

FACULTY OF APPLIED SCIENCES
UNIVERSITY OF WEST BOHEMIA
IN PILSEN

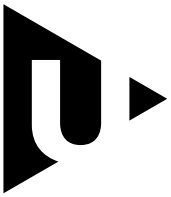
DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Master's Thesis

Automatic Recognition of Parcel Numbers from Historical Maps

Vladimír Holý

PILSEN, CZECH REPUBLIC 2026



FACULTY OF APPLIED SCIENCES
UNIVERSITY OF WEST BOHEMIA
IN PILSEN

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Master's Thesis

Automatic Recognition of Parcel Numbers from Historical Maps

Bc. Vladimír Holý

Thesis advisor
Prof. Ing. Pavel Král, Ph.D.

PILSEN, CZECH REPUBLIC 2026

© 2026 Vladimír Holý

All rights reserved. No part of this document may be reproduced or transmitted in any form by any means, electronic or mechanical including photocopying, recording or by any information storage and retrieval system, without permission from the copyright holder(s) in writing.

Citation in the bibliography/reference list:
HOLÝ, Vladimír. *Automatic Recognition of Parcel Numbers from Historical Maps*. Pilsen, Czech Republic, 2026. Master's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor Prof. Ing. Pavel Král, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2025/2026

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Vladimír HOLÝ**
Osobní číslo:	**A23N0094P**
Studijní program:	**N0613A140040 Softwarové a informační systémy**
Téma práce:	**Automatické rozpoznávání parcelních čísel z historických map**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s historickými mapami stabilního katastru včetně parcelních protokolů.
2. Prostudujte relevantní metody detekce objektů a vhodné metody rozpoznávání textu založené na neuronových sítích.
3. Na základě předchozí studie navrhněte a implementujte prototyp systému pro automatickou detekci a rozpoznávání parcelních čísel na historických mapách stabilního katastru s využitím parcelních protokolů. V systému integrujte dvě různé detekční metody a dvě metody rozpoznávání textu.
4. Funkčnost prototypu otestujte na vytvořené datové sadě.
5. Výsledky diskutujte a navrhněte další možná rozšíření.

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

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**doc. Ing. Pavel Král, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání diplomové práce:	**10. září 2025**
Termín odevzdání diplomové práce:	**14. května 2026**

L.S.

doc. Ing. Miloš Železný, Ph.D. děkan	**doc. Ing. Přemysl Brada, MSc., Ph.D.** vedoucí katedry

V Plzni dne 30. září 2025

Declaration

I hereby declare that this Master's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. During the preparation of this thesis, I utilized the Gemini 3 PRO artificial intelligence tool for linguistic proof-reading and code formatting. Following use of this tool, I carefully reviewed and edited the content as necessary and assume full responsibility for the final content of the work. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

In Pilsen, on 3 May 2026

. . . . . .

Vladimír Holý

The names of products, technologies, services, applications, companies, etc. used in the text may be trademarks or registered trademarks of their respective owners.

Abstract

This master's thesis focuses on the automation of parcel number recognition from historical cadastral maps, which are part of the Stable Cadastre. The objective of the thesis is to develop a system based on neural networks capable of detecting parcel numbers from surveying records and recognizing text from written records. The primary challenge of the work lies in the age and condition of the archival maps and documents. The age of these documents is apparent from various stains and the paper's yellowing. To address this issue, two object detection methods and four text recognition methods were utilized. Despite the challenges associated with the age of the documents, I achieved very high accuracy and a low error rate. The final output is a functional system for automated processing of cadastral maps, facilitating work with data from the historical Stable Cadastre in the Czech Republic.

Abstrakt

Tato diplomová práce se zabývá automatizací rozpoznávání parcelních čísel z historických katastrálních map, které jsou součástí stabilního katastru. Cílem práce je vytvořit systém založený na neuronových sítích, který bude schopen detekovat parcelní čísla z měřických operátů a rozpoznávat text z písemných operátů. Hlavní výzvou celé práce je stáří archivních map a dokumentů. Stáří těchto dokumentů je patrné z různých skvrn a zažloutnutí papíru. Při řešení této výzvy jsem využil dvě metody pro detekci objektů a čtyři metody pro rozpoznávání textu. I přes problémy spojené se stářím dokumentů se mi povedlo dosáhnout velmi vysoké přesnosti a nízké chybovosti. Výstupem práce je funkční systém pro automatizaci zpracování katastrálních map, který usnadňuje práci s daty z historického stabilního katastru v České republice.

Keywords

Handwritten Text Recognition • Historical documents • Object detection • Stable Cadastre • Vision Language Models

Acknowledgement

I want to express my sincere gratitude to my thesis supervisor, Prof. Ing. Pavel Král, Ph.D., for his valuable advice, professional guidance, and the time he dedicated to me throughout the process. My thanks also go to Český úřad zeměměřický a katastrální (ČÚZK) for providing essential data resources and for their assistance with data annotation. Furthermore, I would like to thank my family, my girlfriend, and my friends for their support and patience throughout my studies.

Vladimír Holý,
the template author
(May 2026)

Contents

1 Introduction **1**

2 Specifics of Historical Maps and the Stable Cadastre **2**
2.1 Origin and Purpose of the Stable Cadastre 2
2.2 Surveying Records 3
2.3 Parcel Marking 3
2.4 Written Records 5

3 Methods **7**
3.1 Neural Networks 7
3.1.1 Artificial Neurons 7
3.1.2 Architecture 8
3.1.3 The Training Process 9
3.1.4 Generalization and Data Splitting 10
3.1.5 Hyperparameters 11
3.2 Methods for Object Detection 12
3.2.1 Convolutional Neural Networks 12
3.2.2 Detection Methods 15
3.2.3 Segmentation Methods 16
3.2.4 Candidate Models 17
3.3 Text Recognition Methods 18
3.3.1 CNN + RNN + CTC Models 18
3.3.2 CNN + Attention Models 19
3.3.3 Models Based on the Transformer Architecture 19
3.3.4 Large Vision-Language Models 19
3.3.5 Candidate Models 20

4 Proposed Solution **23**
4.1 System architecture 23
4.1.1 Cadastral Map Pipeline 25
4.1.2 Written Protocol Pipeline 25

4.2 Data Preparation and Annotation 26
4.2.1 Development of Software Tools for Annotation 26
4.2.2 Source Data 33
4.2.3 Annotation Process 33
4.2.4 Annotation Speedup 35
4.2.5 Dataset Splitting 35
4.3 Evaluation Metrics 35
4.3.1 Detection Metrics 35
4.3.2 OCR Metrics 37
4.4 Selection of Specific Models 38
4.4.1 Detection and Segmentation Models 38
4.4.2 OCR Models 40
4.5 Implementation of Experiments and Training 41
4.5.1 Technical Equipment 41
4.5.2 Implementation of Detection Models 41
4.5.3 Custom Evaluation Script 43
4.5.4 Implementation of OCR Models 43
4.6 Final Results and Comparison 50
4.6.1 Detection Evaluation 50
4.6.2 OCR Evaluation 51
4.7 Discussion and Error Analysis 54
4.7.1 Analysis of Model Limits and Errors 54
4.7.2 Possible Improvements 56

5 Conclusion **57**

Bibliography **58**

List of Figures **65**

List of Tables **67**

List of Abbreviations **68**

User Manual **69**

Introduction 1

The Stable Cadastre, originating in the 19th century, was a groundbreaking achievement for its time. It enabled the mapping of the entire territory of the Habsburg Monarchy, covering areas of modern-day states such as the Czech Republic, Austria, Slovenia, and Poland. Today, the entire modern system of land and real estate records in the Czech Republic and beyond is built upon these foundations. These historical records contain vast amounts of data; however, although a large portion has been converted into digital form through raster scans, the data remains non-machine-readable. This poses a significant challenge for historians and authorities working with the records, as manual searching is extremely demanding and time-consuming.

This thesis addresses this issue by attempting to advance the digitization of the Stable Cadastre a step further. My objective is to develop a system capable of automating the recognition of parcel numbers from cadastral maps. Since the Stable Cadastre consists not only of maps but also of written records containing additional information about individual plots, the work also focuses on automating the extraction of data from these documents.

The primary challenge in this automation process is undoubtedly the age of the documents. Over such a long period, the materials have degraded, the paper has yellowed, and the ink has faded. A major challenge also lies in the fact that the records consist of handwritten text.

The structure of the work is as follows: Chapter 2 provides a thorough introduction to the Stable Cadastre. In Chapter 3, I examine neural networks, their fundamental principles, and their operation. Based on this study, I perform an analysis of an ideal system based on neural networks for detecting parcel numbers and for recognizing text within written records. In Chapter 4, I select suitable methods based on the preceding analysis, which are then implemented, tested, and evaluated. Finally, Chapter 5 summarizes the findings and outcomes of the entire thesis.

The ultimate output of this work is a functional and tested system for the automated processing of cadastral maps. This system is designed to be extendable, thereby facilitating further work with data from the historical Stable Cadastre in the Czech Republic.

Specifics of Historical Maps and the Stable Cadastre

2

A key prerequisite for the realization of this thesis was becoming familiar with the provided data, upon which the entire thesis is developed. This data was provided by Český úřad zeměměřický a katastrální (ČÚZK) [Čes26a]. This office is the central body of the state administration for surveying and the real estate cadastre in the Czech Republic, which also includes the mapping of the entire national territory and geodesy. ČÚZK provided the data of the Stable Cadastre for this work.

2.1 Origin and Purpose of the Stable Cadastre

The Stable Cadastre [Čes24] was established in 1817 by Emperor Francis I. Its primary goal was to create a fair and permanent register to serve as a basis for tax collection. The name stems from the requirement that it should be a permanent, or "stable", cadastre. The cadastre consisted of three main parts:

1. **Cadastral Surveying Records**
   The surveying record was a map illustrating all plots of land and their assigned numbers. Land plots were entered into the record after geometric surveying.

2. **Cadastral Written Records**
   The written record complemented the surveying record and linked the plots (specifically their parcel numbers) with additional information about the land. Examples include the name of the parcel owner or the exact area.

3. **Land Valuation Records**
   This document classified land and was used to calculate net yield, from which the tax was subsequently determined.

For the purposes of this thesis, only the first two parts - the surveying and written records - are relevant.

2.2 Surveying Records

As previously mentioned, surveying records contain graphic representations of all plots. These plots used various color codings to identify the type of land. For example, brown was used for roads, blue for water bodies, and light green for pastures. This color coding followed a uniform, fixed legend, which can be found in Figure 2.1. Figure 2.2 shows an example of a surveying record. These are very old maps, created around 1840. The age of these documents is apparent from various stains and the yellowing of the paper.

2.3 Parcel Marking

Based on the goals of this thesis, attention is focused primarily on parcel numbers. Each parcel is assigned its own unique number. These numbers are divided into two groups, each with its own independent numbering. The first group consists of building parcels, whose numbers are written in black on the surveying record. The second group consists of land parcels marked with red numbers. The distinction between these two types of numbering is clearly visible in Figure 2.3.

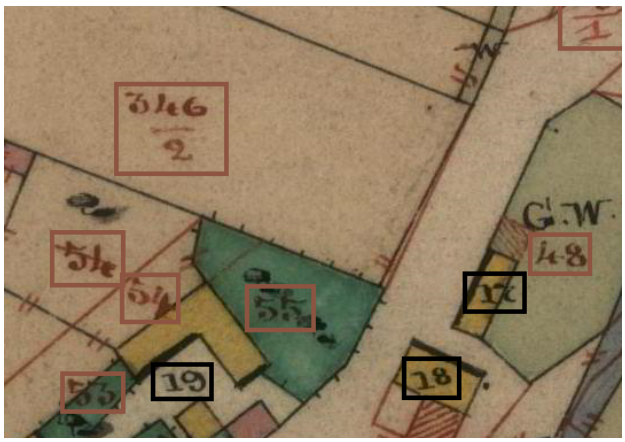


Figure 2.3: Visualization of different parcel number types. Building parcels are highlighted in black boxes, while land parcels are highlighted in red boxes.

