje na tematickou segmentaci historických dokumentů do logických kusů, jako jsou články, slovníkové záznamy nebo novinky v novinách. Tematická segmentace pomáhá knihovníkům a vědcům, kteří mají k dispozici velké množství nekategorizovaných dat. Práce je také součástí projektu semANT na FIT VUT, který si ve spolupráci s Ministerstvem kultury České republiky klade za cíl zlepšit schopnosti vyhledávání v digitalizovaných dokumentech začleněním sémantického porozumění.

Projekt staví na softwaru PERO-OCR, který poskytuje detekci řádků textu a jejich přepis, nicméně nezaručuje správné pořadí detekovaných řádků. Proto definuji problém segmentace textu jako problém shlukování řádků. Každý shluk může představovat buď titulek nebo textový segment. Navržené metody jsou vyhodnocovány pomocí shlukovacích metrik: úplnost, homogenita a V-measure.

Vzhledem k tomu, že nebyly nalezeny žádné veřejné datové sady pro tematickou segmentaci v českém jazyce, bylo nutné vytvořit vlastní. V potaz jsou brány tři různé typy historických dokumentů: knihy, slovníky a periodika. Celkem bylo vybráno 4044 stránek z historických dokumentů z digitální knihovny k zpracování. S pomocí několika knihovníků a dalších studentů byly tyto stránky anotovány. Nejtěžší a nejrozsáhlejší stránky byly poté vybrány jako validační a testovací datové sady. Validací i testovací sada každá obsahuje 90 stránek.

Nejprve bylo implementováno základní řešení problému bez použití technik strojového učení, které je založeno pouze na geometrii stránky. Tato metoda nalezne předchůdce a následníka každého textového řádku pomocí geometrických omezení. Tyto vztahy nemusí být symetrické a v místě, kde je symetrie porušena, je text rozdělen, což má za následek vznik menších segmentů.

Byla vytvořena dvě vylepšení základní metody. Segmenty generované základní metodou jsou dále rozděleny do menších segmentů zavedením heuristické funkce. První heuristika je čistě geometrická. Rozdělení je provedeno, když vertikální vzdálenost dvou po sobě jdoucích řádků je větší než průměr pro celou stránku, který je navíc upravený parametrem.

Druhá heuristika je reprezentována jazykovým modelem BERT, který je doladěn pro predikci návaznosti dvou textových řádků. Podobně jako u heuristiky, založené na vzdálenosti, je rozdělení provedeno, když model u dvou řádků predikuje, že na sebe řádky nena navazují. Předtrénovaný model CZERT, což je varianta modelu BERT pro český jazyk, byl doladěn na predikci návaznosti pomocí anotovaných dat. Také byly předtrénovány vlastní modely BERT na velkém korpusu knižních dat z digitální knihovny. Čtyři modely s různými velikostmi výstupů byly trénovány ve stejném stylu jako originální model BERT. Jejich ladění proběhlo stejně jako u modelu CZERT.

Základní metoda funguje poměrně dobře na stránkách, kde každý segment je tvořen jedním odstavcem a segmenty jsou vizuálně odděleny. Nicméně takových stránek je minimum. Heuristika založená na vzdálenosti dále tuto schopnost zlepšuje, ale metoda má potíže s hustými a složitými dokumenty, zejména ve slovnících, kde položky nejsou vizuálně odděleny. Varianta s jazykovými modely základní metodu také zlepšuje, ale nejlépe funguje varianta založená na vzdálenosti, která dostahuje hodnot V-measure 49.77 % pro knihy, 64.67 % pro slovníky a 77.40 % pro periodika. Jazykové modely mají další nevýhodu, a to je délka zpracování, která se oproti základní metodě zvýší až desetkrát.

Hlavní navržená metoda pracuje ve dvou krocích: počáteční detekce regionů v obrázku stránky obsahujících textové řádky pomocí vizuálního detektoru YOLOv8, kterou následuje jejich spojení pomocí grafové neuronové sítě. Předtrénované modely YOLOv8 byly doladěny k detekci textových regionů v textových stránkách. Byly provedeny experimenty

s nano, small a medium variantami modelu YOLOv8, a také s různými rozlišeními vstupních obrázků od 640px do 1400px na dlouhé straně se zachováním poměru stran. Po natrénování byl vybrán model na základě přesnosti detekce, přesnosti segmentace pomocí metriky V-measure a rychlosti zpracování. Detekované regiony jsou poté namapovány na textové řádky z PERO-OCR a mohou být považovány za finální segmenty. YOLOv8 dosahuje hodnot V-measure 65.33 % pro knihy, 93.36 % pro slovníky a 83.93 % pro periodika.

Problém je, že YOLOv8 může detekovat pouze obdélníky zarovnané podle os, a proto nejsou správně detekovány segmenty, které jsou ve více sloupcích. Z detekovaných regionů je vytvořen kompletní, neorientovaný graf, který je následně předán grafové neuronové síti. Uzly grafu jsou reprezentovány detekovanými regiony a jsou popsány geometrickými příznaky. Hrany jsou reprezentovány jak geometrickými, tak jazykovými příznaky: euklidovskou vzdáleností dvou regionů a hlavně kosinovou vzdáleností mezi jazykovými příznaky, které jsou vygenerovány jazykovým modelem CZERT.

Grafová neuronová síť pracuje jako klasifikátor hran. Předpovídá, zda by měly být dva uzly (regiony) spojeny dohromady. *ResGatedGraphConv* vrstva je použita jako grafová vrstva, spolu s normalizací a nelineární aktivační vrstvou tvoří stavební blok pro grafovou neuronovou síť. Finální grafová síť je tvořena vstupní projekcí s vrstvou dropout, několika grafových bloků a výstupní projekcí. Spojení regionů detekovaných modelem YOLOv8 dále zvyšuje přesnost segmentace. Metoda dosahuje hodnot V-measure 77.93 % pro knihy, 95.79 % pro slovníky a 90.23 % pro periodika.

Segmentation of logical units in text

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Karel Beneš. Supplementary information was provided by Ing. Michal Hradiš, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Martin Kostelník
May 14, 2024

Acknowledgements

I would like to thank my supervisor Ing. Karel Beneš for his expertise and guidance. Furthermore, I would like to express my thanks to Ing. Michal Hradiš, Ph.D. for his valuable consultations. Lastly, I am deeply grateful to my family and friends for their constant support and encouragement.

Contents

1	Introduction	2
2	Background	3
2.1	Convolutional Neural Networks	3
2.2	Graph Neural Networks	6
2.3	Object Detection	9
2.4	Transformer Based Language Models	13
2.5	Overview of Semantic Text Segmentation Methods	15
3	Problem Definition	17
3.1	Evaluation metrics	17
4	Data	20
4.1	Data Collection	20
4.2	Data Labelling	22
5	Proposed Method and Implementation	24
5.1	Page Geometry	24
5.2	Baseline Solution	25
5.3	YOLOv8 Detection	27
5.4	Proto-Bite Joining	28
5.5	Output File Formats	31
6	Experiments and Results	33
6.1	Baseline Experiments and Results	33
6.2	YOLOv8 Experiments and Results	36
6.3	Graph Neural Network Experiments and Results	39
6.4	Results Overview and Discussion	41
7	Conclusion	43
	Bibliography	44

Chapter 1

Introduction

Over the history of humankind, written documents were the most used way of storing and sharing data. There are billions and billions of all kinds of documents, books, dictionaries, encyclopedia, newspapers, legal documents and many more. The documents are gathered and stored by libraries and other institutions. In the information age, a great emphasis is placed on the digitization of these documents with the aim of preserving the heritage of society, but due to the sheer quantity of data, this process is extremely slow. The use of machine learning can massively increase the processing rate, allowing for a faster expansion of digital libraries.

One step in this digitization process, which is the main goal of this project, is the segmentation of the documents into logical units like book chapters, dictionary entries or newspaper articles. The segmentation can serve as a basis for many applications like search engines, where it could aid finding specific text pages based on the article topic or keywords or content recommendation systems in platforms focused on historical materials.

Segmentation of documents is usually done directly on text data, but the structure of individual pages can provide additional information and is often omitted. The challenge here is the combination of these two different kinds of information to form a robust segmentation system. The proposition is to use an object detector to isolate regions containing text and subsequent joining of these regions using a graph neural network that would utilize both structural and text information.

The work is a part of the semANT project developed at BUT FIT in a collaboration with the Ministry of Culture Czech Republic, which aims to improve the search capabilities in digitized documents and the navigation between thematically similar documents.

The text begins with the theoretical background necessary for the understanding of this project in Chapter 2. Chapter 3 defines the problem this project aims to solve and the evaluation metrics. Chapter 4 is all about the data used, its origin, acquisition and labeling. Chapter 5 presents the main methods, their ideas and implementation. Finally, Chapter 6 describes the experiments with the models and the final results.

Chapter 2

Background

This chapter provides an overview of key techniques and concepts crucial for the understanding of this project. It begins with an introduction to two distinct classes of neural networks. First, it goes over convolutional neural networks, their fundamentals and established architectures. Then it takes a look at graph neural network, their basics and illustrating several graph convolution layers. Subsequently, it delves into object detection highlighting several traditional and deep learning approaches. Lastly, it explores transformer-based language models and their use cases within the project’s scope.

2.1 Convolutional Neural Networks

Convolutional neural network (CNNs) [20] are a class of deep learning models mainly used in image and video processing. They have found success because of their ability to capture spatial features and patterns using layers stacked in a hierarchical architecture. These layers perform an operation called convolution, extracting features at different levels of abstraction.

In the context of text segmentation, CNNs are often used as the backbone of OCR (Optical Character Recognition) systems, which rely on CNNs to detect and recognize text regions within images and subsequently perform character recognition and text extraction.

This project utilizes CNNs as an image processing tool. Apart from being the backbone of OCR, they are also used as the backbone to visual detector models like YOLO described later in Section 2.3. For this reason, information provided in this section will assume that image data is used. It is worth noting that convolution as an operation can be applied in spaces with any dimensionality, such as 1D for speech, 3D for video, or others.

2.1.1 Components of Convolutional Neural Networks

Convolutional neural network are composed of several key components, each serving their specific purpose. These are convolutional layers, pooling layers and fully connected layers. As it is common with other neural networks, activation, normalization and dropout layers are often present as well.

Convolutional layers are the core building blocks of CNNs. They perform the convolution operation on the input data using *kernels* with trainable weights. These kernels, also called filters, are small matrices that detect particular features present in the data. They operate by sliding over the input sample, computing element-wise multiplication of

the kernels with a sub-region of the input. The result of applying the kernel across the sample is called a feature map.

The convolution operation is denoted by the star symbol $*$ and for 1D data it is mathematically defined as:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m],$$

where f and g are two discrete functions and n is the time index of the output signal.

In machine learning libraries like PyTorch¹ or TensorFlow², the convolutional layers are implemented to use cross-correlation instead of convolution. The only difference is that, in the case of convolution, the kernel is flipped along all spatial dimensions. The operations become identical when the kernel is symmetrical. The calculation of the values in the feature map is represented as:

$$y_{i,j} = \phi \left(\sum_{l=0}^F \sum_{m=0}^F w_{l,m} \cdot x_{i+l,j+m} + b \right),$$

where $y_{i,j}$ is the output value at position (i, j) , $w_{l,m}$ is the weight of the kernel at position (l, m) , $x_{i+l,j+m}$ is the value in the input sample, b is the bias and ϕ is the activation function. An example of the element-wise multiplication in the convolution calculation can be seen in Figure 2.1.

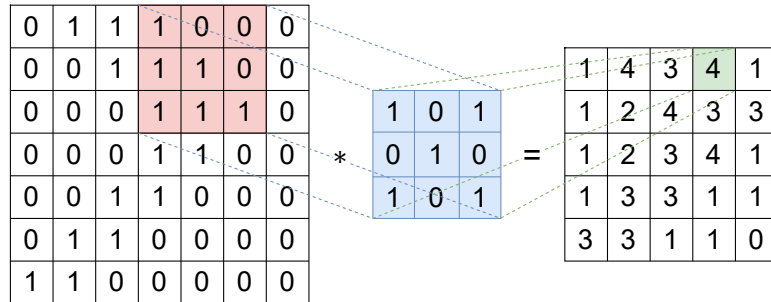


Figure 2.1: Element-wise multiplication in the convolution computation using a 3×3 kernel. Image taken from [34] and recreated.

Pooling layers are used to reduce the spatial dimensions of feature maps. They are typically found after convolutional layers and they operate on each feature map independently. The most common pooling layer variations are maximum pooling and average pooling. Both of these variants can be seen in Figure 2.2. Overall, this technique helps to reduce the computational complexity and also improve the model performance by making it more robust to small translations in the input image.

Fully connected layers are layers that connect every neuron in the previous layer to every neuron in the current layer. They are typically used at the end of the network to produce the desired output in classification tasks.

2.1.2 Established Convolutional Neural Network Architectures

Over the years, many CNNs with varying architectures have been proposed, each with its unique design principles and architectural innovations. This section introduces some of the

¹<https://pytorch.org/>

²<https://www.tensorflow.org/>

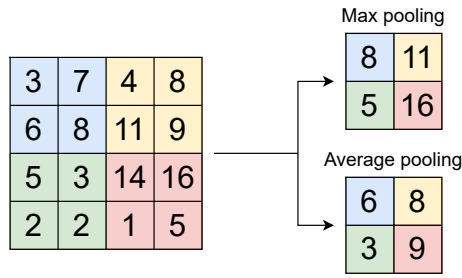


Figure 2.2: Example of max and average pooling layers in convolutional neural networks.

most influential CNN architectures, their features and contributions to the field of deep learning.

LeNet

One of the first CNN architectures, LeNet [22] was developed by Yann LeCun et al. in 1998. It's original purpose was the recognition of handwritten digits. LeNet features two convolution layers followed by three fully connected layers (Figure 2.3). The last fully connected layer uses a softmax activation to output the probability distribution over the digits. The convolution layers use small kernels and pooling layers to extract features from the input images. In total, the network has around 60k trainable parameters.

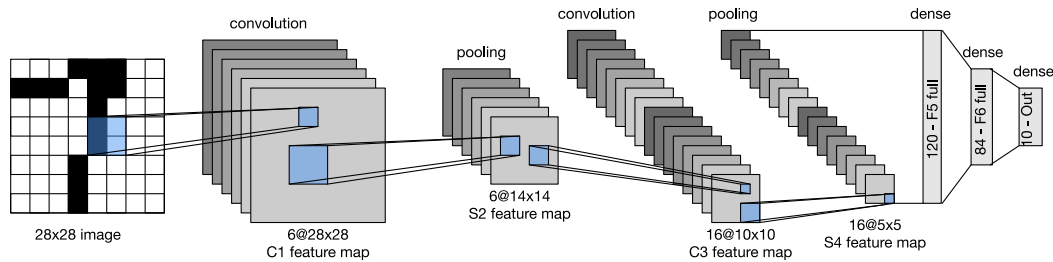


Figure 2.3: Flow of data in LeNet-5 convolutional neural network used for the classification of handwritten digits. Image taken from [27].