All numbers were handwritten, and a different person completed each surveying record. From this perspective, a uniform style and shape of the numbers are not guaranteed, which can make reading more challenging. Another issue is the rotation of the numbers. Although most numbers are written horizontally, some can be

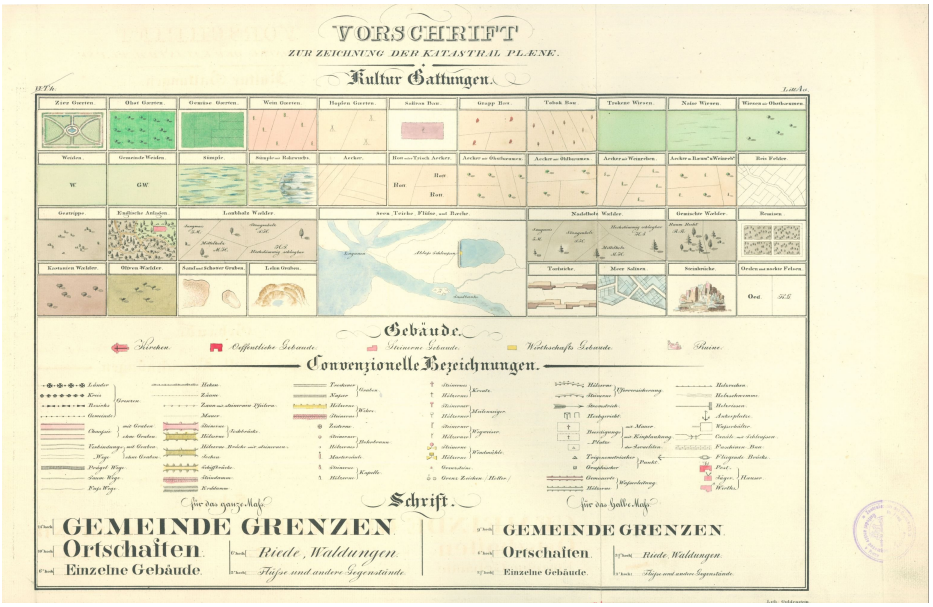


Figure 2.1: Color coding legend for the surveying record. Adapted from [Čes26b].



Figure 2.2: Example of a surveying record from the dataset.

rotated by up to 90 degrees, most commonly in the case of roads. Further difficulty arises when many small parcels are located next to each other, a phenomenon most frequent in built-up areas. In such instances, the numbers are written very close together, making them difficult to locate and distinguish. Added complexity comes from the crossing out of parcel numbers. As cadastral maps were updated over time, parcels became obsolete, and their numbers were crossed out. The opposite problem is parcel division, which was handled by adding a slash and a sub-number to each parcel (e.g., "143/1" and "143/2"). It is important to take these issues into account and attempt to eliminate them as the work progresses.

2.4 Written Records

Written records, or parcel protocols, are tables that map parcel numbers to further information about the plot. Written records are divided into the same groups as parcel numbers: building parcel records and land parcel records. These records differ in structure and the amount of information provided. The differences can be observed in Figure 2.4 and 2.5. Typically, these tables contain twenty rows and a varying number of columns depending on the type. Land parcel protocols contain a total of nineteen columns. The most important information in the protocol is the parcel number. Additionally, the record contains information such as the owner's name, their occupation, the land class, its area, and more. Building parcel protocols have fewer columns, only sixteen.

Same as parcel maps, written records include the age of the paper, stains, and the fact that the text is handwritten. Furthermore, the text is crossed out and supplemented with additional information. The entire table spans two pages, so the scans include the spine binding in the center, and in some cases the pages are not horizontally aligned. Due to the binding, the pages contain a curvature, meaning the horizontal rows of the table are not straight but curved.

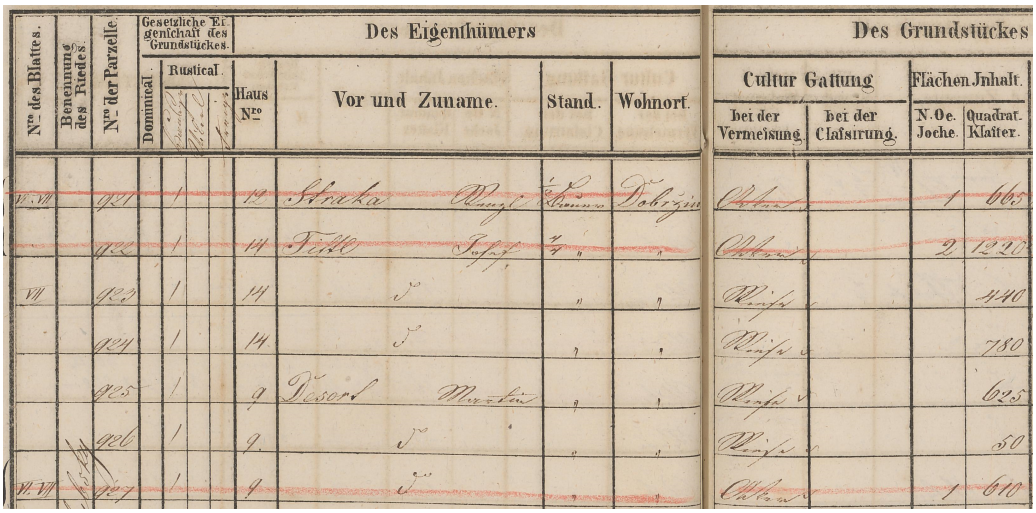


Figure 2.4: Example of a land written record.

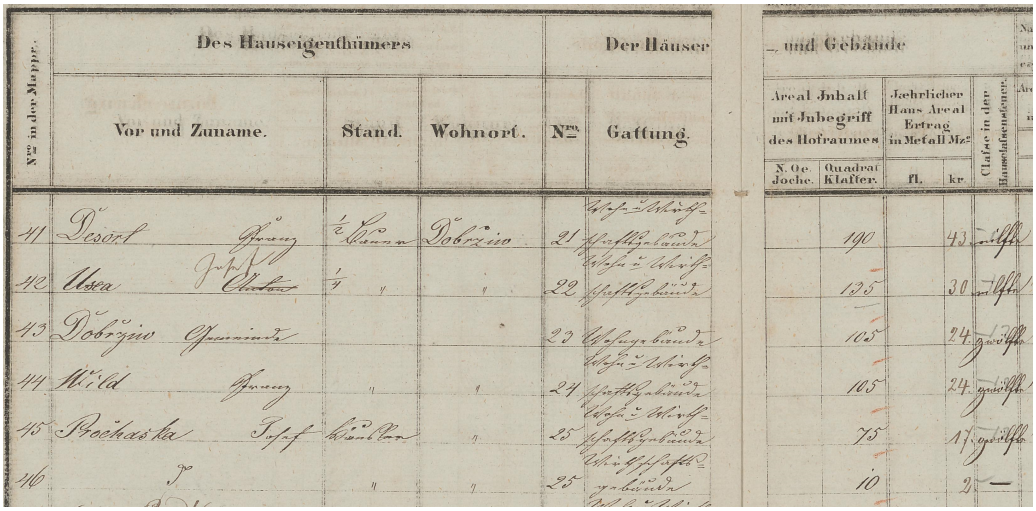


Figure 2.5: Example of a building written record.

Methods 3

In this chapter, I will introduce neural networks and methods for object detection and text recognition.

3.1 Neural Networks

Neural networks [IBM24a] represent one of the primary methods of machine learning. These are complex computational models inspired by the human central nervous system, specifically the brain. Our brain consists of a complex network of interconnected cells called neurons. Individual neurons receive electrical signals from other neurons, process them, and transmit a signal to other neurons. This interconnection enables memory, learning, and the ability to solve complex tasks. In computer science, the neuron is replaced by a simple computational unit called an **artificial neuron**.

3.1.1 Artificial Neurons

The artificial neuron [Rob24] is the fundamental building block of a neural network. The first artificial neurons were developed as early as 1943. An artificial neuron receives an input vector at its input. This vector consists of n elements (signals), which, for example, in the context of image processing, can be represented as individual pixel values. During the processing of the input vector, the so-called **internal potential** z [LJY24] is calculated within the neuron, as follows:

$$z = b + \sum_{i=1}^{n} (w_i \cdot x_i) \tag{3.1}$$

Each input is assigned a corresponding weight w, which represents the importance of that input; the weight thus simulates the input's strength. The internal potential is calculated as the sum of the products of all inputs and their weights. Furthermore, a **bias** (or threshold) is added to this sum, determining the neuron's activation threshold. The resulting value of the internal potential is then transformed

by an **activation function**, and its output is sent to subsequent neurons. During neural network training, the individual values of the input weights and the neuron's bias are determined. A diagram of an artificial neuron can be seen in Figure 3.1.

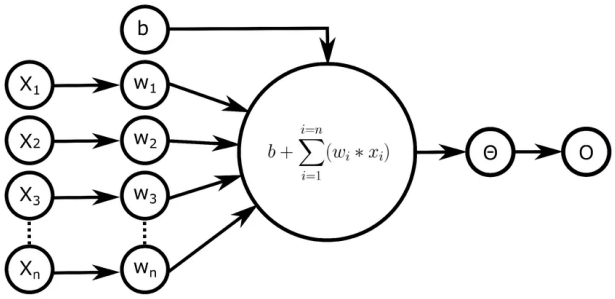


Figure 3.1: Diagram of an artificial neuron. Adapted from [Wei+22].

3.1.2 Architecture

Because a simple interconnection of neurons would only result in a linear combination of inputs, meaning the entire neural network, regardless of its depth, would behave only as a linear transformation, it was necessary to introduce elements of non-linearity. Therefore, **activation functions** [LJY24] were introduced, enabling neural networks to solve complex and sophisticated tasks. There are many activation functions; for example, the **Rectified Linear Unit (ReLU)** function [Ngu23], used by convolutional neural networks, can be demonstrated. This function passes all positive values of the internal potential while setting negative values to zero. Mathematically, it can be expressed as:

$$f(z) = \max(0, z) \tag{3.2}$$

For the representation of a sophisticated object, it is important to connect neurons into larger units, or **layers** [Ama24]. A neural network can consist of several sub-networks, but the basic form is a three-layer network, as shown in Figure 3.2.

1. **Input layer**
   The input layer processes the input data. This layer does not perform any calculations; it merely receives the input and passes it to the next layer.

2. **Hidden layer**
   In this layer, complex non-linear calculations occur; it is essentially the "brain" of the entire neural network. Neural networks can utilize more than one hidden layer, in which case we refer to it as deep neural network. Individual hidden layers gradually work on a complex task.

3. **Output layer**
   The final layer in the neural network provides the final prediction. The output format of this layer depends on the specific problem being solved. If the layer consists of a single neuron, the model performs regression, such as predicting the precise coordinates of a bounding box. If the layer contains multiple neurons, it is typically used for classification. Each neuron represents a specific class, and the output usually represents the probability distribution across these categories.

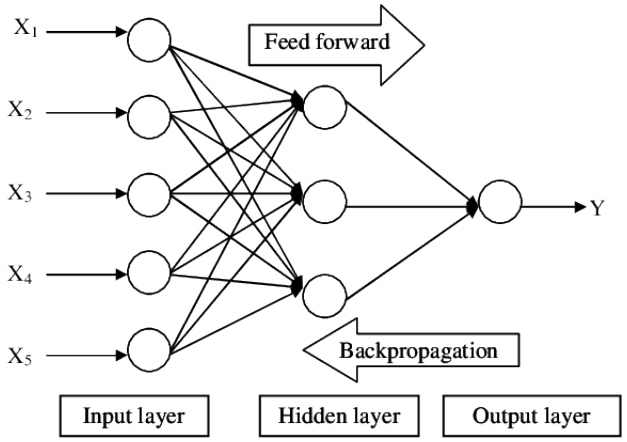


Figure 3.2: Basic layers of a neural network. Adapted from [Hak+11].

3.1.3 The Training Process

During neural network training, the individual values of the input weights and the neuron's bias are determined. Before the training itself, these parameters are usually set randomly. The goal of learning is to find the optimal parameter configuration (weights and biases) that ensures the model returns correct results, i.e., that the difference between the network's prediction and reality is minimized. The learning process consists of several consecutive steps:

1. **Forward propagation**
   In the first phase of the training, input data passes through all layers of the network to the output, and a prediction is determined. In the early stages of training, these predictions are usually very inaccurate.

2. **Loss function**
   To evaluate the accuracy of the prediction, a **loss function** [IBM24b] is introduced. It is a mathematical function that compares the network's prediction with the actual annotated value, known as the **ground truth**. The result is a

scalar value representing the degree of prediction error. The implementation of the loss function depends on the specific problem. Examples include *Intersection over Union Loss* [Ros16] for comparing bounding box overlaps, *Mean Squared Error* for mean squared error, or *Cross-Entropy Loss* for comparing individual characters.

3. **Backward propagation** [Bri26]
   In this phase, it is crucial to determine the contribution of individual neurons to the error. The entire neural network is traversed from back to front, neuron by neuron, and for each neuron, a **gradient** is computed with respect to its weight w. It is calculated as the partial derivative of the loss function:

$$\frac{\partial L}{\partial w} \tag{3.3}$$

   The gradient represents the direction of the individual neuron's error. If the gradient is positive, increasing the weight value would increase the error; if the gradient is negative, increasing the value would decrease the error.

4. **Gradient Descent**
   In this final training phase, the physical adjustment of the neuron weights occurs. The most fundamental and well-known algorithm is Gradient Descent [Goo24a]. The algorithm's principle is to adjust the weights in the direction opposite the calculated gradient. The new weight is calculated as:

$$w_{new} = w_{old} - \alpha \cdot \frac{\partial L}{\partial w} \tag{3.4}$$

   A crucial element in this equation is the parameter α, which represents the **learning rate**. This parameter determines how quickly and effectively the neural network will learn. If the learning rate is too low, the network learns extremely slowly. However, if the learning rate is too high, the neuron may overstep its optimal weight value, causing the error to increase.

These steps are repeated in successive iterations to minimize the Loss Function and find the optimal weights for all neurons.

3.1.4 Generalization and Data Splitting

The goal of training a neural network is not to memorize the training data perfectly, but to **generalize** so that the model can be used on data it has not seen before and still return correct values. The problem where a neural network learns data too precisely is called **overfitting**. It is a state in which the model perfectly recognizes the data

it was trained on, but fails on completely new data. To ensure the generalizability of the model and mitigate overfitting, the dataset is categorically divided into three groups [Goo26]:

1. **Train set**
   This part is used for the model training. The neural network calculates gradients on this data and adjusts the individual neuron weights accordingly.

2. **Validation set**
   This part is crucial in preventing overfitting. It is used to calculate the **validation loss** using the loss function. The training loss continuously monitors progress to ensure the network is learning correctly and not over-optimizing. Through continuous validation, it is possible to estimate when the model is ideally trained and to stop training accordingly. This process of early termination is called **Early Stopping** [APX26]. Furthermore, this part of the dataset is used for adjusting hyperparameters.

3. **Test set**
   This part is fully separated from the previous two and must not be seen by the model during training under any circumstances. It serves only for the final and independent evaluation of the trained model.

3.1.5 Hyperparameters

To correctly understand the learning process, it is important to distinguish between two types of variables: parameters and hyperparameters. Both significantly affect the model's behavior, but there is a fundamental difference:

1. **Model Parameters**
   These variables include the previously mentioned biases and the neurons' weights. These are internal network variables that cannot be set manually. The network calculates and adjusts these values itself during training.

2. **Hyperparameters**
   In contrast, hyperparameters are external configuration values set before training. These values directly influence the training process and the resulting quality of the trained model. Essentially, they determine how and how fast the network learns.

If hyperparameters are set to optimal values, the training and the model itself should achieve the best results. The process of finding these optimal values is called **Hyperparameter tuning** [IBM24c]. This process is very time-consuming

and hardware-intensive, making it virtually impossible to test all possible combinations. Therefore, several principles are used in practice, such as **Grid Search** [IBM24c], which determines suitable value variants for individual hyperparameters and tests all possible combinations. Another example is **Random Search** [IBM24c], where acceptable minimum and maximum boundaries for hyperparameter values are defined, and the algorithm then randomly combines and tests various values.

Common Hyperparameters

There are many hyperparameters, but I will focus only on the most important ones:

1. **Learning Rate** [Gee24a]
   One of the most important hyperparameters is explained in the previous section.

2. **Batch size** [Goo24c]
   The number of input data samples that pass through the network at once before a single weight update step. If the batch size is too small, the model can be easily influenced by data noise; if it is too large, the model may become "lazy" and lose its ability to generalize. This size is primarily limited by hardware, specifically the amount of VRAM.

3. **Epochs number** [Goo24d]
   Determines how many times the network passes through the entire training dataset during training. Too few epochs may result in an undertrained model, while overfitting may cause the training to run unnecessarily long.

3.2 Methods for Object Detection

Detection and segmentation methods are used to detect and segment objects in images. In object detection, it is determined what is in the image and where it is located. So-called bounding boxes are used for object detection. A bounding box is a rectangle that encloses the identified object and is usually defined by coordinates and dimensions. In contrast, segmentation provides greater detail. Like detection, it searches for and delineates objects, but with pixel-level precision. In segmentation [Nit24], every individual pixel is checked, so the shape of the object is precisely traced.

3.2.1 Convolutional Neural Networks

Convolutional Neural Network (CNN) [Pil24] is the standard for image analysis and serve as the technological foundation for many detectors and segmentation

models. The CNN architecture draws inspiration from the human visual cortex, where individual neurons respond only to specific regions of the visual field.

Convolutional Layer and Filters

Since it is practically impossible for an entire image to be loaded into a neural network at once, **mathematical convolution** [Pil24] is used as the fundamental operation of this network. Instead of every neuron being connected to all pixels of the image simultaneously, a so-called convolutional filter, or kernel, is used. It is a small matrix, typically 3x3 or 5x5 pixels, that always covers a small region of the image. This filter gradually slides across the entire image, and at each position, it computes a dot product between the observed pixel values and the network weights. By progressively analyzing the entire image, a new matrix, called a Feature Map, is created. This process is shown in Figure 3.3. The network contains several convolutional layers.

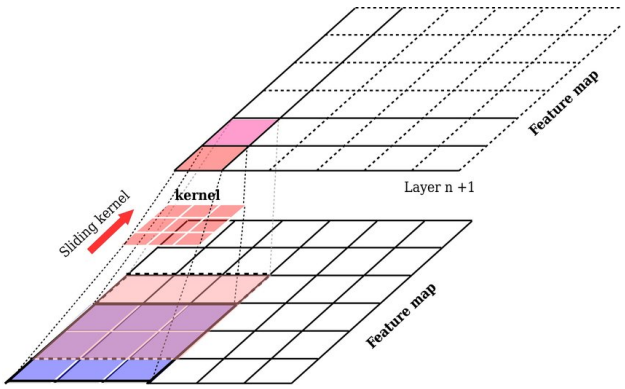


Figure 3.3: Convolutional filter. Adapted from [Sha+18].

Pooling Layer

In addition to convolutional layers, convolutional networks also contain **pooling layers** [Gee24b]. This layer gradually reduces the image's dimensions, specifically its width and height. The main task of this layer is to reduce the number of parameters and computational complexity. An example is the **Max Pooling** method [Pil24], which is among the most commonly used. Similar to the convolutional layer, it slides through the image in small segments, for example, 2x2 pixels. To achieve a reduction, only the maximum value in the matrix is selected for further processing. For comparison, the **Average Pooling** method, which computes the average, is shown in Figure 3.4.

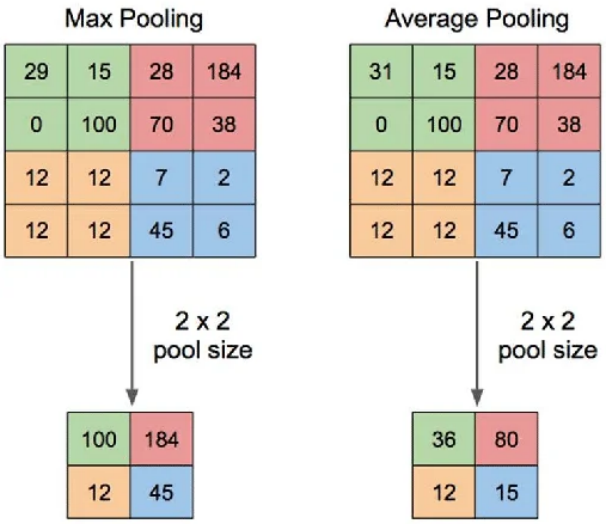


Figure 3.4: Pooling methods, Max Pooling (right) and Average Pooling (left). Adapted from [ON15].

Hierarchy

A convolutional network is composed of several convolutional and pooling layers that alternate sequentially. These layers can be further assembled into more complex structures [Pil24]:

1. **Shallow layers**
   In this initial phase, the neural network begins to detect basic visual features, such as vertical and horizontal edges. The activation functions here serve to filter these basic signals and prepare them for more complex combinations.

2. **Intermediate layers**
   As processing continues, the network begins to detect more complex patterns, such as curves, line intersections, or various blobs. This transition from simple lines to complex shapes is enabled by the nonlinearity introduced between layers, which allows the network to approximate higher-order geometric features.

3. **Deep layers**
   In the final, deep layers, the convolutional neural network assembles these patterns into complete objects.

The output of a convolutional neural network serves merely as a **Backbone** [Ult24a], or the foundation for object detection. The task is to prepare the image data for further processing. The data is passed to another part called the **Head**, which

processes the features extracted by the convolutional network to perform specific tasks, such as image classification, bounding box detection, or segmentation.

3.2.2 Detection Methods

As described in the previous section, convolutional networks can extract visual features from images and classify these features. In this section, I will focus only on detection, while segmentation will be introduced later. Currently, there are two main groups of detectors, both based on convolutional networks.

Two-Stage Detectors

Two-Stage detectors [Gee24d] are detection methods built on convolutional networks. Historically, this is the older group where detection takes place in two steps:

1. **Region Proposal**
   In this first step, an initial detection of objects is performed with low precision and confidence. The result of this step is a large number of imprecise bounding boxes.

2. **Bounding Box Regression**
   In the second step, all bounding boxes are gradually fed into another network and analyzed one by one. The goal is to determine with high precision whether the sought object is in the box. If so, the box is refined as well.

The advantage of Two-Stage detectors is their high detection accuracy. Conversely, the disadvantages include slow processing speed, high complexity, and high resource intensity. The most significant representative of this method is the R-CNN and its improved, faster version, Faster R-CNN [Gee24c].