AlexNet

AlexNet [21] was developed by Alex Krizhevsky et al. in 2012. At that time, it achieved state-of-the-art performance on the ImageNet dataset [3]. The architecture consists of five convolutional layers and three fully connected layers. Compared to LeNet, it is much bigger in the number of trainable parameters, which are 62M. It also uses local response normalization and overlapping pooling layers.

VGG

The VGG (Visual Geometry Group) architecture [33], proposed by Karen Simoyan and Andrew Zisserman in 2014 consists of convolutional layers with small 3×3 filters followed by max-pooling layers. The key innovation is the use of a deeper architecture. The VGG16 variant, which is one of the most used variants, has 138M trainable parameters and contains sixteen layers, compared to eight in AlexNet.

ResNet

ResNet (Residual Network) [9], introduced by Kaiming He et al. in 2015, addresses the problem with training very deep neural networks by introducing residual connections. These connections allow gradients to flow more directly through the network during training. In a sub-network with a certain number of stacked layers performing a function $F(x)$, where x is the input, the residual connection is represented as:

$$y = F(x) + x.$$

Resnet is composed of multiple residual blocks, where each block contains multiple convolutional layers with skip connections that add the original input to the output of the block. Many variants of this network exist, majority of them are smaller in parameters than previously introduced architectures. For example, ResNet-50 has roughly 24M parameters.

GoogLeNet

GoogLeNet [35], also known as Inception-v1, was introduced by a team of researchers at Google in 2014. They proposed an architecture composed of components nicknamed *Inception*, which allow for parallel execution of multiple convolutional operations with different kernel sizes within the same layer. GoogLeNet consists of several of these modules with occasional max-pooling for downsampling. In addition, it incorporates auxiliary classifiers at intermediate layers during training to alleviate the vanishing gradient problem.

MobileNet

MobileNet [11], introduced by researchers at Google in 2017, is designed specifically for mobile and embedded devices with limited computational resources. The key innovation lies in its depth-wise separable convolutional layers, which significantly reduce the number of parameters and the computational cost of the convolution calculation. In depth-wise separable layers, the convolution is split into two separate operations: depth-wise convolution and point-wise convolution. The depthwise convolution applies a single convolutional filter per input channel, while the pointwise convolution combines the output channels of the depthwise convolution using 1×1 kernels. The depthwise convolution is defined as:

$$\hat{G}_{k,l} = \sum_{i,j} \hat{K}_{i,j} \cdot F_{k+i-1,l+j-1},$$

where \hat{G} is the output feature map, \hat{K} is the depth-wise convolutional kernel and F is the input feature map. Various variants of MobileNet exist, each offering improvements in terms of accuracy, efficiency and model size.

2.2 Graph Neural Networks

Graph structured data allow representation of entities and their relationships. Graph $G = (V, E)$ consists of a set of nodes $V = \{v_1, v_2, \dots, v_n\}$ and a set of edges $E = \{e_1, e_2, \dots, e_m\}$. Nodes contain feature vectors describing the object they represent relevant to the domain. Edges connect the nodes, indicating their relationship. They can be either directed or undirected. The interconnected nature of graphs allows them to model a wide range of real world scenarios where entities and mainly their interactions play a major role. For example,

an analysis of social network, where the graph can represent relationships between people, or text analysis, where nodes represent words, sentences or entire paragraphs.

Graph neural networks (GNNs) [14] are a class of neural networks that take graphs as an input. Their strength comes from their ability to model relationships within the graph structured data. Unlike other methods, GNNs can exploit the explicit dependencies, connections and contextual information encoded in these graphs.

Objectives for a GNN can fall into one of three categories: node-level tasks, edge-level tasks and graph-level tasks. Node-level tasks are concerned about predicting certain properties associated with every node. Relevant use case for this project could be classification of text paragraphs to multiple classes such as title, text or image description.

Edge-level tasks analyze local interactions and dependencies between pairs of nodes in a graph. This is the application used in this project, where edge classification is performed to determine whether two or more paragraphs form one coherent segment. Another use case could be recommendation systems or edge predictions, where the model would predict the existence of new edges, discovering potential connections in social networks for example.

Graph-level tasks cover specific cases, where the goal is to discover characteristics and properties of the entire graph. For example, evaluating new chemical compounds in molecule analysis or more general applications like graph classification, graph clustering, which involves partitioning the graph into meaningful subsets or graph regression, associated with the prediction of continuous-valued attributes associated with the entire graph.

2.2.1 GNN Computation Process

When a graph is fed to a GNN, node feature vectors are iteratively updated. During each iteration, a node gathers information from its neighboring nodes and edges, transforming it using a message passing algorithm. This updated information is then incorporated to the original node representation, either by augmenting it through skip connections or replacing it entirely. More precisely, after the first iteration, each node embedding holds information from neighboring nodes within a distance of $d = 1$. After second iteration, each embedding contains information of nodes within a distance of $d \leq 2$. Generally, after k -th iteration, each node includes data from nodes within a distance of $d \leq k$.

Similarly to other neural networks, GNNs also contain multiple layers, which is illustrated in Figure 2.4. Each layer can contain skip connections, sampling operator, convolutional, recurrent or attention operator and pooling. Some operators are introduced in the next section.

2.2.2 Existing GNN Layers

The main strength of GNNs is the use of the message passing algorithm aggregating feature vectors of neighboring nodes together. This can be done in many ways. Over time, many aggregating mechanisms were introduced. This section gives a brief overview of some of them:

GCNConv

Graph Convolutional Operator (GCNConv) [15] is a convolutional operator first developed for semi-supervised classification in graph neural networks. It uses an layer-wise propagation rule based on approximation of spectral convolution on graphs. It is mathematically defined

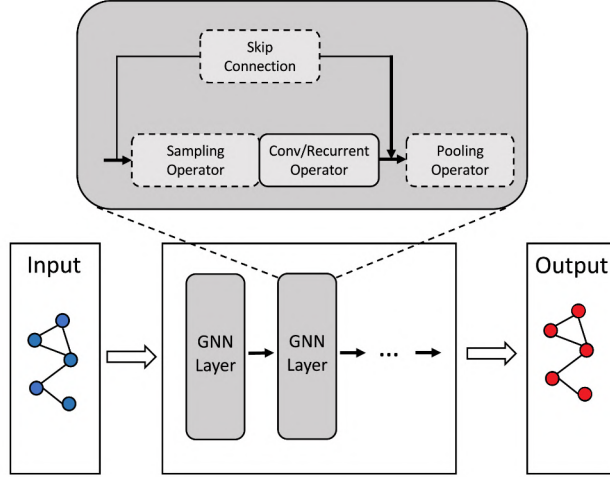


Figure 2.4: Structure of a graph neural network, which takes graphs as an input and propagates node information to its neighbors. Each layer can be composed of multiple components. Output is a new graph with modified node features. Image taken from [40] and edited.

as:

$$Z = \hat{D}^{-\frac{1}{2}} \hat{A} \tilde{D}^{-\frac{1}{2}} \Theta,$$

where $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix, $\Theta \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters, $\hat{A} = A + I$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ its diagonal degree matrix. It can also be defined node-wise as

$$x'_i = \Theta^T \sum_{j \in N(i) \cup \{i\}} \frac{e_{i,j}}{\sqrt{\hat{d}_j \hat{d}_i}} x_j,$$

with $\hat{d}_i = 1 + \sum_{j \in N(i)} e_{j,i}$, where $e_{j,i}$ is the edge weight from node j to node i .

ResGatedGraphConv

Residual gated graph convolutional operator (ResGatedGraphConv) [1] is also a convolutional operator, which introduced gates [25] and residual connections to the computation. In graph neural networks, residuality can play a significant role in improving performance when using multi-layer architectures. Its node wise definition is:

$$x'_i = W_1 x_i + \sum_{j \in N(i)} \eta_{i,j} \odot W_2 x_j,$$

$$\eta_{i,j} = \sigma(W_3 x_i + W_4 x_j),$$

where $\eta_{i,j}$ is the gate and σ denotes the sigmoid function.

GATConv

Graph attentional operator (GATConv) [38] is an operator that uses self-attention instead of convolution. It attempts to overcome the shortcomings of convolution or its approximation

by leveraging self-attentional layers. Unlike traditional convolutional approaches that utilize fixed-weight kernels, GATConv implicitly allows for specifying different weights to different neighboring nodes. Another strength is easy parallelization and the reduction of costly matrix operations like inversion. The node wise definition is:

$$x'_i = \alpha_{i,i}\Theta_s x_i + \sum_{j \in N(i)} \alpha_{i,j}\Theta_t x_j,$$

where the attention coefficients $\alpha_{i,j}$ are calculated as:

$$\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(a_s^\top \Theta_s x_i + a_t^\top \Theta_t x_j))}{\sum_{k \in N(i) \cup \{i\}} \exp(\text{LeakyReLU}(a_s^\top \Theta_s x_i + a_t^\top \Theta_t x_k + a_e^\top \Theta_e e_{i,k}))},$$

where a_s^\top and a_t^\top are the parameters of the attention mechanism³.

2.3 Object Detection

Object detection is a computer vision task, which deals with localizing regions of interest in an image or a video sequence and then classifying them into specific classes. An image can contain an unspecified number of such regions. Use cases range from autonomous vehicles identifying other cars, pedestrians and their surrounding in general, through medical imaging, where object detection techniques can assist in identifying and localizing abnormalities in X-rays or MRIs, to surveillance, satellite imagery analysis or text detection. Examples of object detection can be seen in Figure 2.5.

In the scope of this project, object detectors are employed to analyze scanned images of text pages to identify text paragraphs that hold interesting semantic information. The detector isolates these texts, which allows further analysis and mapping to an output of an OCR system.

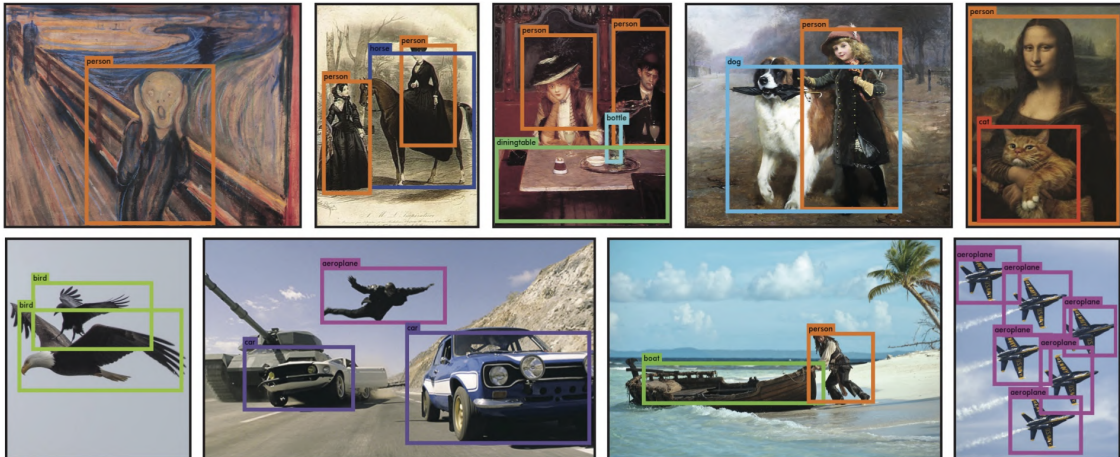


Figure 2.5: Example of object detection. Image taken from [29].

Over the years, numerous object detection techniques have been developed. Among these are traditional algorithms like Haar cascades [39], introduced in 2001, which use

³The attention mechanism is implemented as a single-layer fully connected neural network.

a cascade classifier consisting of multiple stages, each utilizing a set of simple rectangular features called Haar-like features. Another algorithm is Histogram of Oriented Gradients (HOG) [2], which is a feature descriptor technique that operates by computing gradients and their orientations in localized portion of an image. The features are typically fed into a classifier, such as the Support Vector Machine (SVM), to determine whether an object of interest is present within the window.

In recent years, state of the art algorithms increasingly leverage convolutional neural networks (CNNs) and deep learning techniques for object detection. These approaches have demonstrated remarkable performance improvements across various domains.

One significant distinction in these frameworks lies in the design of the detection architecture: one-stage detectors and two-stage detectors. One-stage detectors like YOLO [29] or SSD [24] aim to directly predict bounding boxes and class probabilities in a single pass through the network. This prioritizes speed, which makes them well suited for real time applications. Two-stage detectors, such as R-CNN [6], first propose potential object regions and then refine these proposals with a classification and bounding box regression. Difference is further highlighted in Figure 2.6.

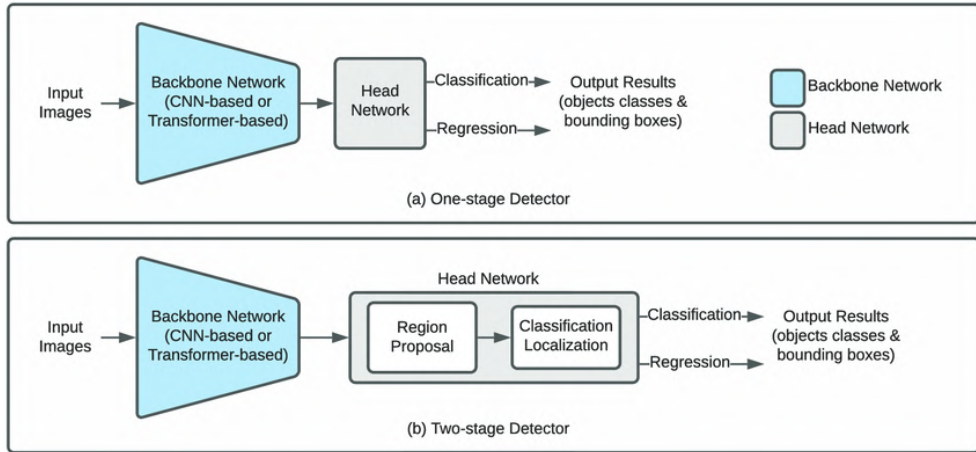


Figure 2.6: Single-stage versus two-stage object detector architecture. The main difference lies in the Head Network part of the model. Taken from [13].