One-Stage Detectors

The second group consists of **One-Stage detectors** [Jor18], which simplify previous methods and rely on a single phase. Because the entire process is unified into a single step, these architectures are much faster and more efficient. Although one-stage detectors historically showed lower accuracy than two-stage ones, today their accuracy has already equalized [Nit+21]. Instead of generating and subsequently filtering multiple boxes, one-stage detectors divide the image into a regular grid. Subsequently, they pass through individual cells, each of which is responsible for detecting objects. A cell is responsible only for those objects whose center lies within it. This approach is so fast that it enables real-time detection. The most significant

representatives of this group are the **You Only Look Once (YOLO)** [Ult24b] algorithms; the **Single Shot MultiBox Detector (SSD)** [Liu+16] or **Efficient and Accurate Scene Text detector (EAST)** [Zho+17] algorithm also belongs here.

3.2.3 Segmentation Methods

The main goal of segmentation methods [Ros22] is to find objects and assign them to the correct category. The segmentors return individual pixels. The output of segmentation models is therefore a mask that precisely traces the object's shape. These methods can be divided into three groups:

1. **Semantic Segmentation**
   This group can classify individual pixels into specific classes but cannot distinguish between individual objects within a single class. For instance, when searching for parcel numbers on a map, this algorithm would return a single mask containing pixels belonging to parcel numbers. Complex post-processing would then be required to detect individual numbers. A well-known example of this group is the U-Net architecture [Pne26].

2. **Instance Segmentation**
   This method of segmentation addresses the deficiency of the previous group by adding the identification of individual object instances. With these segmentation models, it is possible to distinguish which pixels belong, for example, to parcel number 15 and which to number 27. Representatives include the Mask R-CNN [He+17].

3. **Transformer-Based Methods**
   The most modern way of segmenting objects is transformer-based methods [Che+21], which unify semantic and instance segmentation. Sometimes these methods are also called panoptic segmentation because their outputs are panoptic masks. This output assigns each pixel a semantic class, but also provides information about which specific instance of that class it belongs to. These methods are much more complex because they perceive the global context of the entire image. As a result, the final segmentation should be more precise. The disadvantages are greater complexity and the need for a large training set. These methods are based on the attention mechanism, which I will explain in the following section. An example of this group is Mask2Former.

An example of the output of these groups can be seen in Figure 3.5.

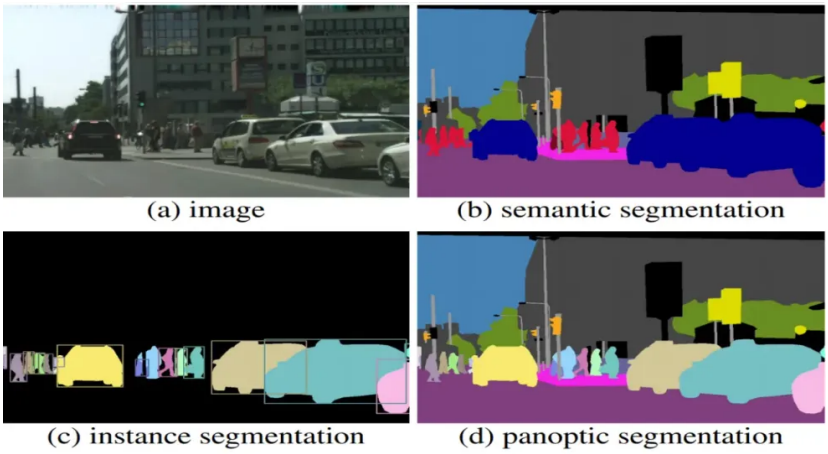


Figure 3.5: Semantic, Instance, and Panoptic segmentation. Adapted from [Lab24].

3.2.3.1 Attention Mechanism

While classical convolutional networks perceive only a small part of the image using a sliding filter, which prevents them from quickly understanding the broader image context, the attention mechanism [Ala24] can observe the entire scene's context globally in a single step. The attention mechanism is not limited to an isolated analysis of local pixels but interprets their mutual global context.

3.2.4 Candidate Models

Based on the literature review and analysis, I have selected a total of four models as potential candidates for future implementation:

1. **Faster R-CNN**
   This model [Gee24c] is a standard representative of the two-stage detector group. The model excels in high precision but comes at the cost of high computational intensity, significant complexity, and slow processing time per image.

2. **SSD**
   SSD [Liu+16] belongs, as the name suggests, to the family of one-stage detectors. Its main innovation was the ability to detect objects of various sizes in a single pass of the image using convolutional networks. In its time, the model was revolutionary, but today it represents a compromise between precision and speed. Currently, it has been surpassed by more modern architectures.

3. **YOLO**
   The revolutionary one-stage YOLO model [Ult24b], which is currently among

the most widely used in this category. The YOLO model family enjoys immense community support and is still actively developed. These models are extremely fast and very precise.

4. **EAST**
   Unlike the previous models, which are primarily intended for general object detection, the EAST architecture [Zho+17] (Efficient and Accurate Scene Text Detector) is designed purely for text detection. Its greatest advantage is its ability to detect text in any orientation. Instead of classical horizontal bounding boxes, EAST returns **oriented bounding boxes**. EAST also belongs to the one-stage detector group.

3.3 Text Recognition Methods

Optical Character Recognition (OCR) [IBM24d] are used for recognizing text in images. This technology can extract text from images, photographs, or scanned documents and convert it to digital form. These methods can be applied to reading both parcel numbers and records from written protocols. In both cases, this involves a specific category of OCR known as **Handwritten Text Recognition (HTR)**. These models can be divided into three main groups.

3.3.1 CNN + RNN + CTC Models

These models [SBY15] currently represent a historical standard and foundation. Text reading in this architecture is divided into three sequential steps:

1. **CNN**
   In the first step, the input is processed using a convolutional network to identify basic graphic symbols and shapes.

2. **Recurrent Neural Network (RNN)**
   The symbols from the convolutional network are subsequently passed to a RNN. This network has its own internal memory, which allows it to maintain a certain context of the entire text and identify individual letters accordingly. While the CNN in this architecture performs visual feature extraction (metaphorically acting as the eyes), the RNN interprets sequential data and attempts to convert symbols from the CNN into actual text characters based on the text context.

3. **Connectionist Temporal Classification (CTC)**
   CTC [Hug26a] serves as a post-processing after the RNN layer. This algorithm

removes duplicate and blank characters and attempts to create a clean final text string.

From an architectural standpoint, this method is very simple, which makes it very fast and requires only a small training set. As a result of this simplicity, these models achieve lower accuracy. A representative of this method is the Convolutional Recurrent Neural Network (CRNN) model [SBY15].

3.3.2 CNN + Attention Models

These models [Wan+24] are based on the previous ones but replace the RNN and CTC components with an attention mechanism. The input phase proceeds identically to the previous type, where a convolutional neural network extracts basic graphic symbols and shapes. Subsequently, the data travels to an attention-based decoder. This approach provides the models with a broader global context over the entire text and allows them to achieve better results than the previous method. Primarily, the models offer better adaptation to irregular handwriting and to long-term letter dependencies. In contrast, however, it requires a larger training set.

3.3.3 Models Based on the Transformer Architecture

These models represents the most modern and currently most powerful approach to text recognition. These methods completely abandon convolutional and recurrent networks and are built entirely on the attention mechanism. Consequently, they analyze the entire text context simultaneously and can detect complex dependencies within it. As a result, they belong to the most accurate group of text recognition methods. A disadvantage is that these are large models requiring an enormous amount of data for training from scratch, on the scale of millions of samples. Because of this, pre-trained models are typically used in practice and are only fine-tuned. Examples include TrOCR [Li+21], or SATRN [Lee+20].

3.3.4 Large Vision-Language Models

Although the Transformers mentioned in the previous section achieve excellent results, they still behave internally as specialized OCR tools. The current technological peak in this field is represented by **Vision-Language Models (VLM)** [NVI24]. These models are based on Transformers, but they are much more complex. The VLM architecture combines two parts: a **Vision Encoder (VE)** and a **Large Language Model (LLM)**. The VE serves as the "eyes" of the architecture. This encoder processes the input image and passes it to the LLM via a Projector. The LLM serves as the "brain"; its input consists not only of the VE's output but also a user prompt

describing what to do. An example of an instruction could be a request to locate a parcel number in an image. This model combines these two elements and generates an output. The entire process is illustrated in Figure 3.6. VLM models require an extreme amount of data for training, and as such, they are trained on terabytes of text from across the entire internet. Examples include the Hunyuan [Ten25] and QWEN [Qwe25] models.

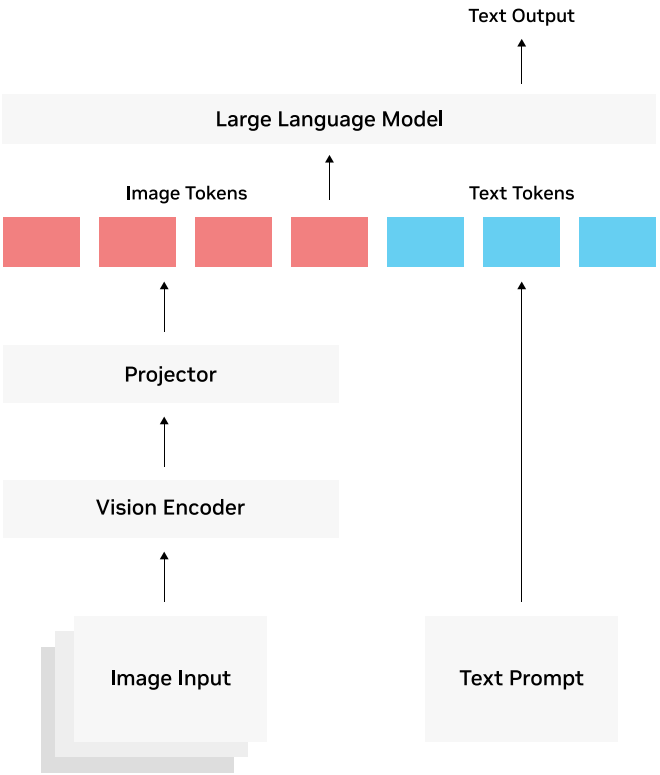


Figure 3.6: Architecture diagram of a multimodal model combining visual and text inputs to generate responses. Adapted from [NVI24].

3.3.5 Candidate Models

Based on a literature review and a analysis of current state-of-the-art architectures, six models have been shortlisted as potential candidates for implementation:

1. **Kraken**
   This model [Kie24] belongs to the group of architectures based on CNN, RNN, and CTC. Kraken is a highly recognized open-source HTR project. Its focus is directly on handwritten texts. During its development, Kraken was trained on a large number of historical documents, which is an advantage for this work. Another advantage is that this model requires little computational power.

2. **TrOCR**
   TrOCR [Li+21] is the sole representative of the Transformers group and is a pure representative of this group. This model from Microsoft was among the first to fully abandon recurrent models and transition entirely to an Encoder-Decoder architecture. The Encoder in this model is a Vision Transformer, which slices the image into small squares and sends them to a language Transformer that reads the text. This model should achieve much higher precision than the previously mentioned model, though it also possesses higher computational requirements.

3. **GOT-OCR 2.0**
   This model [Wei+24] falls into a hybrid category between the attention mechanism group and large VLM. Similar to TrOCR, the image is first processed using a Vision Transformer and a lightweight VLM, specifically Qwen-0.5B, and then processed further. This model can thus be classified as a "lightweight VLM." Its goal was to optimize VLM for speed and efficiency. It is a model that, unlike other large VLM models, does not accept any prompt as input and operates as pure OCR. Nevertheless, it retains the ability to understand complex text semantics.

4. **QWEN**
   QWEN [Qwe25] is a VLM model from Alibaba Cloud. This model is expected to excel in general visual intelligence. Consequently, the model should have a strong understanding of what in the image is text and what is, for example, just a stain. Specifically, the Qwen3-VL-2B-Instruct model would be used. The QWEN family also includes larger models, such as Qwen3-VL-8B-Instruct, which are already too large and would require immense computing power for training. Even training the smaller model will be computationally intensive, but it should yield very precise results.

5. **Hunyuan**
   Hunyuan [Ten25] also specializes in general visual intelligence. In my case, I will specifically use the HunyuanOCR model, which is specialized for text detection and recognition. Due to this specialization, it should likely be the most accurate model. Like the previous model, however, it will be very computationally intensive.

6. **DeepSeek**
   The final representative of large VLM models is DeepSeek [Dee25]. Unlike its predecessors, this model utilizes a Mixture-of-Experts architecture. This approach allows it to activate only the part of its neurons necessary for each task. In practice, this means that even though it is a massive VLM model, it is

highly optimized to reduce computational intensity. DeepSeek, like the previous large VLM models (QWEN and Hunyuan), should be able to understand and accurately read text comprehensively.

Proposed Solution 4

This chapter first propose the system architecture, data preparation and annotation, the selection of the most suitable models, and the subsequent training process. At the end of this chapter, the results of the trained models are evaluated.

4.1 System architecture

Based on the theoretical findings, I propose and implement a system for automated data processing from the Stable Cadastre. The complete system architecture is depicted in Figure 4.1. The system is structured into three distinct pipelines: training, the cadastral map, and the written protocol.

Training Pipeline

The first important feature of the system is the training of models for text detection and recognition. Before the actual training, it was important to prepare the training data. For this purpose, the data must be annotated. After the data is prepared, it must be split into three groups: train, validation, and test. When the dataset is ready, the training process can begin. It must start with hyperparameter tuning, which is crucial for achieving the best possible results. After finding the optimal hyperparameters, the actual training can begin. The models are then evaluated, and the best-performing model is selected for use in the subsequent pipelines.

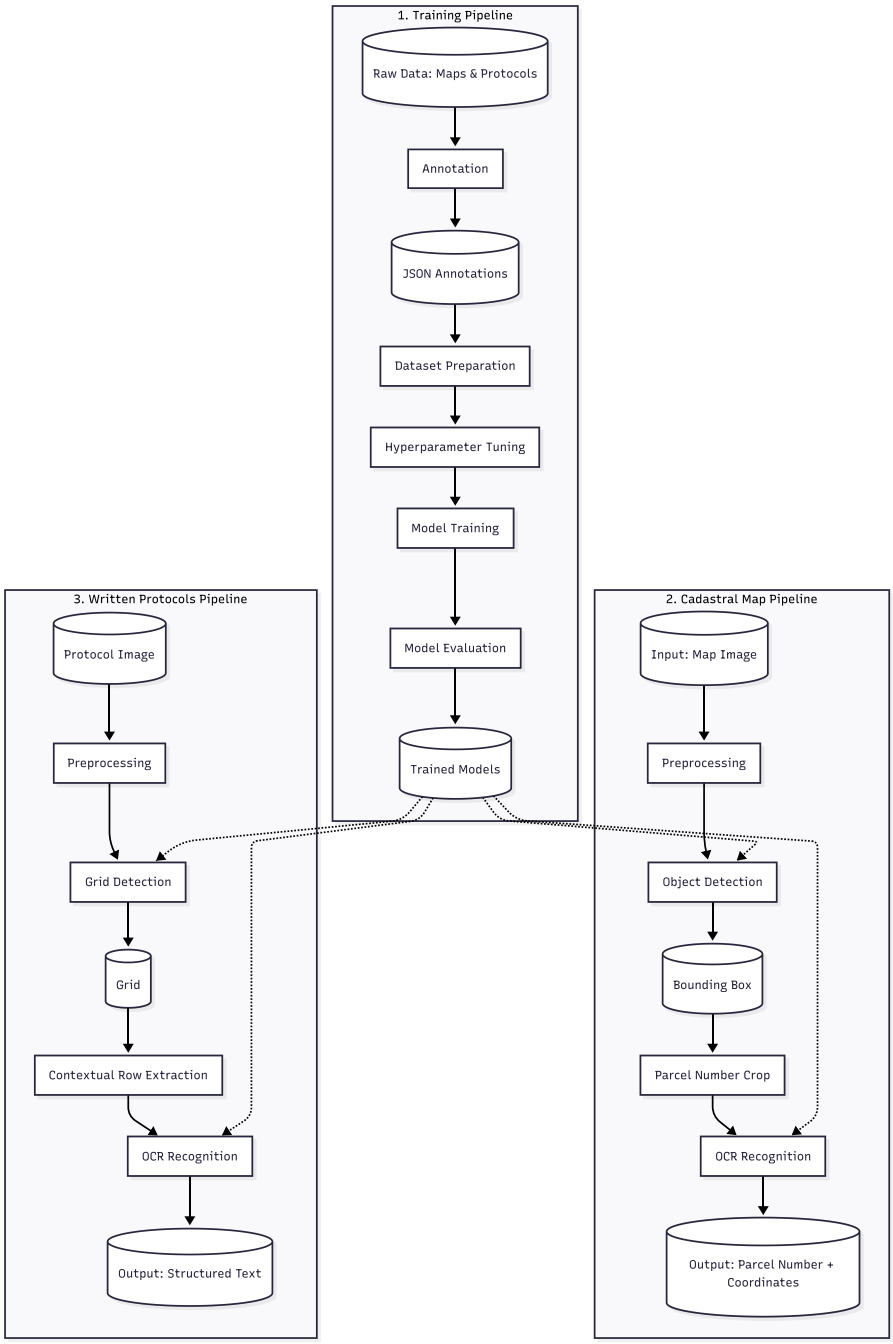


Figure 4.1: Diagram of the system architecture.

4.1.1 Cadastral Map Pipeline

The second feature of the system concerns the actual automation process of cadastral maps, which consists of the following sequential steps:

1. **Data input**
   The first step is to load the cadastral map into the system.

2. **Preprocessing**
   The image is preprocessed to eliminate any noise, such as stains or paper yellowing, and to enhance the contrast of the numbers. This step is crucial for improving the accuracy of subsequent detection and recognition.

3. **Object Detection**
   The trained detection model is applied to the preprocessed image to identify the locations of parcel numbers. The output of this step is a set of bounding boxes around the detected parcel numbers, along with their coordinates.

4. **Parcel Number Crop**
   After detecting the parcel numbers, the system crops the area around each detected number to create individual images for each parcel number. These cropped images are then passed to the OCR model for recognition.

5. **OCR Recognition**
   The cropped images are processed by the OCR model to recognize the text of the parcel numbers.

6. **Data Output**
   The data output of this process is a structured dataset containing the detected parcel numbers and their corresponding coordinates on the map.

4.1.2 Written Protocol Pipeline

The third feature of the system focuses on the automated extraction of data from written protocols. Given the tabular nature of these documents, the process follows these sequential steps:

1. **Data Input**
   The process begins by loading high-resolution scans of the handwritten protocols into the system.

2. **Preprocessing**
   Similar to the map processing, the images undergo noise reduction and contrast enhancement. Additionally, specific preprocessing techniques are ap-

plied to address issues such as the binding fold and curved lines, which are common in these documents.

3. **Grid Detection**
   Instead of simple object detection, the system performs layout analysis to identify the table structure. It detects the horizontal lines of the grid to precisely locate individual rows. This step ensures that the spatial relationship between cells is maintained.

4. **Contextual Row Extraction**
   The system crops the entire horizontal row from the protocol and processes it as a single unit.

5. **OCR Recognition**
   The extracted row images are processed by the OCR model. The model handles the sequential nature of the text, recognizing characters and words across the entire row.

6. **Data Output**
   The final output is a structured digital record (JSON file) where each row of the protocol is represented as an array of cell values.

4.2 Data Preparation and Annotation

It was necessary to prepare training data for both tasks: detection and OCR. For detection, I needed to annotate parcel numbers on maps, while for OCR, I needed to annotate the text within individual table cells.

4.2.1 Development of Software Tools for Annotation

Before preparing the training data and training the models, it was important to choose the right annotation tool. It is necessary to create a bounding box around a number and enter its value. Annotating written records is more complex. Before filling in the text in individual cells, it is necessary to find the coordinates of the table and the cells. For this reasons, I chose to create a custom annotation application for the written records and cadastral maps.

4.2.1.1 Application for Annotating Map Sheets

For the development of this application, I chose the Python [Pyt26] programming language, which is very widely used and significantly simplifies the development of annotation applications due to its rich library ecosystem. I chose the PySide6 library

[The26] for the Graphical User Interface (GUI). The application allows for opening cadastral maps and subsequent annotation. The output of this application is a JSON file for each annotated image. This file contains an array of objects representing individual annotations, where each annotation includes the coordinates of all its corners and the parcel number value. How the application looks can be seen in Figure 4.2.

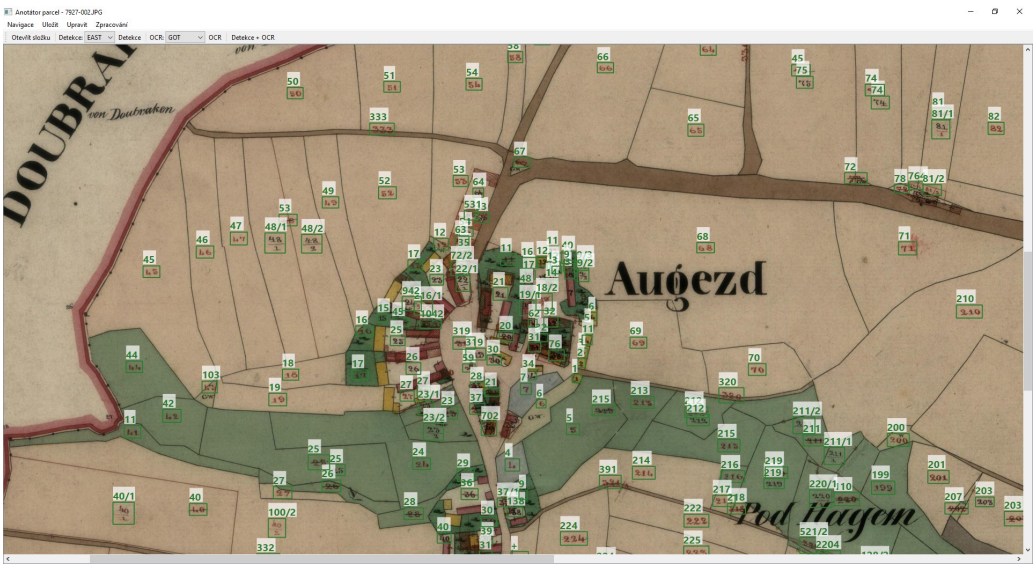


Figure 4.2: Annotation application for cadastral maps.

4.2.1.2 Application for Annotating Parcel Protocols

This application is based on the previous one and is built using the same technologies. This application must handle the detection of individual cells so that the user can efficiently fill in their values. The construction of the algorithm presented a significant technical challenge for the following reasons:

- **Binding fold**
  A binding fold was present in the middle of each scan. This was primarily a problem because the pages were not on the same plane, and there was sometimes a difference of several degrees between them.
- **Curved lines**
  In most cases, the table scans did not capture all rows in a straight position. Instead, the rows from individual tables exhibited non-linear distortion corresponding to second- to third-degree polynomials. This mostly occurred at the edges of the scans.

- **Crossed-out cells**
  Due to frequent changes, many strikethroughs appeared in the written protocols. These lines could pose a problem as the detection algorithm might misinterpret them as table lines.

The application interface can be seen in Figure 4.3.