2.3.1 Faster R-CNN

Faster R-CNN (Faster Region-base Convolutional Neural Network) [30] is an object detection model released in 2015. It builds upon the R-CNN [6] family of models and mainly addresses their computational inefficiency. This model was the first to feature a Region Proposal Network (RPN). The architecture of Faster R-CNN consists of a RPN and a subsequent Fast R-CNN [5]. The RPN generates region proposals as bounding boxes with associated objectness score, which indicates the likelihood that the region contains an object. The Fast R-CNN takes the region proposals and performs the object classification and bounding box regression.

The other key innovation are shared convolutional features between the RPN and Fast R-CNN, which reduces the computation time. Both components use a CNN backbone, such as VGG, which is shared between them. Therefore the forward pass requires only a single

pass of the input sample through the backbone. This approach also helps the training process, as it helps the model to learn feature representations effectively for both tasks.

2.3.2 YOLO

YOLO (You Only Look Once) is a single-stage object detector introduced by Joseph Redmon et al. in 2015 [29]. While other models like R-CNN described above use a Region Proposal Network, which feeds into a classifier recognizing these regions, YOLO performs all predictions in one evaluation. Because of this, it can be optimized end-to-end directly for the performance of the detection.

The authors defined the detection as a regression problem. It divides the input image into a $S \times S$ grid and for each cell the model predicts B bounding boxes, their confidence and class probabilities.

Since its introduction in 2015, seven more versions of the YOLO model were released, with YOLOv8 [12] being the latest. Developed by Ultralytics under the AGPL-3.0 license, it does not have a scientific paper released alongside it. Built on YOLOv5 [36], also developed by Ultralytics, it increased the total number of training epochs from 300 to 500. Some changes⁴ to the model architecture were also introduced, like a different convolution module or decoupled heads.

Ultralytics released multiple pre-trained models for more tasks than just object detection like image segmentation or pose estimation. The model also exists in various sizes, ranging from a small model with just 3.2 M parameters to the biggest model with 68.2 M parameters. Comparison can be seen in Figure 2.7.

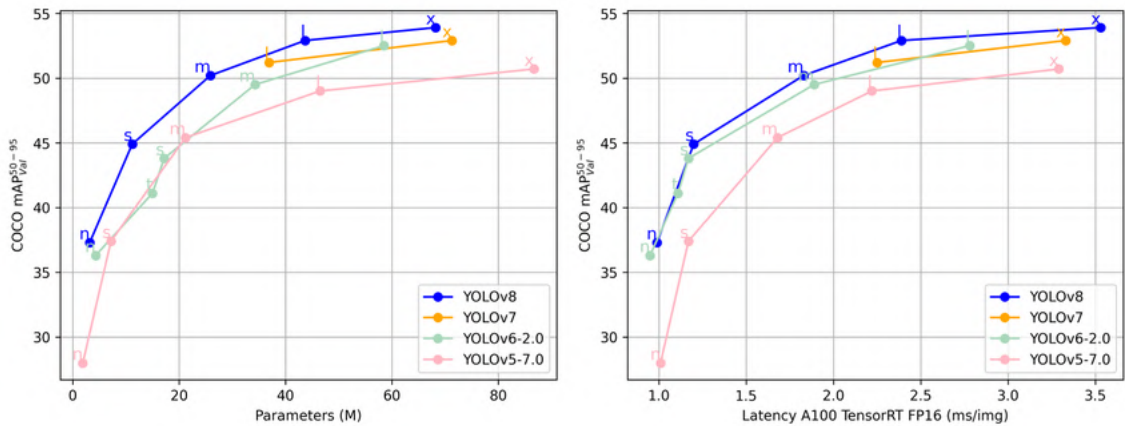


Figure 2.7: Comparison of four versions of YOLO models in all available model sizes. Both graphs showcase the mAP50-95 metric (higher is better) on COCO dataset. The left one has the number of model parameters along the x axis and the right one inference time for one image. Image taken from [23].

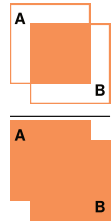
YOLO models have a weakness when trying to detect many objects that are in a close proximity. It is the effect of the non-maximum suppression, which is a technique for filtering region proposals based on the confidence score. It decreases the confidence in overlapping bounding boxes and eliminates bounding boxes with low confidence.

⁴<https://github.com/ultralytics/ultralytics/issues/189>

2.3.3 Object Detection Metrics

This section introduces several object detection metrics that lead to the mean average precision (mAP) metric, which is used to evaluate detectors trained in this project. The information contained in this section comes from [26].

The first metric is the **Intersection over Union (IoU)** metric, which measures the overlap between two bounding boxes – one predicted by the model and the ground truth. It represents the quality of the detection. Calculation involves dividing the bounding boxes overlap by their union, which is illustrated in Equation 2.1. When calculating other metrics at a later stage, IoU is used as a threshold signaling whether a predicted bounding box should be treated as a true positive (TP) or a false positive (FP).

$$\text{IoU} = \frac{A \cap B}{A \cup B} = \frac{\text{Area of intersection}}{\text{Area of union}} = \frac{\text{Diagram 1}}{\text{Diagram 2}} \quad (2.1)$$


Precision and Recall are two metrics that work in contrast to each other. In the scenario where the model predicts every possible bounding box at every pixel of an image, a classifier would correctly find every object in the image, but it would have many wrong predictions. On the other hand, if the model predicts no bounding boxes, it will never have a FP. Precision refers to the ability to detect relevant object only. Recall on the other hand refers to the ability to find all ground-truth objects. They are calculated by the following formulas:

$$\text{Precision} = \frac{\sum_{n=1}^S TP_n}{\sum_{n=1}^S TP_n + \sum_{n=1}^{N-S} FP_n} = \frac{\#TP}{\#\text{all detections}},$$

$$\text{Recall} = \frac{\sum_{n=1}^S TP_n}{\sum_{n=1}^S TP_n + \sum_{n=1}^{G-S} FN_n} = \frac{\#TP}{\#\text{all ground truths}},$$

where G is a set of ground truths, N is a set of detections and $S \subset N$ is a set of correct detections. As one of the outputs of an object detector is a confidence level, the precision-recall calculation can be slightly modified. If the confidence level is lower than a given threshold τ , the bounding box is considered as a negative.

Consequently, he precision and recall metrics can be plotted in a precision-recall curve⁵, which shows the performance of the model at all thresholds. **Average precision** is a metric that comes from evaluating the precision-recall curve. It is the area under this curve and it ranges from 0 to 1, where $AP = 1$ means perfect precision at all confidence thresholds.

Mean average precision (mAP) is used in multi-class object detectors. To compute the mAP, the average of AP values for all classes is obtained:

$$mAP = \frac{1}{C} \sum_{i=1}^C AP_i,$$

⁵The precision-recall curve is also called ROC curve (receiver operating characteristic curve).

where AP_i is the AP value for class i and C is the total number of classes. As described above, precision can change depending on the IoU threshold selected. This in turn also changes the mAP value. In this project, models are evaluated using the following:

1. mAP50 – IoU threshold is fixed at 50%.
2. mAP50-95 – IoU threshold ranges from 50% to 95% with a step of 5%. These ten values are then averaged to get the final metric.

2.4 Transformer Based Language Models

Language modeling is a NLP task of learning the probability distribution of natural language. In recent years, it has seen a major growth, mainly due to the popularization of transformer based language models. They were originally introduced by Vaswani et al. in 2017 [37].

In earlier language models, recurrence and convolution was used to process text sequences. Recurrent neural networks sequentially processed inputs and maintained the hidden states over time, capturing temporal dependencies, while CNNs used sliding windows of one dimensional filters to extract local features of text. Transformers eliminates these techniques entirely and are based solely on the attention mechanism, allowing to capture global dependencies.

The computation of attention and subsequently the forward pass through the transformer are parallelizable, which significantly improves training and inference time, making the models scalable. This allows processing of sequences, such as those encountered in documents or books. The attention is computed in multi-head attention modules, whose architecture can be seen in Figure 2.8.

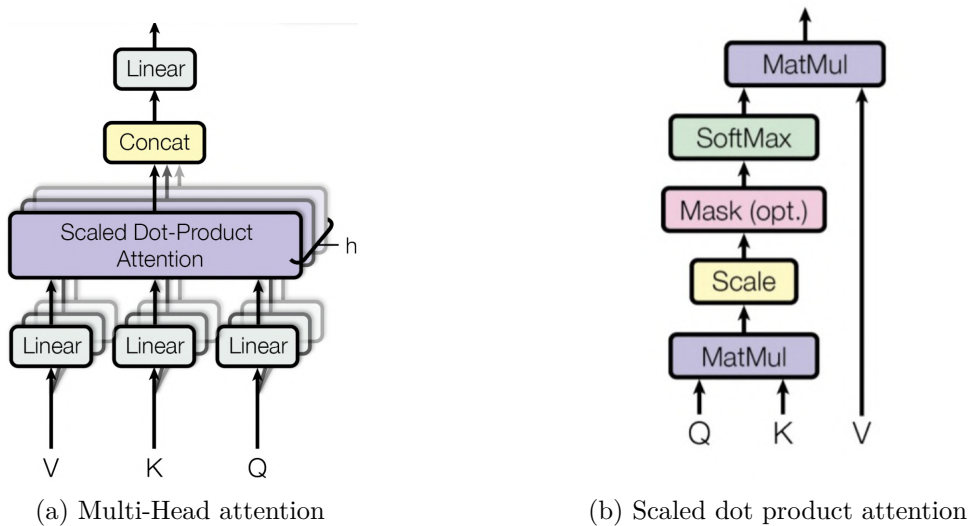


Figure 2.8: Multi-Head attention consists of several parallel attention layers. Scaled dot product attention is used to calculate attention values in the transformer model. Images from [37].

Transformers adapt an encoder-decoder architecture, originally proposed to handle sequence-to-sequence tasks like machine translation. The architecture in detail can be

seen in Figure 2.9. The encoder processes the input sequence to generate a contextual representation (embedding), while the decoder takes this encoding to generate the output sequence.

The encoder is composed of a stack of identical layers, where each layer has two sub-layers: the multi-head attention and a feed forward layer. Residual connections and layer normalization are used to improve stability during training. The decoder is composed of the same blocks, but it incorporates an additional multi-head attention mechanism that operates on the encoder output. The first multi-head attention is also masked, preventing it from attending to subsequent positions. This ensures that the predictions for position i depend only on the outputs at positions less than i .

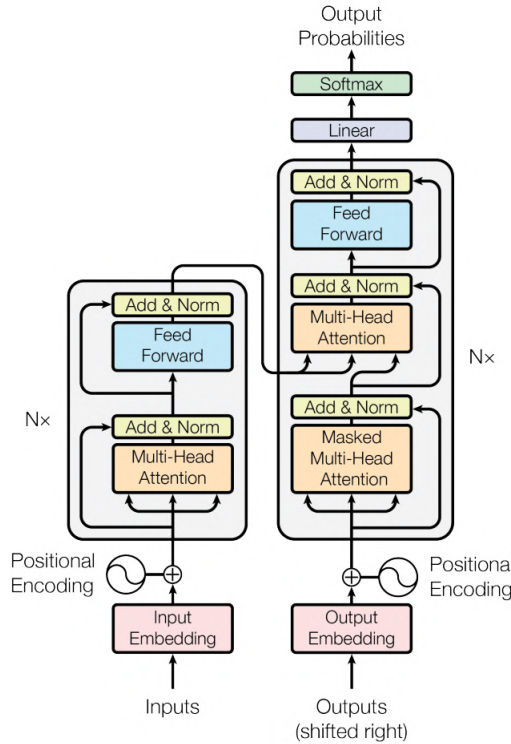


Figure 2.9: Architecture of the transformer model, consisting of an encoder part (left) and a decoder part (right). Image taken from [37].

2.4.1 BERT

Bidirectional Encoder Representation from Transformers (BERT) [4] is a language model based on the transformer architecture. It is an encoder only model that introduces bidirectional context modeling. Unlike models that process text in left to right manner, the bidirectional approach allows to consider context from both preceding and following words simultaneously. The core idea behind BERT model is using a large pretrained model that is then fine-tuned for specific tasks.

Pretraining is done with two objectives: masked language modeling (MLM) and next sentence prediction (NSP). The inputs are pairs of sequences delimited with special tokens. MLM teaches the model to understand contextualized representations of words. In the input sequence, random tokens are masked and the model is tasked with predicting the

original token. In the original implementation, tokens are chosen for masking with a 15% probability. When a token is chosen, it has 80% chance to be replaced with a special [MASK] token, 10% chance to be replaced with a random token and 10% of the time it is left unchanged. NSP is used in addition to MLM. Here, the model tries to predict whether the second sentence follows the first sentence in the original text. This teaches the model to understand the relationships between sentences.

Because the pretraining is a time-consuming process, fine-tuning for specific downstream tasks using task specific objective function is often utilized. Many NLP problems can be solved using BERT in this manner. These include text classification, named entity recognition, question answering, sentiment analysis and others. During fine-tuning, task-specific output layers are added to the pretrained BERT model, allowing it to adapt to specific requirements of the target task. Fine-tuning BERT-based models on task-specific datasets enables them to effectively transfer the knowledge learned during pretraining to new tasks.

2.4.2 CZERT

CZERT [32], developed by a group of researchers in Plzeň in 2021 is a monolingual BERT-like model for the Czech language. It was pretrained on 340K sentences, which is more than usually utilized in multilingual models for a single language, allowing better understanding.

The model was pretrained in the same way as the original BERT, only with slightly modified NSP task. The difference lies in the sampling of negative pairs of sentences. Sentences from the same paragraph that do not follow each other were also used, compared to the original BERT, which uses random sentences only.

2.5 Overview of Semantic Text Segmentation Methods

Semantic text segmentation, or topic segmentation is a technique in natural language processing (NLP) used to divide texts into multi-paragraph units. Essentially, its function revolves around identifying major subtopic shifts indicating a break in the text flow. Application can be found in various domains like information retrieval or content summarization. This section gives an overview about different text segmentation approaches that have surfaced over the years.

2.5.1 TextTiling

TextTiling was introduced by Marti A. Hearst in 1997 [10]. It is an unsupervised method for text segmentation that utilizes identification of lexical co-occurrence and distribution patterns. It draws the inspiration from the metaphor of tiling a floor with interlocking tiles, where each tile represents a distinct text segment.

The algorithm operates in three main parts: tokenization, lexical score determination and boundary identification. It assumes that a set of lexical items is used during a given subtopic discussion. When the subtopic changes, significant portion of the lexical items changes as well. Items that appear often and through the entire text are indicative of the main topic in the text. Then there are less frequent items but with a uniform distribution, these tend to be neutral and do not give information about the subtopic change. The remaining items are of upmost interest. They are grouped together and this group is con-

sidered as representative of a subtopic. Consequently, the segmentation problem becomes determining beginnings and ends of these groups.