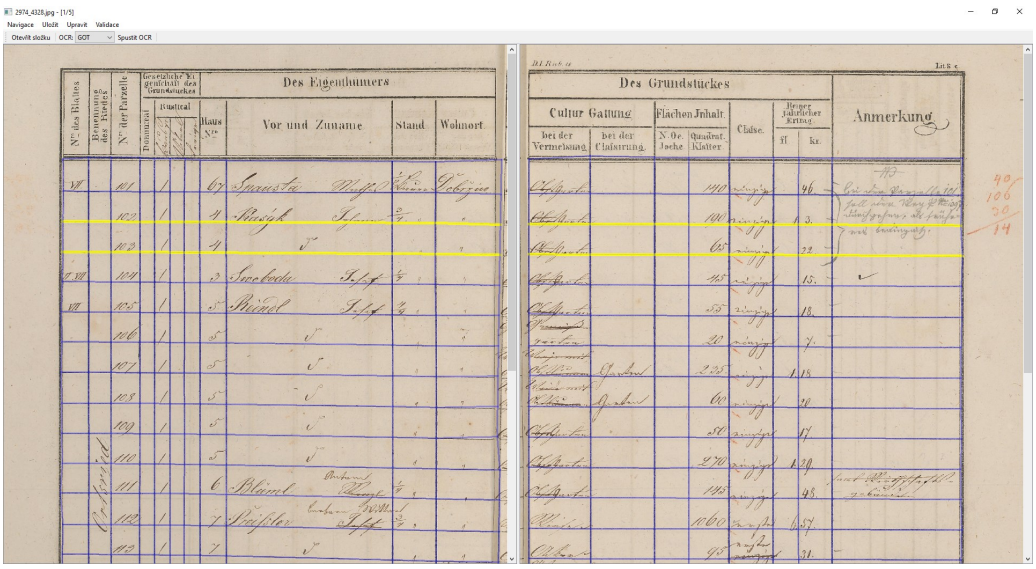


Figure 4.3: Annotation application for written records.

Grid Detection Algorithm

To eliminate the issues I implemented a custom algorithm for grid detection. Schema of the algorithm can be seen in Figure 4.4.

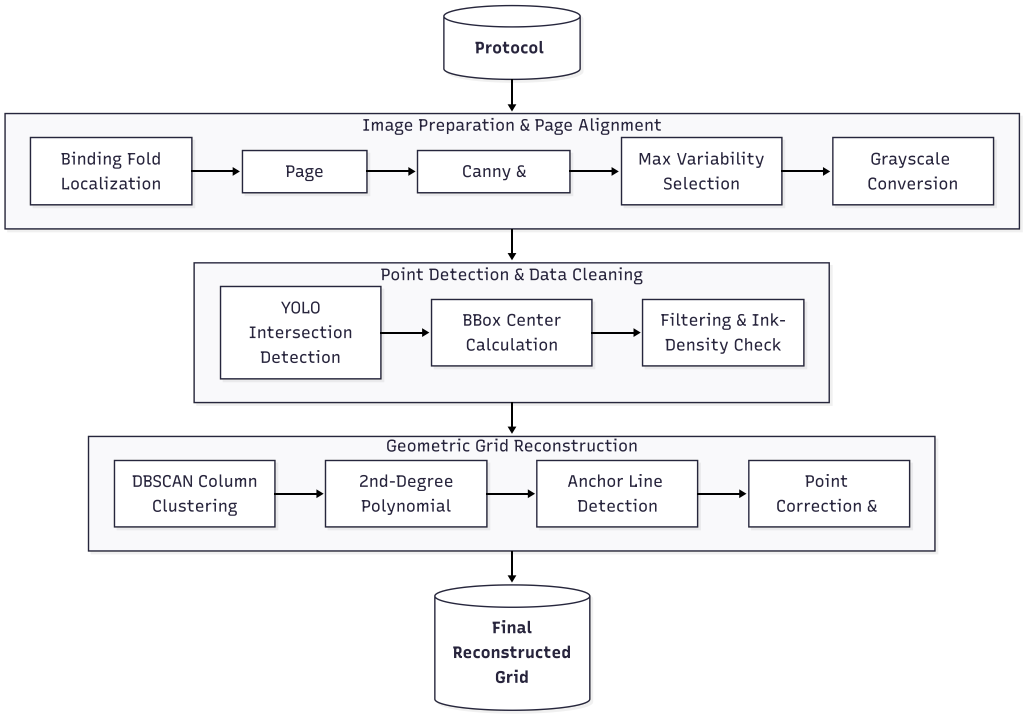


Figure 4.4: Diagram of the grid detection algorithm.

The first step is to find the binding fold. The algorithm focuses on an area ranging from 47% to 53% of the total width. This narrow search window was chosen based on an empirical analysis of the dataset; since the scans are performed with the spine approximately in the middle, limiting the search prevents the algorithm from erroneously identifying dark margins or ink artifacts at the outer edges as the binding. In this region, the vertical sum of pixel intensity is calculated. Subsequently, a global minimum is sought among these sums, corresponding to the point with the lowest brightness. This location should correspond to the shadow of the book binding. To prevent splitting the pages in the wrong place, I added a penalty function that penalizes sums based on their distance from the center. This center weighting ensures that the most probable central location is prioritized, even when darker artifacts appear near the boundaries of the search area. Along with Gaussian filtering [Sci26], this ensures a robust localization of the fold. After finding the binding, the scan is split into two pages. Since the pages are rotated independently, it is important to straighten both pages in the next step. Before searching for the correct rotation angle, significant visual noise is eliminated using a Canny detector [sci26a]. Next, an iterative process of rotating the page is performed for rotations in the range of $-5°$ to $+5°$, where the vertical sum of pixel intensity is again calculated for each iteration. The iteration that achieves the maximum variability of

vertical sums is selected. This maximum variability indicates the state in which the vertical grid lines are perfectly aligned with the Y-axis, resulting in sharp peaks and deep troughs in the projection. I chose vertical lines rather than horizontal ones for rotation detection because vertical lines exhibit minimal curvature.

After splitting and straightening the pages, I continue with the grid detection, which takes place in several steps:

1. **Preprocessing**
   To eliminate the influence of paper yellowing or color stains, I converted the pixels from color format to grayscale. This minimizes the differences in color between individual images. This difference can be observed in Figure 4.5.

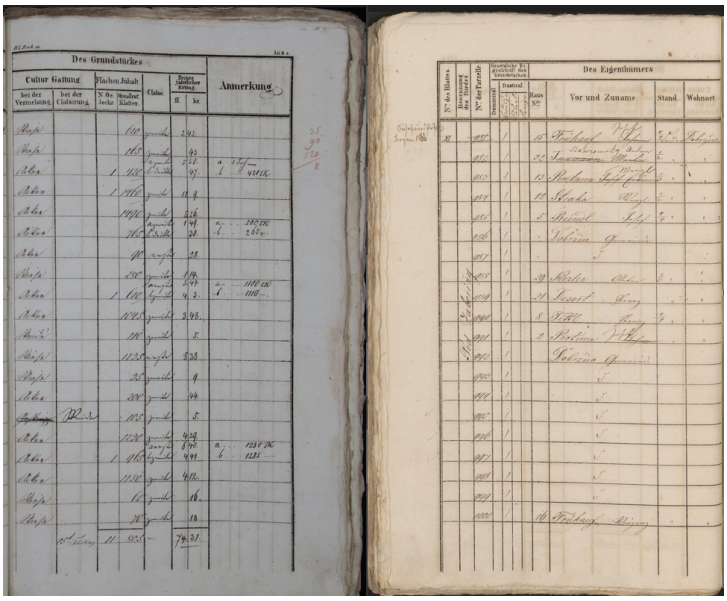


Figure 4.5: Difference in color of written records.

2. **Intersection Detection**
   For the grid detection, I chose a method where the intersections of the grid are first found using a neural network, and then the entire grid is constructed from these points. For this purpose, I trained a small YOLO model on three manually annotated written protocols. This resulted in a model capable of detecting table line intersections. Since the model does not return points directly but rather bounding boxes, I calculated the center of each box to create a point. Unfortunately, the model sometimes finds extra points, for example, due to strikethroughs in the protocols, so it is important to filter them before further processing.

3. **Point Filtering**
   Point filtering occurs in two phases. In the first phase, duplicate points are filtered out because YOLO searches the image in overlapping patches of 640x640 pixels. These overlaps create various duplicate points. I subsequently cluster these duplicate points into a single point. The threshold at which points are considered identical is a distance of 20 pixels. In the second phase, I encountered a problem where the model produced false positive detections. I filtered these hallucinations by calculating the number of dark and light pixels in the vicinity of the point (15x15 pixels) to determine if there was a sufficient amount of dark color (representing ink) near the point. The threshold for a valid point was at least 5%. Both mentioned issues can be seen in Figure 4.6.

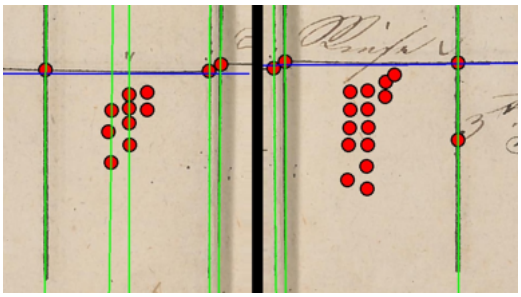


Figure 4.6: Detected points on a written record.

4. **Vertical Clustering**
   Next, I focused on detecting vertical lines. I used the DBSCAN clustering [sci26b] algorithm on the filtered points. This algorithm works by grouping points into clusters based on the differences in their x-coordinates. If points are within a certain range of each other, the algorithm groups them into the same cluster. In my case, I settled on a range of 15 pixels, which appeared to be the best during testing. From the clusters obtained this way, where each cluster represented one column, I could then create lines. I gradually fitted all points in a cluster with a first-degree polynomial line. Although the vertical table lines seemed perfectly straight, the first-degree polynomial could not accurately follow their shape, and I had to switch to a second-degree polynomial, which approximated the actual shape of the table lines with high precision and minimal deviation.

5. **Horizontal Clustering**
   I performed the same clustering for the rows, with the difference that I had to increase the range. Originally, I wanted to fit the cluster points with a curve and then construct the entire table from them, similar to the columns.

However, this clustering was not very accurate because some rows were so curved that the clustering picked up points from adjacent rows. Ultimately, I only used this clustering to find the top line of the table, which is thicker than the other lines. Thanks to this, it was again easy to determine exactly which line it was using the amount of dark pixels. This first line serves as an anchor for the subsequent steps in grid detection.

6. **Median Row Height Calculation**
   Since the column detection is very accurate, it was effective to build upon it. Based on this, I performed further filtering and removed points that were not within 15 pixels of any column curve. Subsequently, I used these points to calculate the median row height. In calculating the median, I did not include points that were closer than 30 pixels or further than 200 pixels from each other.

7. **Point Correction**
   Once I had the column curves and highly accurately filtered points, it was time for the most important step: calculating the coordinates of the final points. In this part, I iterated through individual points from the columns and corrected them one by one. Information about the location of the first line, the median row height, and the number of rows in the table was crucial. This allowed me to replace points that YOLO did not detect, move points that were detected inaccurately to the correct position, or delete points that were detected incorrectly. To ensure maximum accuracy, I also implemented synchronization across neighbors of individual columns, where the algorithm looks at previous columns when correcting points.

The result of this algorithm is the set of individual intersection points of the table lines. Thanks to these points, it is very easy to crop individual table rows and cells. This implementation represents a sophisticated approach to structural layout analysis, specifically optimized for the non-linear curves. This algorithm is very accurate and can find the table with high precision even when the rows are significantly curved. Unlike conventional grid detection methods, which typically rely on perfectly straight lines, this adaptive technique maintains exceptional structural integrity across skewed or warped pages. The ability to reliably map these intersections is a major advancement toward ensuring the robustness of the entire data extraction pipeline. An example of the final grid detection is shown in Figure 4.7. By providing such a precise geometric foundation, the algorithm effectively eliminates spatial errors that would otherwise compromise the subsequent OCR recognition, making it a critical component for achieving high-fidelity digitization of the Stable Cadastre.