The author proposed two solutions to determine the lexical score. In block comparison, adjacent text blocks are compared to evaluate how many words the blocks have in common. Vocabulary introduction method assigns a score to each token-sequence gap. This is based on how many new words were seen.

Boundary identification assigns a depth score to each token-sequence gap. The score corresponds to the cue intensity for subtopic change on both sides. These scores are then sorted and used to determine the boundaries.

2.5.2 GraphSeg

GraphSeg, introduced by Goran Glavaš et al. in 2016 [7] is a graph based algorithm. At its core, it constructs a graph representation of the text, where nodes correspond to sentences and edges exist for pairs of semantically related sentences. The segmentation is then done by finding maximal cliques of the graph.

2.5.3 LSTM-Based Neural Model

Released in 2018 by Omri Koshorek et al. [19], it is a neural network approach formulated as a supervised learning problem. Along with the proposed method, a large and labeled dataset called Wiki-727K was released. The neural model is divided into a hierarchy of two smaller models. Both of these are LSTM-based. The first LSTM generates sentence-level embeddings and the second one takes in a sequence of sentence embeddings and passes them through the second LSTM. A fully connected layer is then applied to each LSTM output, which together with the softmax activation produces a segmentation probability, indicating an end of a segment.

2.5.4 Two-Level Transformer Neural Model

Another method proposed by Goran Glaviš [8] utilizes two transformer based models to perform the segmentation. The model, CATS, was trained in a supervised setting. It combined segmentation prediction with explicit auxiliary coherence modeling. The first transformer encoder is structures similarly as the LSTM-based model introduced earlier. The low-level transformer encodes sentences and generates input for the high-level transformer encoder of sentence sequences.

They trained the model on two tasks. It tried to learn predicting sentence segmentation labels and that the original text snippets are more coherent than corrupt sentence sequences.

2.5.5 Review of Existing Methods

Overall, the problem of text segmentation in entire pages of text is not very well researched. All methods treat text as a continuous stream of data with a well defined reading order, neglecting any information about the layout, structure and the source of the text itself. This information could greatly aid the segmentation process, but combining the different types of data is a challenging task.

Chapter 3

Problem Definition

Text segmentation is a problem in natural language processing (NLP). It is a process of dividing a text page, or any block of text, into one or more segments of meaningful and coherent units. These units can be of syntactic nature, such as individual words, sentences or paragraphs or semantic nature, such as topics. This project aims to segment text pages into logical units, namely articles, dictionary entries or book chapters. The resulting segments are further going to be referred to as *bites*. An example of segmented page can be seen in Figure 3.1.

All input data are in the form of images, and as the project builds upon PERO-OCR¹ software, access to text transcriptions and also to the layout of the page in the PAGE-XML [28] format is available. PERO-OCR works in two stages: first, using its layout engine [17], which is a CNN-based model, it finds text baselines, text line polygons, and text blocks. The text blocks are obtained by clustering text lines based on local text block boundary estimation. In the second stage, the transcription engine [16, 18], a CNN-based model with recurrent layers, is used to obtain the text transcriptions.

While PERO-OCR divides the page into text blocks (regions) by itself, these divisions lack accuracy from a semantic standpoint and cannot be considered as bites. The main goal of the project is to improve the detected regions.

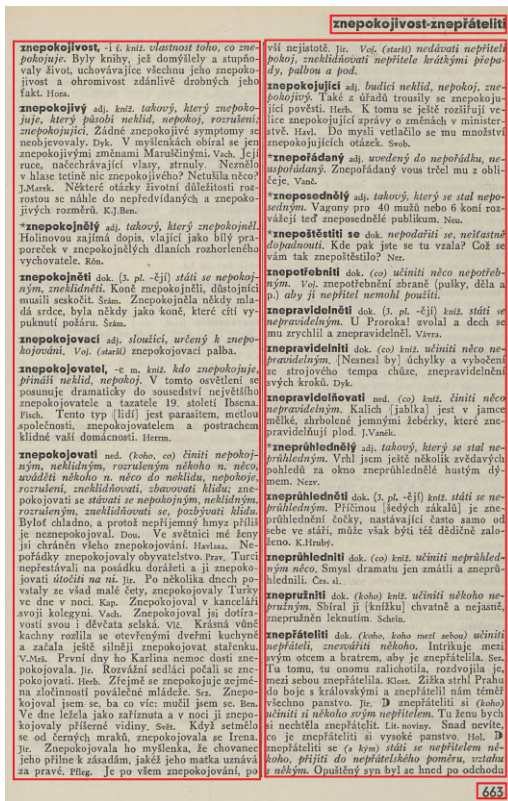
In addition to the primary objective, several secondary objectives are addressed. Not every piece of text on a page necessarily belongs to a bite. For instance a page number provides structural information about the document but lacks semantic value. Therefore such texts have to be excluded. Furthermore, a word or sentence may serve as the title for another bite. Additionally, each bite may possess a name that is a part of one of its paragraphs, such as dictionary keys. Lastly, a bite may consist of visually disconnected smaller text regions, spanning multiple columns or featuring a non-trivial layout.

Since PERO-OCR provides individual lines transcriptions and their enclosing polygon, I define the problem as a clustering problem. Individual lines are grouped into clusters, where each cluster represents a bite. Each bite consists of a list of text lines, a name (dictionary key, ...) and a class, which can be either basic text or a title.

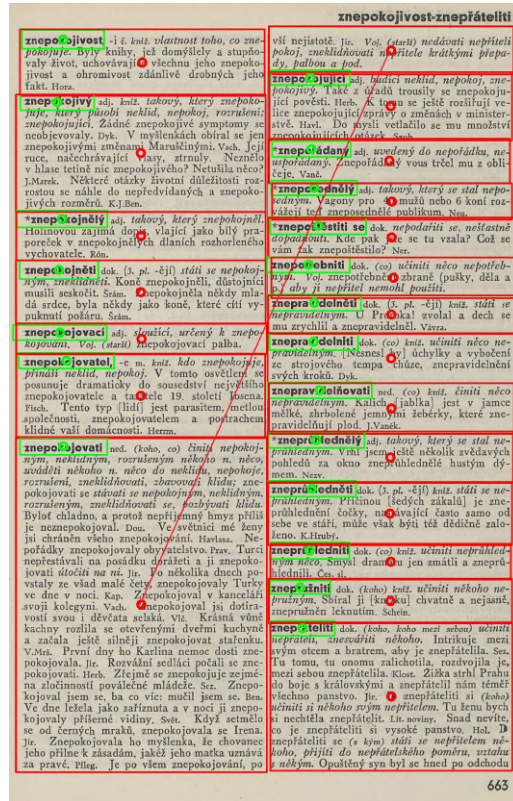
3.1 Evaluation metrics

To evaluate the performance of the method and to compare them against each other, three clustering metrics [31] are used: completeness score, homogeneity score and V-measure

¹Code repository available at <https://github.com/DCGM/pero-ocr>



(a) PERO-OCR



(b) TextBite

Figure 3.1: Example of PERO-OCR detected regions and desired output on a sample page from a dictionary. Each bite is denoted by red bounding box. The dictionary keys are colored green as the names of each bite. The red line connecting two columns showcases that a bite continues on the next column.

score. Completeness ensures that all text lines sharing similar semantic content are grouped together within the same bite. Homogeneity, on the other hand, guarantees that each bite is composed exclusively of text lines belonging to a single coherent segment. Formally, they are given by:

$$\text{Homogeneity} = 1 - \frac{H(C|K)}{H(C)},$$

$$\text{Completeness} = 1 - \frac{H(K|C)}{H(K)},$$

where $H(C|K)$ is the conditional entropy of the classes given the cluster assignments and is given by:

$$H(C|K) = - \sum_{c=1}^{|C|} \sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \cdot \log \left(\frac{n_{c,k}}{n_k} \right),$$

and $H(C)$ is the entropy of the classes and is given by:

$$H(C) = - \sum_{c=1}^{|C|} \frac{n_c}{n} \cdot \log \left(\frac{n_c}{n} \right),$$

where n is the total number of text lines, n_c and n_k is the number of text lines respectively belonging to class c and bite k . $n_{c,k}$ is the number of text lines from class c assigned to bite k . The conditional entropy of clusters given class $H(K|C)$ and the entropy of clusters $H(K)$ are defined in a symmetric manner.

In addition to these two, V-measure is another metric used. It is the harmonic mean of completeness and homogeneity, providing a balanced assessment of clustering quality. It is defined as:

$$V\text{-measure} = \frac{2 \cdot \text{Completeness} \cdot \text{Homogeneity}}{\text{Completeness} + \text{Homogeneity}}.$$

All three metrics are in the range of $[0, 1]$, where the value of 1 means perfect clustering.

Chapter 4

Data

Data collection and processing is a big part of this project. The quality of the data significantly contribute to the performance of a model. These topics are covered by this chapter: the description of the data used, how it was collected and labeled and what steps were taken to ensure the quality of data.

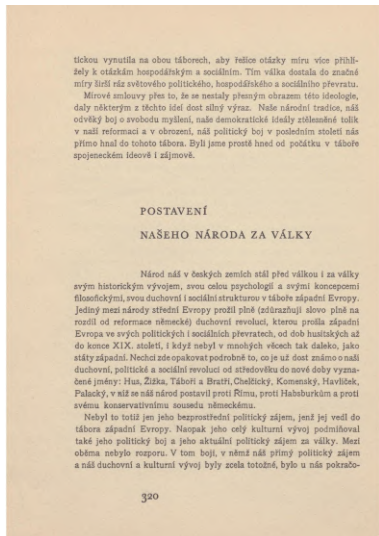
As no public labeled datasets in Czech language were found, a custom dataset had to be created and labeled. Three different types of historic documents are considered for this project: books, dictionaries and periodicals. Example pages of each document type are shown in Figure 4.1. Generally speaking, books have the simplest structure, as they mostly contain a single column of text, which corresponds to a single bite. To counter this, the most interesting and challenging pages were selected. These are book pages that contain chapter breaks, multiple titles, or images.

Dictionaries are more complex. They can be composed of multiple columns and they contain many bites, as each dictionary entry is a bite on its own. The bites are often densely stacked, which can make any visual separation attempts challenging. Dictionaries have one unique feature, the dictionary keys are an important piece of information to extract, as they can help with information retrieval or creating topic embeddings. The keys are different from standard titles found in books, as they are a part of the bite itself. Locating the keys cannot be done as a segmentation, because that would separate the key from the text, making it nonsensical.

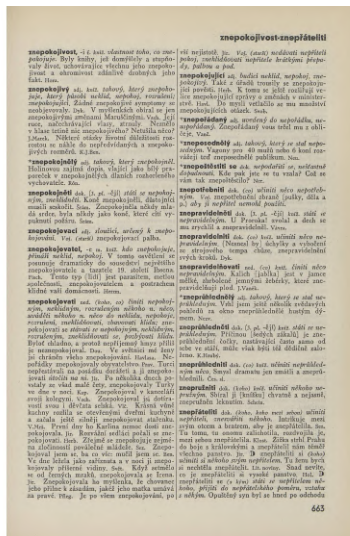
Periodical are the most complex document type. They are composed of many bites that can span multiple columns. Many different newspaper publishers existed in the past, each creating their unique brand with different styles, colors and page layouts. Additionally, the same periodical can have different looks as time progresses. They share the same feature with the dictionaries. Sometimes, in news entries, the news title is a part of the text, but this can vary and is not always the case. Periodicals also contain many levels or hierarchies of titles, this is especially noticeable in title pages. A lot of information that holds no value to topic segmentation is present in the title pages: periodical title, year of publication, edition, among others. This information has to be filtered out of the segmentation evaluation, as there is no interest in extracting page metadata.

4.1 Data Collection

Some Czech libraries or other institutions digitize historic documents and make them available online in digital libraries in the form of raster images. They are made free to access for



(a) Book



(b) Dictionary



(c) Periodical

Figure 4.1: Examples of the three different types of historical documents used in the project: books, dictionaries and periodicals.

educational and research purposes¹. This is the source for all the data used in this project. All pages were handled from hand picked documents available online. Only documents written in the Czech language are considered, but other languages occasionally appear, mainly in dictionaries for translation or periodicals, which, at the of their publication, had small entries written in the German language. In total, 4044 pages from 63 different documents were collected. At this point, the pages are in the form of images. From these pages, several unique, interesting and challenging pages were hand picked for the validation and test datasets. Table 4.1 shows the exact number of pages in all categories. The books are the least represented category, because of the simplicity of their layout, there is very little variance between different documents, reducing the need for more pages. They are followed by dictionaries, where layout sometimes varies between documents. However, not many different dictionaries were found in the digital library. The most represented category is the periodicals, with over 74 % representation. Their diverse content and common updates create a need for a large number of pages. When the dataset was collected, all images were processed using the PERO-OCR to detect polygons enclosing all text lines and to extract their transcription.

Table 4.1: The amount of pages in each document category. The dataset consists of 4044 total pages and we excluded 90 pages as validation and test sets. Each subset has a consistent amount of pages from each category.

	Books	Dictionaries	Periodicals
Train	169	690	3005
Validation	15	30	45
Test	15	30	45

¹<https://www.digitalniknihovna.cz/>

4.2 Data Labelling

To prepare the data for the use in text segmentation, it had to be labeled. Some experiments were performed to see if the training data could be segmented (and partially labeled) in an unsupervised manner. TextTiling (Section 2.5.1) was used, but it did not yield reliable performance. Another unsupervised method is GraphSeg described in Section 2.5.2, but no suitable implementation was found and implementing it from the scratch would most likely not be cost-effective.

The data had to be labeled on the image level by humans. A person would draw axis aligned bounding boxes (AABBs) over the titles and paragraph over the text page. As bites can be composed of visually disconnected pieces, they would also have to connect the AABBs. Labeling a single page took between 30 seconds for simple books to 4 minutes for complex periodicals, making it a very time consuming tasks. With an average labeling time of 3 minutes per page (periodicals have larger representation), it would take roughly 200 hours to label all 4044 pages. To complete the labeling process in a reasonable time, several people from the Moravian Library², along with several BUT FIT students, were asked for assistance. A set of instructions was created to make the labeled data as consistent as possible. Still, small inconsistencies surfaced, although none were a major problem and could be resolved during the data processing stage.

Furthermore, it was decided to not segment any page metadata, like page numbers or publisher name. Even though it could be used for a different project, the added labeling time outweighed the benefits. In title pages of periodicals, the metadata can cover a significant portion of all text present on the page. An example of metadata from a periodical title page can be seen in Figure 4.2.



Figure 4.2: A header from a title page of a periodical showing date of publication, edition, title, description and a quote as a page metadata that should not be labeled.

To label the data, an open source tool called Label Studio³ was used. The task was set up as an object detection with bounding boxes. The bounding boxes were classified with the following labels: *Title*, *Text*, *Note*, *Image* and *Image Description*. Title is either a short, concise description of text, usually located above a text paragraph, or as mentioned earlier, a part of the paragraph itself, as a dictionary key for example. Text label is used on a coherent piece of writing. Note is present very rarely and represents a footnote or author's/editor's note. Labels for images and their descriptions were also added. Detecting

²<https://www.mzk.cz/>

³<https://labelstud.io/>

the notes, images and their description was not included in this project, but can be of use in further projects.

On a page with multiple columns, it is a common occurrence that a bite is split into two or more pieces of text. The Label Studio offers a convenient tool to address this. Annotators can create relations between regions, signaling that more regions form a single bite. Since the relations are directed, they also explicitly provide the reading order. In addition, the relations were used to connect titles to their respective bites. Relations were utilized only when the title was not enclosed in the bite itself, as overlapping regions can be detected automatically. This became a source of labeling inconsistencies, as relations between enclosed bounding boxes were sometimes created by misunderstanding.

Another issue was labeling the relationships between titles. The titles can exhibit hierarchical structure. There can be a main title, like **News**, that would cover a section of multiple news entries, which all have a title of their own. Regrettably, it was not decided in a timely manner how to approach this problem, which led to the biggest labeling inconsistency. An example of the possible approaches is showcased in Figure 4.3. If the main title is connected to all other titles, it leads to a situation, where it is not clear to which bite the main title belongs. The remaining options are to: connect the main title to the smaller title of the first entry or not connecting it to anything. Both approaches simplify the segmentation but sacrifice some contextual information. In the implementation, the data are processed in such a way that the main title is considered as a part of the first entry. However, this only applies if there is a labeled connection. In the case of non-existing connection, the main title is considered as a standalone bite.

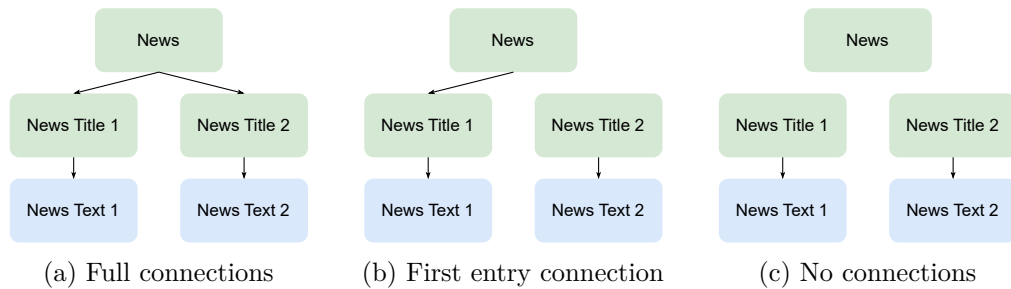


Figure 4.3: Three different strategies for the labeling of relationships between hierarchical titles in historic documents.

Chapter 5

Proposed Method and Implementation

Two different approaches to perform the segmentation were designed and implemented. A baseline solution was developed based solely on the geometric properties of the page and text lines. This method does not use any deep learning techniques. Subsequently, two heuristic methods were added to the baseline solution to improve its performance: a heuristic based on the distance between two neighboring text lines and a heuristic based on a BERT-like language model.

Contrasting with the baseline approach, the main proposed method involves a two-stage process: initial detection of regions of interest containing text lines (proto-bites) using the YOLOv8 object detection model, followed by joining them to form bites using a graph neural network. This chapter begins with obtaining a geometry description for a given page in Section 5.1, which forms the basis for all methods. The baseline method with its heuristics is explained in Section 5.2. The first stage of the main method is described in Section 5.3, while the second stage is explained in Section 5.4. The implementation is available online in GitHub repository¹.

5.1 Page Geometry

Both the baseline method and the graph neural network for proto-bite joining use geometric features of text lines or regions. However, PERO-OCR only provides enclosing polygons and text transcriptions of the individual lines, neglecting any ordering or relationships. Therefore, a data structure available of conveying such information had to be created using the PERO-OCR outputs only. The page geometry created here is in a form of a directed graph, where each text line represents a node in the graph. A constraint is imposed on this graph, each node can only have a maximum of two outgoing edges, one pointing to the precedent text line and the second one pointing to the subsequent text line. In practice, this means the lines above and below the current text line. A visualization of the created geometry can be seen in Figure 5.1.

There are two main strengths: this branching structure implicitly creates a reading order for the page in local neighborhoods. In pages with only one column present, a global reading order is created in most cases. The second strength comes from the ability to infer

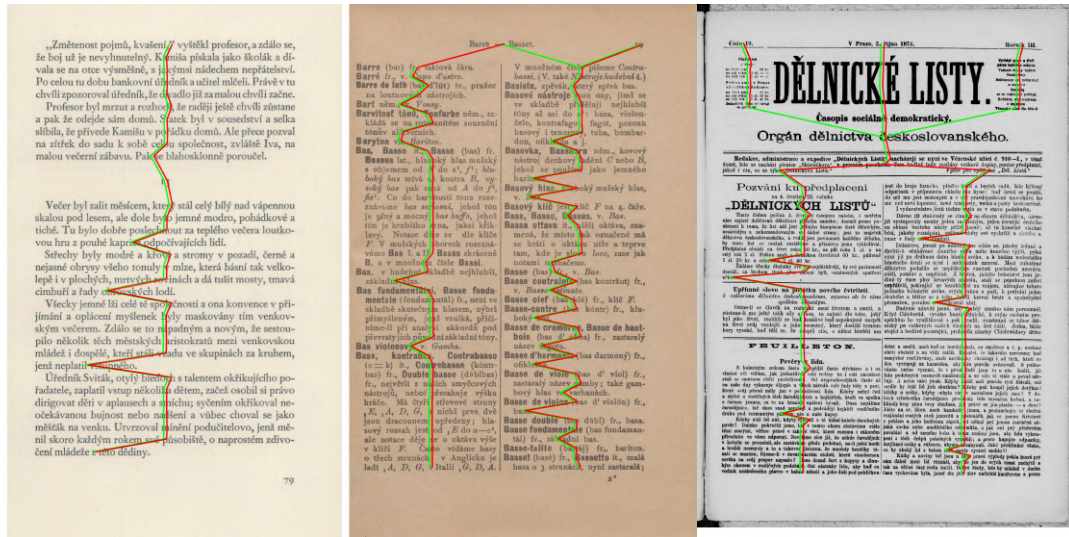
¹<https://github.com/Faz0lek/textbite/>

geometric features like the euclidean distance between two consecutive text lines or the amount of text lines in the column.

The geometry is created by first initializing all nodes of the graph with the text lines. Then, each node is assigned a predecessor and a successor, if they exist. In this process, the text lines are represented solely by their enclosing AABB. A text line A is the predecessor of text line B if it satisfies the following conditions:

1. A is not B,
2. horizontal overlap of A and B is at least 1 px,
3. the center of A is vertically located above the center of B, and
4. of all text lines satisfying 2 and 3, the center of A is the lowest.

The successor is defined in a symmetrical fashion, the verticality constraints in 2 and 3 are reversed.



(a) Book

(b) Dictionary

(c) Periodical

Figure 5.1: Showcase of a page geometry on three different documents. Red lines denote a direction from the line to its predecessor, green lines denote a direction from the line to its successor. The image of the periodical shows that a text line can be the predecessor or the successor to multiple other text lines at the same time.

5.2 Baseline Solution

The baseline solution serves as a reference point for the main, method. In the base form, it is implemented without the use of any machine learning techniques. It is based on the geometry of the page. The relations between text lines in the geometry might not be symmetric and if they are not, the connections are severed, resulting in individual text segments. Such segments are considered as individual bites, as they are disjoint sets of text lines. Figure 5.2 illustrates the process of disconnecting these non-symmetric connections.

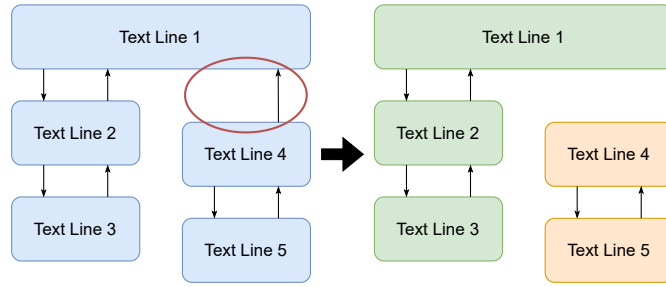


Figure 5.2: The working of the baseline method. All asymmetric connections in the page geometry are removed, resulting in a segmented document. The asymmetry is denoted by the red ellipse. Each segment can be considered as a standalone bite, which is illustrated by the different colors.

As discussed later in Chapter 6, the main advantage of this method lies in its speed and simplicity. However, it struggles greatly in linear documents with tight spacing between the text lines. This especially emerges in dictionaries, as they try to eliminate as much empty space as possible. To solve the issues, we implemented two heuristic methods, which further split the segments created by the baseline method into smaller pieces, were introduced. The first one is a heuristic based on the average distance between each pair of consecutive text lines. The second one is a deep learning approach. The heuristic function is represented by a language model performing next sentence prediction. Another weaknesses are the inability to detect bites spanning multiple columns and capturing the bite names, which is one of the secondary goals discussed in Chapter 3.

5.2.1 Distance Heuristic

The page geometry used in the baseline method is not in any way separated by the vertical distance of two lines. This leads to a situation, where two consecutive text lines (as far as the page geometry is concerned) are part of the same bite, even though there might be a significant vertical gap between them. This is usually not the case in reality, as a vertical gaps signal a break in the text and a possible change of the topic.

To take the vertical gaps into account, a heuristic method, which further splits the bites generated by the baseline method, is added to the pipeline. First, the average vertical distance between consecutive text lines is calculated. This is done on the entire page geometry, not in the input bites only. Then, pairs of text lines are iterated over in each input bite and a break is inserted, if the distance between the two lines is greater than the average for the entire page, creating a new bite. The average distance might not be the optimal value for the break insertion, so a linear scaling hyperparameter is introduced. The process of tuning this hyperparameter is described in Section 6.1.

5.2.2 Language Model Heuristic

Both the baseline method and the distance heuristic do not use the text transcriptions provided by the PERO-OCR, discarding a lot of useful information that could be used to aid the segmentation process. The second heuristic works in a similar way to the distance heuristic. It inserts breaks into existing bites, splitting them into multiple smaller bites. In this case, a deep learning approach is used – a BERT-like language model specifically fine-tuned for the next sentence prediction task. Such model returns a probability that two

text lines (their transcriptions) follow each other. If the probability of succession is lower than a certain threshold, a break is inserted. The value for the threshold was experimented with, finding a value yielding the best results.

As a starting point, a pre-trained CZERT language model (Section 2.4.2) was fine-tuned on the training data. It is freely available on the Huggingface² platform in multiple variants. In this project, the variant *Czert-B-base-cased* is used. The model was already pre-trained on the NSP task (along the MLM task), but the authors published only the transformer backbone, the classification heads are not included.

CZERT is a rather large model in size, with the total of 110M parameters, making a single NSP inference take a considerable amount of time. Combined with the fact that the NSP needs to be calculated for each pair of consecutive sentences, this approach slows the baseline solution. To address this, smaller BERT-like models with different output embedding sizes were created. In total, four different models with embedding sizes of 72, 132, 264 and 516 were pre-trained on a large corpus of Czech historic books.

The pre-training was done in a similar fashion to the original BERT model, except a few modifications. The training data was split into smaller regions of roughly 50 lines. When sampling the negative samples of sentence pairs, only sentences in the current region were considered. Another modification lies in the tokenization process, the [SEP] token between the two sentences is fixed in place, with the sentences padded from both sides to a fixed sequence length of 65 tokens. The tokenized pairs then have the following form:

[CLS] [PAD] ... [TOK] [TOK] ... [SEP] [TOK] [TOK] ... [PAD] ... [SEP].

The original BERT uses the contextual embedding of the [CLS] token for all downstream tasks. To specifically promote the NSP task that is used here, the embedding of the SEP token placed between the two sentences is used for classification instead.

5.3 YOLOv8 Detection

The first stage of the main proposed method is the detection of regions of interest using the YOLOv8 object detection model. Similarly to the CZERT model in the baseline method, a pre-trained YOLOv8 from Ultralytics³ is used and fine-tuned on our training and validation data. The training process and results are discussed in more detail in Section 6.2. The model finds bounding boxes directly on the image of a page and classifies them into two classes: titles and text regions. In the case that the model detects a bounding box that is fully enclosed by another of, larger bounding box of the same class, it gets filtered out to avoid possible inconsistencies. However, the overlapping bounding boxes do not have to be of the same class – the model is specifically trained for this. During the labeling process, the bite names were assigned their own, usually small, bounding box. An example is shown in Figure 5.3. When a bounding box is detected in this way, it is assigned to the enclosing box as it’s name, solving one of the secondary objectives defined in Chapter 3.

To assign the text lines to detected bounding boxes, obtaining the bite representation, each text line provided by the PERO-OCR is assigned to the bounding box with the highest intersection over union. Note that not every single text line has to be assigned, as explained in Chapter 3, some lines like page numbers are of no interest. Because the bite names are a part of the bite itself, they are usually represented by a small segment of one text line.

²<https://huggingface.co/UWB-AIR/Czert-B-base-cased-long-zero-shot>

³<https://github.com/ultralytics>

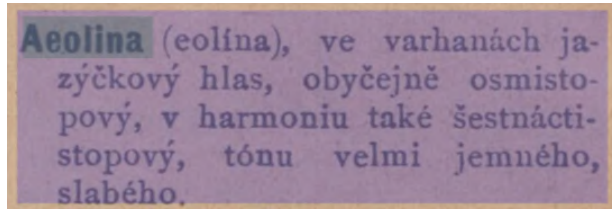


Figure 5.3: A dictionary labeling example of a single bite (blue) with a name (green) enclosed within.

The bounding boxes of individual words are matched with the name bounding boxes using a slightly different metric. Instead of IoU, the intersection is divided by the area of the word bounding box. To avoid edge cases, only words with the intersection over area larger than 0.2 are used. The matched words are then concatenated, forming the name of the bite.

This approach shares one weakness with all the baseline variants. According to the detection results, all predicted regions are standalone bites (with the exception to the bite names). In reality, a bite can continue in the next column or have a title associated with it, which would be placed above the text region itself. To overcome this issue, a second model is introduced with the objective of connecting these proto-bites.