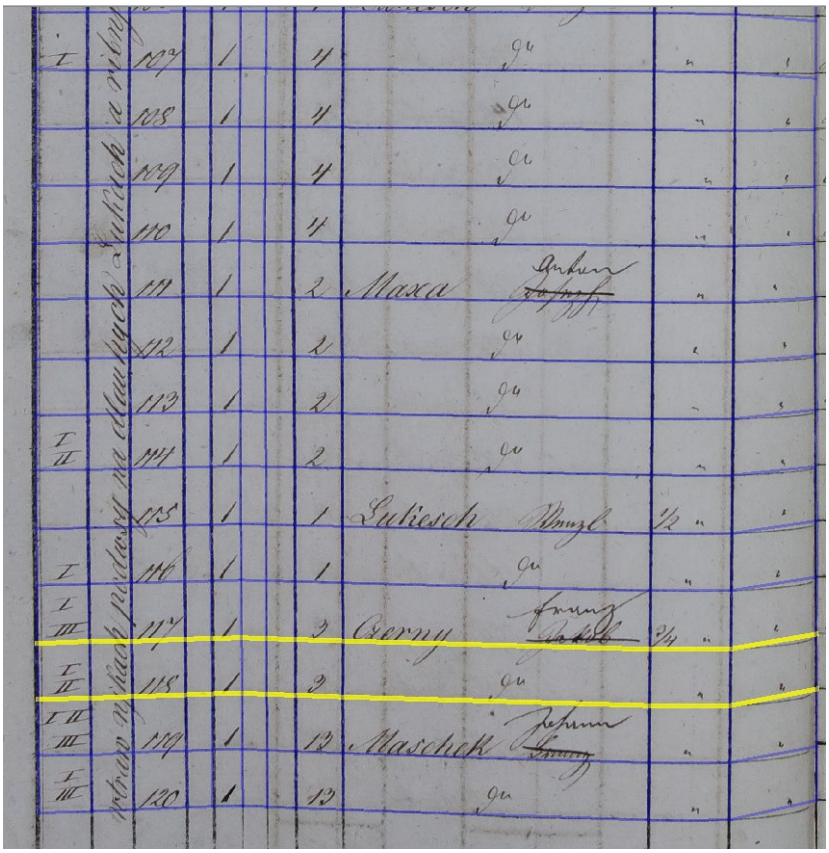


Figure 4.7: Example of successful grid detection of a written record, where the blue highlighting represents the detected grid and the yellow highlighting shows the selected row.

4.2.2 Source Data

ČÚZK provides all source data. In both cases, the data consisted of scans, and the files were sent in JPG format. While the written protocols had a relatively stable resolution of 5400x4500 pixels and a stable file size of 2.4 MB, the cadastral maps had different resolutions and sizes. Their file sizes ranged from 1 to 8.5 MB. This was because most maps had a resolution of 8400x7000 pixels, but lower-resolution maps, such as 8400x1800, also appeared in the data. In these cases, they were usually the edges of parcel areas containing only a few parcels.

4.2.3 Annotation Process

The annotation of both cadastral maps and written protocols took place in the custom annotation applications. Regarding the cadastral maps, 40 maps were annotated. From these maps, some of which had only dozens of parcels and others hundreds, a total of 5,084 individual parcel-number annotations were created. Out of these, 4605

parcel numbers were land parcels, and the remaining 479 were building parcels. Of the total number, 639 were crossed out, 1092 were in X/Y format, and 383 parcels were not in a horizontal position. These special cases were annotated with a special flag. The comprehensive statistics of these annotations are presented in Table 4.1.

Category	**Count**
Total annotated maps	40
Total parcel number annotations	5,084
Land parcels	4,605
Building parcels	479
Crossed out numbers	639
X/Y format numbers	1,092
Non-horizontal orientation	383

Table 4.1: Statistics of annotated parcel numbers from cadastral maps.

Regarding the written records, a total of 99 protocols were annotated. In 11 cases, these were building protocols; in the remaining 88, land protocols. Because building and land protocols have different formats, I had to split them into two datasets. Annotation and subsequent training were conducted row by row rather than cell by cell. I decided on this approach because, during testing by individual cells, the models would lose information regarding the context of the entire row. From the land protocols, I obtained a total of 1,760 rows, of which 653 contained strikethroughs, and 248 had text in cells in two rows above each other. These special cases were also annotated with special flags. The detailed distribution of these annotations is summarized in Table 4.2.

Category	**Count**
Land protocols	88
Building protocols	11
Total protocols	99
Land Protocols:	
Total rows annotated	1,760
Rows with strikethroughs	653
Rows with multi-line text	248

Table 4.2: Overview of annotated written protocols.

4.2.4 Annotation Speedup

After annotating a small number of cadastral maps, I trained the first YOLO detection model on this data. I then implemented these models directly into the annotation application. Thanks to this implementation, subsequent annotation proceeded much faster. Although the model lacked sufficient data for full training and misidentified many parcel numbers, it still significantly simplified and accelerated annotation work. This implementation was also the first demonstration that I was on the right track and that the models could detect parcel numbers despite the high noise in the maps.

4.2.5 Dataset Splitting

Once I had both datasets ready, it was important to split them. As mentioned in Chapter 3.4, a dataset is typically divided into three groups. In practice, a split of 70% for training, 15% for testing, and 15% for validation is most commonly used. This represents a 70/15/15 ratio. Another widespread ratio is 80/10/10. In my case, I split both datasets in a 70/20/10 ratio, as detailed in Table 4.3. By increasing the test set at the expense of the validation set, I ensured greater robustness and rigor during the final evaluation.

Dataset Category	Train	Test	Validation
Cadastral maps (images)	28	8	4
Parcel numbers	3,755	954	375
Land protocols rows	1,232	352	176

Table 4.3: Dataset split into training, testing, and validation sets.

4.3 Evaluation Metrics

To objectively compare the performance of individual models, it is essential to define a set of measurement metrics. These metrics measure the consistency between the ground truth, which represents the manually annotated true value, and the model's prediction.

4.3.1 Detection Metrics

For detection, I focus on metrics that evaluate whether the model correctly detects individual boxes and how precisely it delineates them.

IoU

Intersection over Union (IoU) [Ros16], is a fundamental metric that measures the accuracy of individual boxes. It is calculated as the ratio of the area of intersection between the predicted box (B_p) and the actual box (B_{gt}) to the area of their union.

$$IoU = \frac{Area(B_p \cap B_{gt})}{Area(B_p \cup B_{gt})}$$

The IoU value ranges from 0 to 1. The higher the IoU value, the more accurately the model predicted each box. In practice, a threshold is determined (e.g., 0.5 or 0.75). If the IoU exceeds this threshold, the detection is considered correct.

Precision and Recall

These two metrics, **Precision** [Goo24f] and **Recall** [Goo24g], are very similar, yet each determines something entirely different. Before explaining the terms themselves, it is important to define the concepts of TP, FP, and FN.

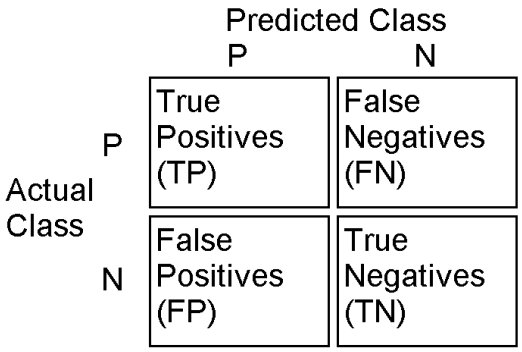


Figure 4.8: Confusion Matrix. Adapted from [AAA17].

Figure 4.8 shows a table where the ground truth results cross with the model's predictions. The table indicates that, for example, TP (True Positive) is a correctly detected value. Furthermore, FP (False Positive) is a value the model detected but should not have, and FN (False Negative) is a value the model should have detected but failed to.

- **Precision**
  Indicates how many of the detections identified by the model are actually correct. A low value of Precision means the model generates many false positives.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**
  This metric indicates what percentage of the total number of objects the model was able to detect. A low value of Recall means the model missed many objects.

$$Recall = \frac{TP}{TP + FN}$$

Both metrics range from 0 to 1. The higher the value, the more accurate the model.

F1 Score

The **F1 Score** [Goo24e] is derived from the previous metrics, specifically from Precision and Recall. This metric gives equal weight to both values and is useful for overall model comparison. It is calculated as their harmonic mean. As with the previous metrics, the values range over $[0, 1]$, with higher values indicating a better model.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

mAP

Mean Average Precision (mAP) [Goo24b] is a standard metric, used in many competitions. It is the most complex of all the aforementioned metrics. This metric is calculated as the area under the Precision-Recall curve; hence the name Average Precision, or AP. The "Mean" in the name signifies that it is the average AP value across all classes the model is searching for. In practice, mAP can be encountered in other notations, such as mAP@50 or mAP@50-90. These variations are defined by the specific IoU threshold required for a successful match. mAP@50 means that a detection is considered correct (TP) if the boxes overlap by at least 50% (i.e., IoU > 0.5). mAP@50-90 is the average mAP across IoU values from 50% to 90%. The higher the mAP value, the more robust the detector is in finding all boxes with minimal false detections.

4.3.2 OCR Metrics

In the text recognition field, bounding box precision is not the primary focus; rather, the focus is on whether the predicted text matches the ground truth.

CER

Character Error Rate (CER) [Gal24] is one of the most fundamental metrics for text recognition. It is based on the so-called Levenshtein distance, which determines

the minimum number of operations (substitutions, deletions, insertions) required to transform one text string into another. The formula for calculating CER is:

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

- **S**: number of substitutions
- **D**: number of deletions
- **I**: number of insertions
- **N**: total number of characters in the original text (ground truth)

Sequence Accuracy

While CER checks precision at the character level, **Sequence Accuracy** [Goo24b] (often referred to as Word Accuracy in OCR tasks) is a much stricter metric that examines the precision of the entire prediction. This metric evaluates the precision of the entire predicted string as a single unit. It is defined as the ratio of completely correct predictions to the total number of samples:

$$Sequence\_Accuracy = \frac{Correct\_Sequences}{Total\_Number\_of\_Sequences}$$

In the context of the Stable Cadastre, Sequence Accuracy is the most critical performance indicator. Since parcel numbers serve as unique identifiers, a single character substitution or omission (e.g., recognizing "143/1" as "143/7") results in a total loss of informative value, as the prediction no longer refers to the correct real estate entity. Therefore, even a low CER may be insufficient if it does not translate into high Sequence Accuracy.

4.4 Selection of Specific Models

The next task will be to compare and select specific models for detection and text recognition.

4.4.1 Detection and Segmentation Models

In this work, detection or segmentation is performed only on cadastral maps, and the objective is to determine the positions of individual parcel numbers. The first decision was whether to take the detection, segmentation, or both approach.

4.4.1.1 Comparison of Detection and Segmentation Approaches

This comparison was based on the analysis of both approaches in Section 3.2. My task was to choose only one approach. While it was possible to select both approaches in parallel, combining them would be extremely time-consuming for training data preparation. As previously mentioned, both approaches require different input formats. Therein lies a major disadvantage of the segmentation approach: annotating maps would require creating pixel masks for each individual parcel number. In contrast, annotating a dataset for the detection approach is very simple, as the number only needs to be marked with a bounding box.