YOLO Format

Training the YOLOv8 model requires labels in the YOLO format. YOLO format is a text format for representing objects present in an image. The objects are described by AABBs, where each line corresponds to one bounding box and has the following format:

```
<label> <x> <y> <width> <height>
```

Where `label` is an integer, in this case either 0 for text or 1 for title, `x` and `y` are the normalized coordinates of the bounding box center and `width` and `height` are normalized dimensions.

5.4 Proto-Bite Joining

To join proto-bites generated by the YOLOv8 model, two methods were implemented. The YOLOv8 results can be efficiently represented by a graph, where the nodes of the graph are represented by detected regions. The first, main method is based on a graph neural network. The second method was developed at a later stage as an experiment to analyze whether global feature aggregation of the GNN brings any benefits. It is based on a multi-layer perceptron classifier (MLP), which attempts to classify a flattened version of the input graphs. Both methods are explained in the following sections.

5.4.1 Joining with GNN

The first variant of a model for bite joining is based on a graph neural network. It is implemented using the PyTorch Geometric⁴ library, which is a library for creating and training graph neural networks. The model is designed to work as a binary classifier of

⁴<https://pytorch-geometric.readthedocs.io/en/latest/index.html>

the edges in the graph. The label of the edge signifies, whether two nodes representing a proto-bite should be merged together.

During the data labeling process, relations were only made between the consecutive regions, not between all the regions belonging to the bite. This raised a question as to how to label the edges of bites composed of more than two proto-bites. The two possible labeling strategies are shown in Figure 5.4. Based on the fact that all proto-bites should be merged together, the strategy of positively labeling all edges was used. Experiments with the model, hyperparameters and the model structure are later described in Section 6.3.

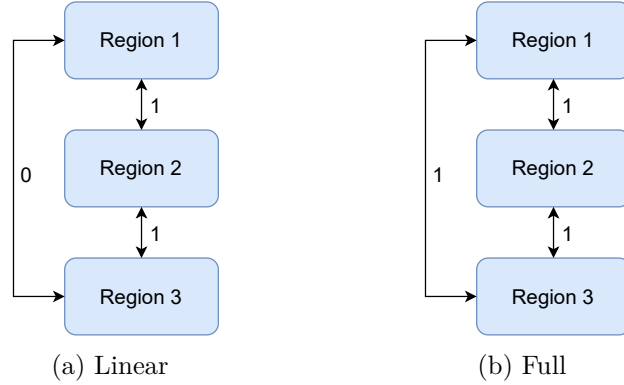


Figure 5.4: Two strategies for creating labels in bites with more than two regions.

The implemented model is composed of multiple layers. First, the features are linearly projected to the hidden size of the subsequent graph convolutional blocks. These blocks are composed of the graph convolutional layers from Section 2.2.2, layer normalization and the GELU activation function. The model architecture is visualized in Figure 5.5. Lastly, the convoluted features are linearly projected to the original input size. Dropout is also used right after the input projection to improve the generalization capabilities of the model. After a forward pass of a graph through the model, a new graph representation is available.

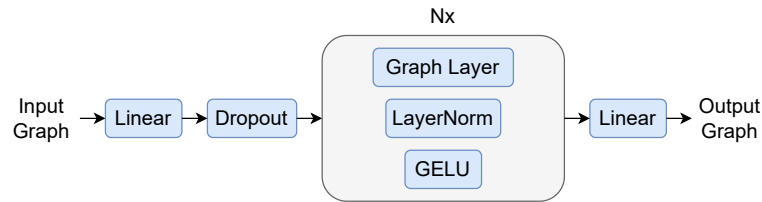


Figure 5.5: The model structure of the graph neural network used for joining the proto-bites obtained using the YOLOv8 detector. After initial projection and dropout, a graph convolutional layer is applied and its outputs are normalized. Nonlinearity is introduced by applying the GELU function on the normalized values.

To classify an edge, cosine similarity is calculated between the new node representations, which is then compared against a classification threshold.

Feature Engineering and Graph Initialization

Some graph operators implemented in the PyTorch Geometric library allow both the graph nodes and edges to hold features vectors. The node features describe the properties

of a single region, while the edge features describe their relationships and differences. To pass as much information into the network as possible, node features and edge features are utilized. There are a total of 29 geometric features representing the nodes and 12 edge features. The edges features are a mix of geometric and text features.

The geometric edge features are: distances between the centers of the two regions (both absolute and relative to the page dimensions) in both axes, euclidean distance between centers and then the same distances, but between the closest edges of the regions. The text features are represented by the outputs of a pre-trained CZERT model. The Huggingface implementation of the BERT models provides the contextual representation of the CLS token and a pooler representation, which is obtained by applying a fully connected layer and a hyperbolic tangent activation function to the CLS token representation. These two values are calculated for both regions connected by the edge, and their cosine similarity represents the final features.

Normalization

Both the node and edge features contain values both absolute and relative. This creates a situation where the scale and value ranges differ significantly between features, which can lead to issues while training the model. To solve this, all features are normalized using the Z-Score normalization, which sets the mean of all the values for a specific features to 0 and the standard deviation to 1. The normalization is performed by applying the following:

$$z = \frac{x - \bar{\mu}}{\bar{\sigma}},$$

where z is the new, normalized value, x is the old value, $\bar{\mu}$ is the empirical mean value of the feature and $\bar{\sigma}$ is its variance. The mean values and standard deviations for all features are calculated from the training dataset and these values are used to normalize the features of validation and test datasets as well as during inference.

Graph Initialization and Batching

The graphs are created as complete undirected graphs. During the experimentation stage, some attempts on edge pruning were done, but since the amount of nodes is small, it did not yield any performance improvement. The PyTorch Geometric does not allow for any efficient undirected graph representation, each pair of nodes has to have two directed edges between them in both directions with identical feature vectors.

Batching graphs differs from batching other types of data. When a model accepts a single feature vector as input, multiple vectors can be stacked into a batch by concatenating them along a new spatial dimension. However, with graphs, batching involves joining them into one larger graph, where each disconnected subgraph represents one of the original graphs.

5.4.2 Joining Using MLP

To analyze whether a graph neural network approach yielded any meaningful performance increase over a simpler model, an MLP approach was developed. The graphs created are flattened into individual feature vectors. A feature vector is created for every edge present in the graph and is obtained by concatenating the node features of both regions with the

edge features in the following scheme:

$$\left[n_1^{(1)}, n_2^{(1)}, \dots, n_F^{(1)}, n_1^{(2)}, n_2^{(2)}, \dots, n_F^{(2)}, e_1, e_2, \dots, e_E \right].$$

These flattened feature vectors are then passed through the MLP, which, similarly to the GNN, works as an edge classifier with a connection probability as its output. If this probability is higher than a classification threshold, the regions are merged.

5.5 Output File Formats

The final output consists of two files: a JSON file containing the representation of each bite in a given file and an enhanced PAGE-XML file. The text lines in the JSON file are represented by their ID, in addition, the bite name, class and bounding box are provided as well. The JSON representation is mainly used for further processing, evaluation and visualization purposes. An example of one bite within the JSON file is shown in Listing 5.1.

Listing 5.1: Example of a single bite in the output JSON file.

```
{
  "cls": "text",
  "bbox": [
    519.0216674804688,
    702.5872192382812,
    2175.725341796875,
    1000.8356323242188
  ],
  "lines": [
    "07662546-8163-4721-af02-ea239b00ae4f",
    "e7935103-951f-45ee-8c26-9b548d98c5de",
    "c2893624-956f-4212-aacb-719211bd0353",
    "0ac3b8a3-68a1-4cdd-a33d-808d0b67c2ff",
    "ce988abb-c69d-481a-8412-85f4a9fcf5d6"
  ],
  "name": "Pastikove formy"
}
```

Concerning the PAGE-XML file, there are two changes compared to the original files provided by PERO-OCR. First, regions are labeled using the *type* field, e.g.: `<TextRegion id="r002" type="heading">` and second, a new structure is created, specifying an explicit reading order that groups regions together, if they are a part of one bite. An example of the grouping is shown in Listing 5.2. The implementation of this PAGE-XML enhancing process was created in collaboration with my supervisor.

Listing 5.2: Example of the XML structure added to a processed PAGE-XML document. It joins several regions together into logical segments of texts called bites.

```
<ReadingOrder>
  <UnorderedGroup id="root">
    <OrderedGroup id="bite_1">
      <RegionRefIndexed regionRef="r007" index="0"/>
    </OrderedGroup>
  </UnorderedGroup>
</ReadingOrder>
```

```
<OrderedGroup id="bite_2">
  <RegionRefIndexed regionRef="r001" index="0"/>
  <RegionRefIndexed regionRef="r005" index="1"/>
</OrderedGroup>
<OrderedGroup id="bite_3">
  <RegionRefIndexed regionRef="r000" index="0"/>
</OrderedGroup>
<OrderedGroup id="bite_4">
  <RegionRefIndexed regionRef="r003" index="0"/>
  <RegionRefIndexed regionRef="r004" index="1"/>
  <RegionRefIndexed regionRef="r002" index="2"/>
</OrderedGroup>
</UnorderedGroup>
</ReadingOrder>
```

Chapter 6

Experiments and Results

This chapter focuses on the experiments done with the models, their training and hyperparameter tuning. First, experiments with the heuristics of the baseline method are conducted, followed by the fine-tuning of the YOLOv8 detector and the joining of protobites using the GNN. At the end, an overview of all methods, experiments and result is provided. All experiments were conducted on SGE computational cluster provided by BUT FIT. All time measurements were done on NVIDIA RTX A6000.

6.1 Baseline Experiments and Results

The baseline method in its basic form does not contain any machine learning model that requires training or hyperparameters that could be tuned. However, as discussed in Section 5.2.1, the average distance between lines might not be the optimum value for inserting breaks. To address this, the average value is multiplied by a hyperparameter. The optimal value for this hyperparameter was found by iterating over values in the range of $[0.2, 10]$ with a step of 0.05. The target metric was the V-measure on the validation data and the results are plotted in Figure 6.1. The resulting optimal value for the coefficient is 1.45, achieving V-measure of 67.07 %.

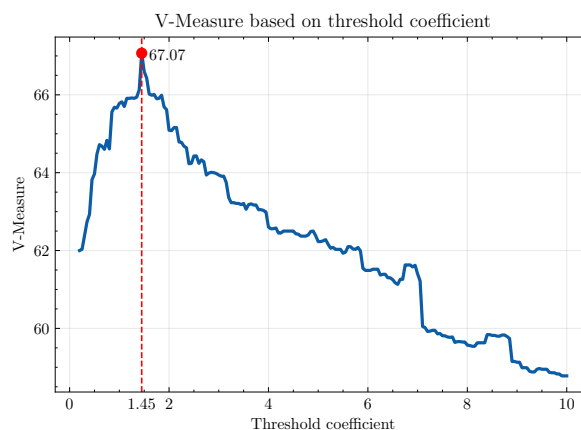


Figure 6.1: Finding the optimal coefficient for the average line distance in the baseline method with distance heuristic with a threshold step of 0.05.

6.1.1 Language Model Training

Section 5.2.2 discussed the problem with the size and slow inference time of the CZERT model. To reduce the computational cost, custom BERT-like language model were pre-trained with four different output embedding sizes: 64, 132, 264 and 516. Only other architecture change besides the output embedding size was the hidden size of the Feed Forward layer, which is set to be a quadruple of the output size. The models were trained on a dataset consisting of 61M sentence pairs taken from various books available in digital libraries. The models were trained for three epochs, after which the MLM objective was dropped and the models continued to train on the NSP task only. The Adam optimization algorithm was used with the learning rate of $5e^{-5}$ and one training batch contained 128 sentence pairs. When the models were pre-trained, they were fine-tuned on the training data created for this project for one epoch using the AdamW optimizer with a learning rate of $1e^{-4}$, while the batch size remained the same. The training progress of the language model with 264 output features is shown in Figure 6.2. The Figure only shows the first three epochs of the training process, the NSP continuation and subsequent fine-tuning is not included.

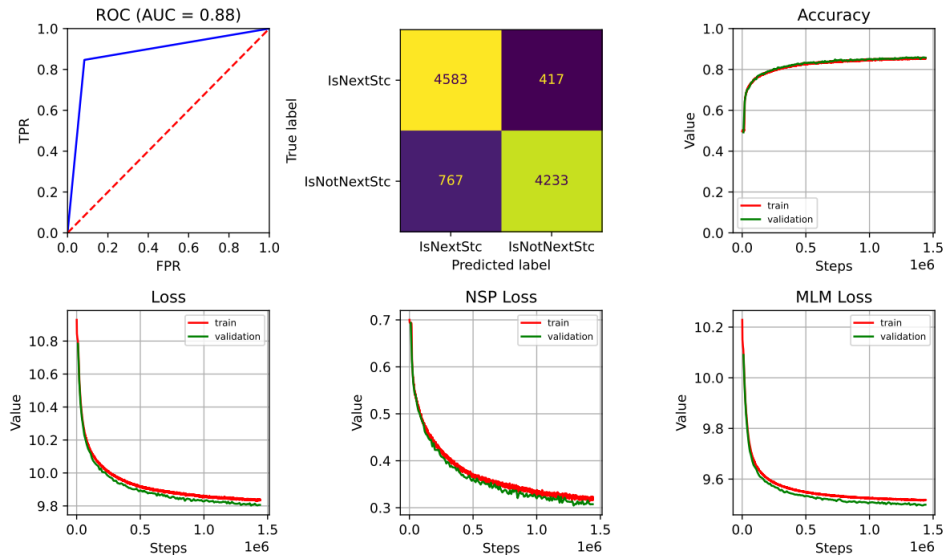


Figure 6.2: Pre-training of custom BERT-like language model with 264 output embedding size. The accuracy measures the NSP task only. The training progressed smoothly and after around 1M steps, performance on the MLM objective stagnated. In response, the MLM task was dropped and the model continued to train on the NSP task only.

Both the base and distance based variant are extremely fast. However, the use of the language model slows down the inference considerably, as forward pass needs to be done for each pair of consecutive text lines. All measured times are displayed in Table 6.1. The custom models show some improvement over the big CZERT model, but a slightly larger improvement was expected.

6.1.2 Comparing Baseline Variants

The baseline and its variants serve as the first attempt at segmentation, laying a foundation for the main method. The baseline variant with no heuristic only works quite well

Table 6.1: Inference times of the baseline method and all its variants. Adding the distance based heuristic does not hinder the inference time at all, while the language model based heuristic slows down the inference considerably.

Method	Time [s]
Baseline	0.16
Baseline+Dist	0.16
Baseline+LM72	1.35
Baseline+LM132	1.50
Baseline+LM264	1.62
Baseline+LM516	1.71
Baseline+CZERT	1.84

in very simple, linear documents, but, the performance is subpar when the page layout complexity increases even by a small margin. However, the distance based heuristic seems to greatly improve the performance, especially in those complex layouts. The language model heuristic displays similar quantitative performance as the distance based heuristic, but when examining the individual pages directly, the language models often insert text breaks where there should not be any. The quantitative results can be seen in Table 6.2 and example pages of all three variants are shown in Figure 6.3. The language models trained and used all show comparable performance to one another, making the smallest model the best choice due to the lowest inference time. The baseline variant with the distance heuristic displays best results both quantitatively and qualitatively, making it the baseline variant used for the comparison with the main method.

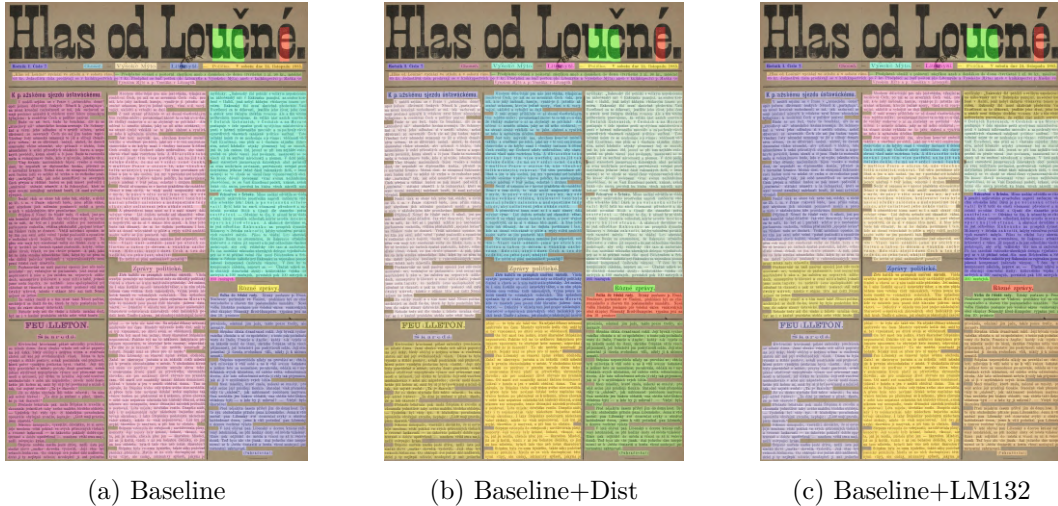


Figure 6.3: Examples of segmented pages by the baseline method and its two heuristics: the distance based and language model based heuristics. The plain baseline fails to detect any horizontal breaks in the segment, while the language model based variant breaks segments at wrong places. The distance based variant shows the best results.

Table 6.2: Results of the baseline method and its variants, including all language models. The language model based variant improves the plain baseline method, but the distance based variant shows the best results overall. The best value in each metric is highlighted.

Method	Books			Dictionaries			Periodicals		
	H	C	V	H	C	V	H	C	V
Baseline	82.61	26.63	19.72	79.22	31.42	40.11	70.01	48.85	52.20
Baseline+Dist	41.69	95.26	49.77	79.91	60.29	64.67	72.51	89.03	77.40
Baseline+LM72	41.74	87.87	44.84	73.5	81.34	75.01	60.59	84.60	68.46
Baseline+LM132	39.57	91.65	44.32	75.47	83.52	77.19	62.53	87.12	70.66
Baseline+LM264	41.31	94.38	49.83	76.05	72.07	71.00	64.25	83.33	69.90
Baseline+LM516	47.50	89.19	51.14	75.24	70.57	69.88	63.50	82.68	69.32
Baseline+CZERT	39.46	85.47	44.14	73.51	81.84	74.23	60.62	82.61	66.50

6.2 YOLOv8 Experiments and Results

The YOLOv8 models are available in several sizes: nano, small, medium and others. Experiments were done only with the first three, because the larger models were deemed to be too big for the task. There are several areas of interest when it comes to training and evaluating the detector: the training metrics, inference speed and the performance on the validation dataset.

6.2.1 YOLOv8 Training

When it comes to the training of the YOLOv8 models, the target metrics are mAP50-90 and the mAP50 from Section 2.3.3. The Ultralytics library for the model training allows automatic resizing of the input images, allowing the models to be trained using different resolutions. Experiments were done with 5 different resolutions: 640px, 800px, 1000px, 1200px and 1400px. The value denotes the size of the longer side of the image, while the shorter side is scaled accordingly. The resolution of the original images was around 2500x3500px. In total, 14 models were trained – three model sizes with five resolution values each. The medium variant with the highest resolution was omitted, as it was too computational heavy.

The models were trained for the maximum of 200 epochs, with an improvement patience of 50 epochs, meaning that if the model did not see any improvements over the last 50 epochs, the training process was terminated. The used optimization algorithm is AdamW for the first 10000 iterations, after which it is switched to SGD, which is the default Ultralytics configuration. Both optimizers use learning rate of 0.01. Each training batch was composed of 16 images.

The training process of the YOLOv8 small variant with the resolution of 800px is displayed in Figure 6.4. Table 6.3 shows the target metrics for each model variant along the final V-Measure on the validation dataset. The best values of each metric are highlighted, but the performance variance is small between the models and also changes between the individual metrics.

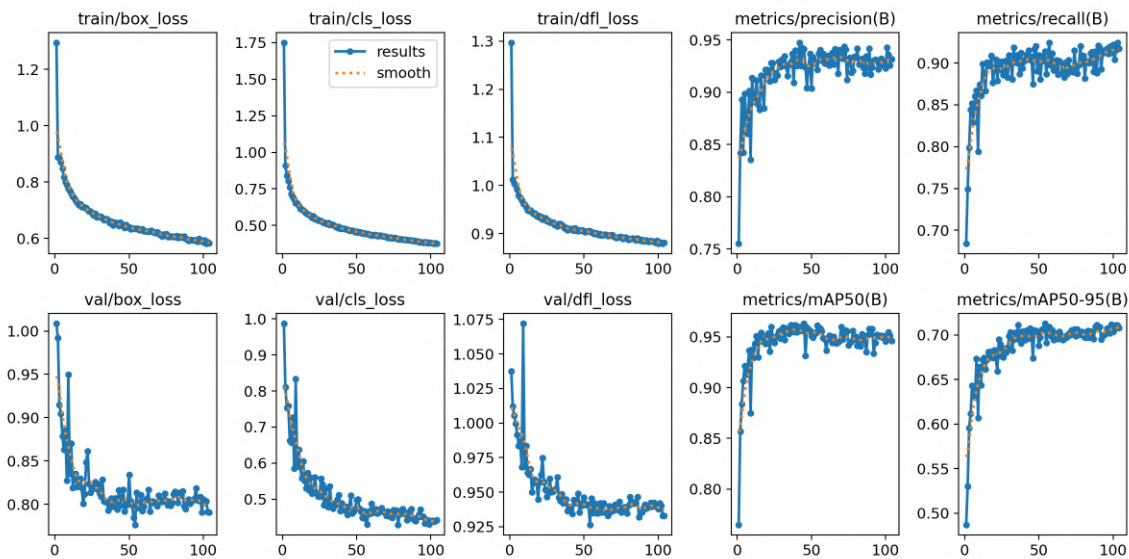


Figure 6.4: The training process of the YOLOv8-s detection model with the resolution of 800px on the larger side of the image. This is the model version that was chosen as the best variant.

Table 6.3: Training results of YOLOv8 object detection models in three sizes with five different resolutions and three different target metrics. The best value in each category is highlighted.

Resolution	640	800	1000	1200	1400
mAP50-95					
YOLOv8n	0.696	0.710	0.720	0.728	0.725
YOLOv8s	0.700	0.713	0.722	0.723	0.725
YOLOv8m	0.698	0.720	0.722	0.726	-
mAP50					
YOLOv8n	0.957	0.957	0.966	0.964	0.970
YOLOv8s	0.948	0.961	0.964	0.963	0.957
YOLOv8m	0.950	0.964	0.969	0.961	-
V-Measure					
YOLOv8n	0.825	0.834	0.801	0.830	0.805
YOLOv8s	0.823	0.840	0.839	0.843	0.816
YOLOv8m	0.833	0.831	0.848	0.841	-

Another metric that is specifically important for the inference is the time it takes for the model to detect the regions. Average time each model took was measured on the validation dataset and the results are displayed in Table 6.4. The inference time increases with both the model sizes and the resolution used.

Table 6.4: YOLOv8 inference times on the validation dataset, average per page.

Resolution	640	800	1000	1200	1400
			Time [ms]		
YOLOv8n	116	117	128	143	146
YOLOv8s	131	135	147	163	182
YOLOv8m	146	171	196	245	-

6.2.2 YOLOv8 Results

The final model was selected based on the the combination of the object detection metrics, V-Measure on the validation dataset and the inference time. All models were very close in terms of all the training metrics, but varied in the inference time. The selected model, YOLOv8-s with the resolution of 800px, was chosen for having low inference time while achieving average to above average training metrics.

Examples of text pages segmented by the model are shown in Figure 6.5. The detection of bite names is displayed in the examples, but the performance on this objective was not measured quantitatively on its own. However, because they were labeled as individual regions, some precision is captured by the mAP metric. The performance of the selected model was evaluated on the test dataset, where it achieved the V-measure of 65.33%, 93.36% and 83.93% for books, dictionaries and periodicals, respectively.



Figure 6.5: Logical text segments detected by the YOLOv8-s visual detector working with the resolution of 800px of the longer side. Individual logical segments are denoted by the red color, titles and segment names are denoted by the color blue.

6.3 Graph Neural Network Experiments and Results

As described in Chapter 5, regions detected by the YOLOv8 model are joined together using a graph neural network. The main things that can be experimented with are the model architecture (different graph layers and their number) and the training hyperparameters.

6.3.1 GNN Training

The graph neural network for bite joining works as a edge classification model, where positive edge prediction means that the connected bites should be merged. To monitor the training process, precision, recall and F1-score can be used. However, avoiding the joining of two wrong bites is a priority. Therefore, the target metric is precision on the positive edges. The evaluation is done separately for each document type, but there are two problems: the dataset is unbalanced between document types and also the documents contain different number of edges. Books generally have less edges than dictionaries and periodicals. This means that high precision could be achieved, while having subpar performance on book pages. To avoid this, the target metric is the sum of precisions for all document types – the value range becomes $[0, 3]$.

To achieve the best possible performance, experiments were done with various model configurations. All experiments are shown in Figure 6.6. When it comes to the graph layer used, three variants were considered: *GCNConv*, *GATConv* and *ResGatedGraphConv*. *GCNConv* was used at initial stages of development, but it was discarded since it does not support edge features. The results show that the *GATConv* does not learn at all, while the *ResGatedGraphConv* layer shows great performance. The number of blocks, learning rate and batch size do not seem to affect the model performance in a meaningful way, while using larger hidden size in the blocks improves it. The performance also seems to be extremely sensitive to the classification threshold – values larger than 0.5 are generally better, but the classification collapses to one class with values larger than 0.72. To further improve the model, a closer tuning of the threshold was performed and the results are shown in Figure 6.7.

The final model uses three blocks with the hidden size of 512. It uses a learning rate of $5e^{-3}$ and each batch contains 64 graphs. The classification threshold was set to 0.7 during training and then changed to 0.68 after tuning. The training of this specific model is shown in Figure 6.8.

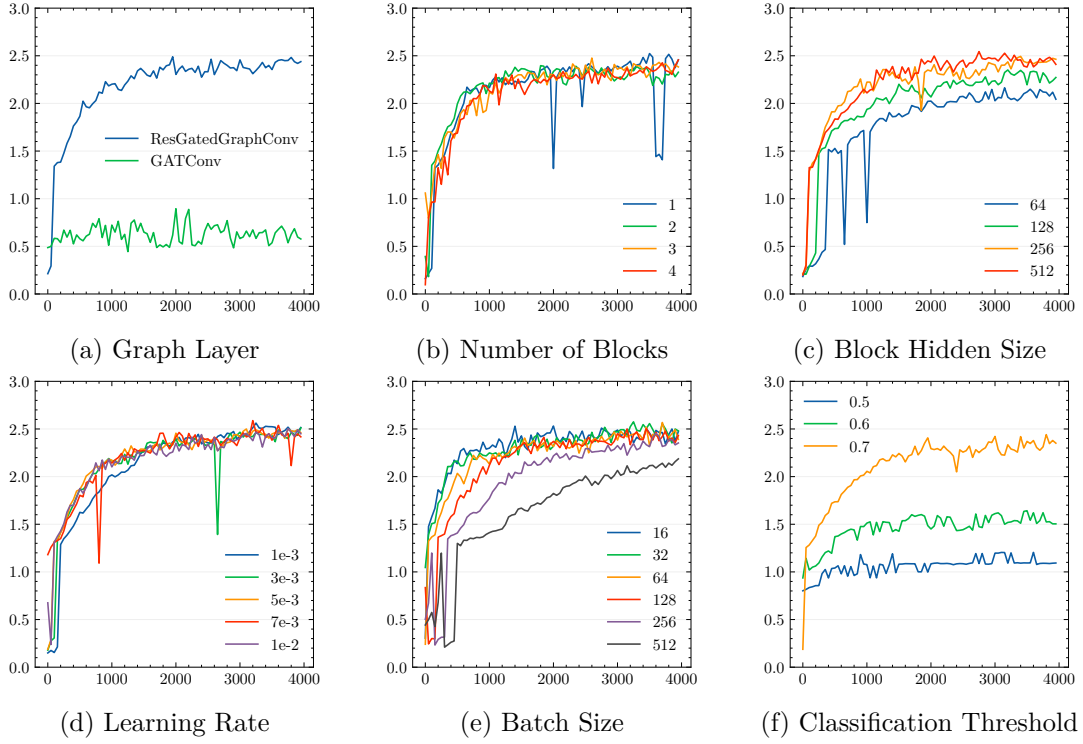


Figure 6.6: Tuning of the hyperparameters of the graph neural network used for the joining of regions. The default model uses three blocks with the hidden size of 64, learning rate of $5e^{-3}$, batch size of 64 and the classification threshold of 0.7.

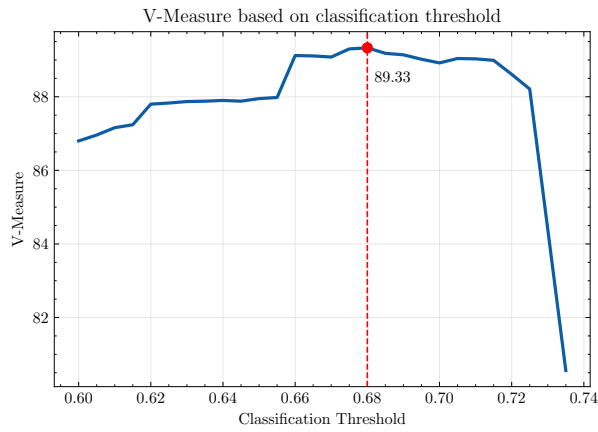


Figure 6.7: Tuning of the classification threshold in graph neural network on the validation dataset. Around the threshold value of 0.72, the classification starts to collapse to one class.

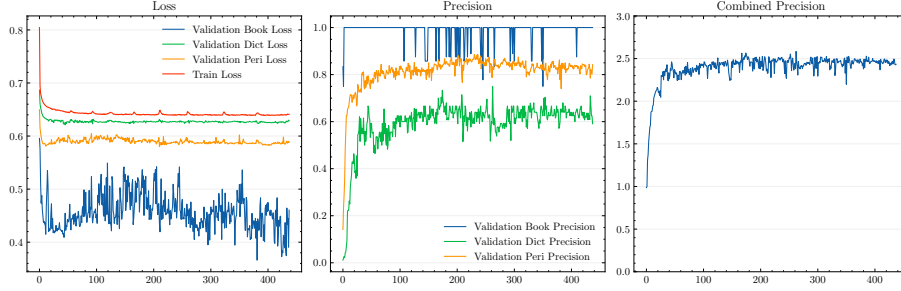


Figure 6.8: The training process of the graph neural network for bite joining. The model was trained with three graph operator blocks with the hidden size of 256

6.3.2 Results and Comparison with MLP

After training the model, the final performance was evaluated on the test dataset. The graph neural network achieved a V-Measure of 77.93 %, 95.79 % and 90.23 % for books, dictionaries and periodicals, respectively. This result upgrades the bare YOLOv8 approach by roughly 6 %. In terms of inference speed, processing a single text page takes 660 ms on average.

As explained in Section 5.4.2, MLP classifier can be used to see whether the global feature aggregation of the graph neural network is useful compared to using just local features of the two neighboring regions. A simple MLP was trained with three layers with the hidden size of 64. This model yielded a V-measure of 66.58 %, 94.97 % and 84.01 % for books, dictionaries and periodicals, respectively, which is significantly worse than the graph neural network on books, similar in dictionaries and worse in periodicals.

6.4 Results Overview and Discussion

The basic baseline method fails to meaningfully segment the pages. The heuristic approaches offer some improvement, but the results are still much worse than the main method, while also being significantly slower on average. The YOLOv8 detection models were trained in three sizes with five different resolutions. Here, some more experiments could have been done, especially with data augmentation or hyperparameter tuning in the Ultralytics training framework. The models show very similar performance to one another and the performance overall is good, outperforming the baseline solution even without the joining of the regions. The subsequent joining of the regions using a graph neural network further enhances the results, outperforming all baseline variants. The graph based model was also compared to the MLP classifier to assess the importance of using a global method. While the MLP slightly improves the segmentation, the graph neural network works better. In total 21 models were trained and experimented with. All results are shown in Table 6.5 and examples of segmented pages using the main method can be seen in Figure 6.9.

Table 6.5: Final results of all the implemented methods. The language model used in the Baseline+LM method is the custom model with 132 output features, which displayed the best overall performance. The main method, YOLOv8+GNN, produces the best results.

Method	Books			Dictionaries			Periodicals		
	H	C	V	H	C	V	H	C	V
Baseline+Dist	41.69	95.26	49.77	79.91	60.29	64.67	72.51	89.03	77.40
Baseline+LM	39.57	91.65	44.32	75.47	83.52	77.19	62.53	87.12	70.66
YOLOv8	61.06	92.86	65.33	90.26	99.14	93.36	80.76	92.94	83.93
YOLOv8+MLP	70.62	85.48	66.58	93.30	98.75	94.97	82.97	89.92	84.01
YOLOv8+GNN	83.39	92.81	77.93	95.24	97.90	95.79	93.16	90.58	90.23



(a) Baseline+Dist

(b) YOLOv8

(c) YOLOv8+GNN

Figure 6.9: Logical text segments detected by the baseline, the YOLOv8-s visual detector and the a graph neural network. Individual logical segments are denoted by different colors. Notice the importance of GNN for the joining of regions.

Chapter 7

Conclusion

The goal of this project was to research the state-of-the-art methods in semantic text segmentation and then design and implement a new segmentation system able to segment historical documents into logical units like dictionary entries or newspaper articles. The project was presented at the Excel@FIT conference hosted by BUT FIT, where it received an award from the academic committee and an award from an industrial partner.

Most text segmentation methods insert breaks into continuous texts directly, while discarding any geometric information about the page. Therefore, the proposed segmentation pipeline is a combination of smaller models utilizing both the language and geometric properties of text pages.

In collaboration with several Czech libraries, the project aimed to aid in the digitization process of historic documents, focusing solely on documents in the Czech language. As there are no public datasets available for text segmentation in Czech, a custom dataset was created using text pages from books, dictionaries and periodicals obtained from digital libraries. The dataset, composed of 4044 pages, was labeled with the help of librarians and other BUT FIT students.

The first step of the digitization process is the analysis of the page layout and OCR. Both are achieved using PERO-OCR, which serves as the basis for this project. However, PERO-OCR does not guarantee the reading order of all text lines, which means that classic text segmentation metrics cannot be used. I treat this problem as a line clustering problem and evaluate the performance using clustering metrics: homogeneity, completeness and V-measure.

I implemented a baseline solution in three variants, which serve as a comparison to the main method. The first two variants are based on imposing geometric constraints on the text page, while the last utilizes our own pre-trained BERT-like model to break text into segments. The main proposed method is based on the YOLOv8 object detector to detect regions containing the first segment representation. These are then refined using a graph neural network that predicts, which segments should be joined together. The graph neural network contains both geometric features and text semantic embedding generated by the CZERT language model. This method achieves the V-measure of 77.93 %, 95.79 % and 90.23 % for books, dictionaries and periodicals, respectively, which exceeds the results of all baseline variants.

A continuation in the project could involve creating a language model for topic embeddings, which could be then used to further improve the performance of the graph neural network. Another idea would be to completely restructure the pipeline into an end-to-end model that would be trained to detect and join the regions in one step.

Bibliography

- [1] BRESSON, X. and LAURENT, T. Residual Gated Graph ConvNets. *CoRR*, 2017, abs/1711.07553. Available at: <http://arxiv.org/abs/1711.07553>.
- [2] DALAL, N. and TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, vol. 1, p. 886–893 vol. 1.
- [3] DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K. et al. Imagenet: A large-scale hierarchical image database. In: *Ieee. 2009 IEEE conference on computer vision and pattern recognition*. 2009, p. 248–255.
- [4] DEVLIN, J.; CHANG, M.-W.; LEE, K. and TOUTANOVA, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: BURSTEIN, J.; DORAN, C. and SOLORIO, T., ed. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, p. 4171–4186. Available at: <https://aclanthology.org/N19-1423>.
- [5] GIRSHICK, R. Fast R-CNN. *CoRR*, 2015, abs/1504.08083. Available at: <https://ieeexplore.ieee.org/document/7410526>.
- [6] GIRSHICK, R.; DONAHUE, J.; DARRELL, T. and MALIK, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, p. 580–587.
- [7] GLAVAŠ, G.; NANNI, F. and PONZETTO, S. Unsupervised Text Segmentation Using Semantic Relatedness Graphs. In: . January 2016, p. 125–130.
- [8] GLAVAŠ, G. and SOMASUNDARAN, S. Two-Level Transformer and Auxiliary Coherence Modeling for Improved Text Segmentation. *CoRR*, 2020, abs/2001.00891. Available at: <http://arxiv.org/abs/2001.00891>.
- [9] HE, K.; ZHANG, X.; REN, S. and SUN, J. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 770–778.
- [10] HEARST, M. A. TextTiling: segmenting text into multi-paragraph subtopic passages. *Comput. Linguist.* Cambridge, MA, USA: MIT Press, mar 1997, vol. 23, no. 1, p. 33–64. ISSN 0891-2017.

- [11] HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, 2017, abs/1704.04861. Available at: <http://arxiv.org/abs/1704.04861>.
- [12] JOCHER, G.; CHAURASIA, A. and QIU, J. *Ultralytics YOLO*. January 2023. Available at: <https://github.com/ultralytics/ultralytics>.
- [13] KANG, J.; TARIQ, S.; OH, H. and WOO, S. A Survey of Deep Learning-Based Object Detection Methods and Datasets for Overhead Imagery. *IEEE Access*, january 2022, vol. 10, p. 1–1.
- [14] KHEMANI, B.; PATIL, S.; KOTTECHA, K. and TANWAR, S. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, january 2024, vol. 11.
- [15] KIPF, T. N. and WELLING, M. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv e-prints*, september 2016, p. arXiv:1609.02907.
- [16] KIŠŠ, M.; BENEŠ, K. and HRADIŠ, M. AT-ST: Self-Training Adaptation Strategy for OCR in Domains with Limited Transcriptions. In: *Lladós J., Lopresti D., Uchida S. (eds) Document Analysis and Recognition - ICDAR 2021*. Springer Nature Switzerland AG, 2021, vol. 12824, p. 463–477. Lecture Notes in Computer Science. ISBN 978-3-030-86336-4. Available at: <https://www.fit.vut.cz/research/publication/12464>.
- [17] KODYM, O. and HRADIŠ, M. Page Layout Analysis System for Unconstrained Historic Documents. In: *Lladós J., Lopresti D., Uchida S. (eds) Document Analysis and Recognition - ICDAR 2021*. Springer Nature Switzerland AG, 2021, p. 492–506. Lecture Notes in Computer Science. ISBN 978-3-030-86330-2. Available at: <https://www.fit.vut.cz/research/publication/12493>.
- [18] KOHÚT, J. and HRADIŠ, M. TS-Net: OCR Trained to Switch Between Text Transcription Styles. In: *Lladós J., Lopresti D., Uchida S. (eds) Document Analysis and Recognition - ICDAR 2021*. Springer Nature Switzerland AG, 2021, vol. 12824, no. 1, p. 478–493. Lecture Notes in Computer Science. ISBN 978-3-030-86336-4. Available at: <https://www.fit.vut.cz/research/publication/12463>.
- [19] KOSHOREK, O.; COHEN, A.; MOR, N.; ROTMAN, M. and BERANT, J. Text Segmentation as a Supervised Learning Task. In: WALKER, M.; JI, H. and STENT, A., ed. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, p. 469–473. Available at: <https://aclanthology.org/N18-2075>.
- [20] KRICHEN, M. Convolutional Neural Networks: A Survey. *Computers*, 2023, vol. 12, no. 8. ISSN 2073-431X. Available at: <https://www.mdpi.com/2073-431X/12/8/151>.
- [21] KRIZHEVSKY, A.; SUTSKEVER, I. and HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F.; BURGESS, C.; BOTTOU, L. and WEINBERGER, K., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, vol. 25.

- [22] LECUN, Y.; BOTTOU, L.; BENGIO, Y. and HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998, vol. 86, no. 11, p. 2278–2324.
- [23] LEE, J. and YOU, S. Balancing Privacy and Accuracy: Exploring the Impact of Data Anonymization on Deep Learning Models in Computer Vision. *IEEE Access*, january 2024, PP, p. 1–1.
- [24] LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR*, 2015, abs/1512.02325. Available at: <http://arxiv.org/abs/1512.02325>.
- [25] MARCHEGGIANI, D. and TITOV, I. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In: PALMER, M.; HWA, R. and RIEDEL, S., ed. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, September 2017, p. 1506–1515.
- [26] PADILLA, R.; LOBATO PASSOS, W.; DIAS, T.; NETTO, S. and SILVA, E. da. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics*, january 2021, vol. 10, p. 279–306.
- [27] PATEL, M. and KALANI, N. A survey on Pose Estimation using Deep Convolutional Neural Networks. *IOP Conference Series: Materials Science and Engineering*, january 2021, vol. 1042, p. 012008.
- [28] PLETSCHACHER, S. and ANTONACOPOULOS, A. The PAGE (Page Analysis and Ground-Truth Elements) Format Framework. In: *2010 20th International Conference on Pattern Recognition*. 2010, p. 257–260.
- [29] REDMON, J.; DIVVALA, S.; GIRSHICK, R. and FARHADI, A. You Only Look Once: Unified, Real-Time Object Detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun 2016, p. 779–788. ISSN 1063-6919.
- [30] REN, S.; HE, K.; GIRSHICK, R. and SUN, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: CORTES, C.; LAWRENCE, N.; LEE, D.; SUGIYAMA, M. and GARNETT, R., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2015, vol. 28.
- [31] ROSENBERG, A. and HIRSCHBERG, J. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In: . January 2007, p. 410–420.
- [32] SIDO, J.; PRAŽÁK, O.; PŘIBÁŇ, P.; PAŠEK, J.; SEJÁK, M. et al. Czert - Czech BERT-like Model for Language Representation. *CoRR*, 2021, abs/2103.13031. Available at: <https://arxiv.org/abs/2103.13031>.
- [33] SIMONYAN, K. and ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: BENGIO, Y. and LECUN, Y., ed. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. Available at: <http://arxiv.org/abs/1409.1556>.

- [34] SUN, X.; PENG, J.; SHEN, Y. and KANG, H. Tobacco Plant Detection in RGB Aerial Images. *Agriculture*, february 2020, vol. 10, p. 57.
- [35] SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S. et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 1–9.
- [36] ULTRALYTICS. *YOLOv5: A state-of-the-art real-time object detection system* <https://docs.ultralytics.com>. 2021.
- [37] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L. et al. Attention is All you Need. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R. et al., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, vol. 30.
- [38] VELIČKOVIĆ, P.; CUCURULL, G.; CASANOVA, A.; ROMERO, A.; LIÒ, P. et al. *Graph Attention Networks*. 2018.
- [39] VIOLA, P. and JONES, M. Rapid Object Detection using a Boosted Cascade of Simple Features. In: February 2001, vol. 1, p. I–511. ISBN 0-7695-1272-0.
- [40] ZHOU, J.; CUI, G.; ZHANG, Z.; YANG, C.; LIU, Z. et al. Graph Neural Networks: A Review of Methods and Applications. *CoRR*, 2018, abs/1812.08434. Available at: <http://arxiv.org/abs/1812.08434>.