This leads to the output of each approach. An advantage of segmentation models is the removal of background and noise around the numbers, such as individual parcel lines. This would assist subsequent text processing using OCR tools, especially classical models like Kraken. On the other hand, modern VLMs can handle background noise with ease. Another disadvantage of the segmentation approach is the hardware and computational intensity. When processing large maps at high resolutions, they must classify every pixel, which would lead to extremely slow processing of individual maps. In contrast, detection algorithms are designed for maximum time efficiency. Although the segmentation approach should yield higher-quality, cleaner data for OCR tools, I have chosen the detection path for this thesis. This is mainly due to the complex annotation of training data, as thousands of parcel numbers will need to be annotated.

4.4.1.2 Comparison and Model Selection

Based on the analysis of detection models in Section 3.2, I selected four models for comparison: Faster R-CNN, YOLO, SSD and EAST. Table 4.4 shows a comparison of these models. Based on this comparison, the most suitable candidates can be selected. When comparing the YOLO and SSD models, both from the same group, the clear winner is YOLO, which should dominate across all areas. If I further compare the YOLO and Faster R-CNN models, the comparison becomes more complex. Both models are expected to exhibit roughly similar precision. A significant difference between these models is clearly speed. Since the Faster R-CNN model processes each image twice, it will be much slower than the YOLO model, which processes each image only once. Another advantage is that the YOLO model is currently very widespread and therefore constantly developed and improved. I have thus identified the YOLO model as the more suitable option among these. The final model is the EAST model. Although it is an older model, it is the only one among the previous models specifically aimed at text localization. Its precision should therefore be high. Based on this comparison, I ultimately selected two models: YOLO and EAST. The YOLO model should be a "safe bet," where high precision and speed are expected.

The EAST model should also achieve these values, but, as mentioned, it is an older, less widely used model. It is therefore a matter for further research and testing to see how these two models perform.

Metric	Faster R-CNN	YOLO	SSD	EAST
Expected Precision	★ ★ ★ ★ ★	★ ★ ★★	★ ★ ★	★ ★ ★ ★ ★
Expected Speed	★	★ ★ ★ ★ ★	★ ★ ★★	★ ★ ★★
OCR Quality	★ ★ ★	★ ★ ★	★ ★ ★	★ ★ ★ ★ ★
Suitability for Parcels	★ ★ ★★	★ ★ ★★	★ ★ ★	★ ★ ★ ★ ★

Table 4.4: Comparison of detection models (more ★ indicates better performance).

4.4.2 OCR Models

Regarding OCR models, specifically HTR, I have selected six models for comparison based on the analysis in Section 3.3.5. Among these models is one representative from the CNN+RNN+CTC group, one from the Transformers group, and the rest from the VLM group. My selection does not include any model from the group that combines CNN with an attention mechanism. The reason is that it is a group whose complexity and precision fall between those of the previous groups. This group should, in theory, not achieve results as good as those of models built on the Transformer architecture or large multimodal language models. Based on this, I did not select any model from this group.

4.4.2.1 Comparison and Model Selection

To compare the candidate models, I evaluated them all on a prepared test set with parcel numbers; the results are shown in Table 4.5. These were models that had not been fine-tuned in any way. I calculated the CER and Accuracy for individual models over the test set. From this measurement, it was found that the best model is HunyuanOCR, which achieved a baseline CER of 28.0%. Following it were QWEN and GOT-OCR 2.0, with almost identical results: 38.5% and 38.9% CER, respectively. The DeepSeek model ranked next with a 55.8% CER, followed by the TrOCR model with 67.9%. The Kraken model placed last with a 78.3% CER. As shown in the results, large VLM models ranked in the top positions, while the Kraken model, built on a classical architecture, placed last. Based on these results, I selected a total of four models instead of two. These models are HunyuanOCR and QWEN, which will serve to compare large VLM models, followed by the GOT-OCR 2.0 model, which serves as a comparison between general VLM models and a small specialized VLM model. The final model in the selection is the classical OCR architecture Kraken, which I selected to compare modern approaches with the classical one.

Model	CER	Accuracy
Kraken	78.3	10.6
TrOCR	67.9	13.3
GOT	38.9	44.1
QWEN	38.5	43.7
Hunyuan	**28.0**	**51.7**
DeepSeek	55.8	13.1

Table 4.5: Comparison of OCR models [in %].

4.5 Implementation of Experiments and Training

With the datasets prepared, the next phase involved optimizing the training of the models.

4.5.1 Technical Equipment

Regarding technical equipment, I had access to two devices with different graphics cards for my work:

1. **NVIDIA GeForce GTX 1660 Ti**
   This was a laptop with a GPU memory of only 6 GB. With such small memory, it was possible to train smaller and less demanding models like Kraken, but large models could not be trained.

2. **NVIDIA GeForce GTX 1080 Ti**
   A desktop computer with this graphics card was used for training and testing all models. With 11 GB of GPU memory, it was possible to fine-tune even larger models such as QWEN or Hunyuan.

4.5.2 Implementation of Detection Models

For the detection of parcel numbers, I implemented a total of two methods. This section describes how this implementation was carried out.

4.5.2.1 YOLO

I implemented the YOLO models in my work using tools from Ultralytics. The implementation was carried out in a Python environment using the rich and intuitive

ultralytics library. This library contains functions for training, evaluation, and hyperparameter optimization. When selecting the specific model version in the second half of 2025, I chose the latest version, YOLO11.

Model Sizes

YOLO11 models were released in five sizes: nano, small, medium, large, and extra-large. A smaller model has fewer network layers and is very shallow. Such a model will be very fast and less demanding, but also less accurate. In my work, I tested three sizes: nano, medium, and extra-large. All these models were trained on a smaller training set and subsequently tested.

Metric	YOLO11n	YOLO11m	YOLO11x
mAP50-95	70.8	74.3	**76.7**
mAP50	95.0	95.2	**95.4**
Precision	98.1	96.4	**98.7**
Recall	90.0	**92.7**	90.6
IoU	89.4	83.3	**94.2**

Table 4.6: Comparison of YOLO model sizes [in %].

As can be observed in Table 4.6, the largest model achieved the highest accuracy on the test set. Despite it being the slowest model, and considering that speed does not play a critical role in this case, I chose the YOLO11x model for further work.

Hyperparameters

The *ultralytics* library has its own function for finding optimal hyperparameters. I identified 22 optimized hyperparameters, including learning rate and momentum. Therefore, I only manually set four basic hyperparameters for this model:

1. **Epochs count = 500**
   Since the *ultralytics* library [Ult26] returns the best model at the end of training, I set the number of epochs this high. However, a smaller number should be sufficient to find the optimum.

2. **Image size = 640px**
   This image size was chosen as the "gold standard" for the YOLO11 model.

3. **Batch size = 8**
   The batch size was determined through incremental increases. This specific size was the highest at which training remained stable without crashing due to insufficient VRAM capacity.

4. **CPU Workers = 8**
   This value was derived from the number of processor threads, which was 12. Utilizing 8 of them was determined to be an ideal number for stable training.

4.5.2.2 EAST

I implemented the EAST model using code from GitHub [Sak17]. This repository contains scripts for both training and evaluation.

Hyperparameters

For the EAST model, I set Batch size and CPU Workers to the same values as the YOLO model because I trained it on the same device. I also chose the same value for image size. The EAST model only accepts images whose size is divisible by 32, and 640px satisfies this. Compared to the YOLO model, I increased the number of epochs from 500 to 600 because it was specified in the training script. Per the configuration of the official GitHub repository, the learning rate hyperparameter was set to 1e-3.

4.5.3 Custom Evaluation Script

Even though both models had their own evaluation scripts used during training, the final evaluation results were surprising. Although YOLO appeared better in these results, visual inspection of the detections on the map suggested it was a less accurate model. Based on this, I programmed my own testing script, which I used for the final testing of both models. I utilized the aforementioned test methods within the script.

4.5.4 Implementation of OCR Models

This section describes in detail the technical implementation of training OCR models. I present the optimization steps that led to the final version of the models.

4.5.4.1 Problem with VLM Models Training

Initially, I planned to use the *Transformers* library for training the large VLM models (Hunyuan and QWEN) and the GOT model. This library belongs to Hugging Face [Hug26b] and is an open-source library that provides access to various models, ranging from image and text recognition to large VLMs. Using this library, I fine-tuned the Hunyuan and QWEN models, though for Hunyuan, the improvement was not significant over the base version. While the Hunyuan and QWEN models were successfully fine-tuned, the GOT model encountered persistent technical challenges

due to environmental dependencies and library conflicts. To ensure a stable training pipeline, the decision was made to pivot to a more robust framework, specifically **ms-swift** (Scalable lightWeight Infrastructure for Fine-Tuning) [Mod24]. This is also an open-source project under the Alibaba group. This tool is an extension of the transformers library. The tool is launched via the command line, where only the model name, data, and hyperparameters are specified. Thanks to this, I successfully fine-tuned the GOT model and eventually retrained the QWEN and Hunyuan models, with Hunyuan undergoing significant improvement. However, I still use the *transformers* library to run these models.

4.5.4.2 Fine-tuning Configuration Using Adapters

Since it was not possible to fine-tune large models with billions of parameters, such as QWEN or Hunyuan, on the available hardware, I had to utilize the **LoRA** [Hu+21] (Low-Rank Adaptation) method, which is part of the ms-swift framework. This technique drastically reduces the number of trained parameters while maintaining accuracy comparable to full fine-tuning. It works on the principle that the original model weights remain frozen and that low-rank matrices, or adapters, are integrated into the model structure and subsequently trained. This makes the training of these large models less computationally and memory-intensive.

4.5.4.3 QWEN

For the QWEN model, I searched for the optimal learning rate over a small number of epochs. The ideal value was selected based on the training loss function and the resulting test error rate. In total, I tested five values ranging from 1e-4 to 8e-6. Regarding the best convergence in the training phase, two values were among the best configurations: 1e-4 and 2e-5, where the former had a better starting loss and the best gradient norm. Ultimately, I chose 2e-5 over 1e-4, as it achieved a CER of 28.5% on test data, while 1e-4 achieved 31.7%. The largest difference between these models was in the crossed-out parcel numbers, where the CER difference was 25%. Other configurations were overall similar, except for 8e-6, which showed clearly the worst CER and Grad Norm.

4.5.4.4 Hunyuan

Similar to the QWEN model, I searched for the optimal learning rate (LR) in the range from 2e-4 to 2e-5. Although all tested LR values showed similar overall CERs (ranging from 26.8% to 27.7%), the largest differences were in the Grad Norm. This norm was best for the LR value of 2e-4. Likewise, this value also had the best CER

error rates. Therefore, I chose 2e-4 as the clearly best value and continued with it in further training.

4.5.4.5 GOT

For the GOT model, I decided to use the same learning rate of 2e-5 as for the QWEN model without a deeper search for the optimal value. This decision was made because both models share a similar internal architecture and because training is highly computationally intensive. In subsequent analysis of training progress, the model achieved stable loss function convergence and high validation accuracy.

4.5.4.6 Kraken

I chose the Kraken model as the baseline comparison. For training, I used the official *Ketos* tool. This tool, combined with the *kraken* library, allows for training new Kraken models or performing fine-tuning. In my case, I used an already trained model, which I further trained on my data. In total, I tested three models without fine-tuning, compared them, and selected one. All three are pre-trained, either fully or partially, on German text, which is a great advantage for this thesis.

Model	CER	Accuracy
greek-german_serifs_bsb10234118	97.2	0.2
greek-german_serifs_sophokle1v3soph	95.5	0.0
McCATMuS	**78.4**	**10.6**

Table 4.7: Comparison of Kraken models [in %].

This Table 4.7 shows the performance of different Kraken models, from which it is evident that the best model based on the best CER value 78.4% is *McCATMuS*, which I proceeded with. I then fine-tuned this model using early stopping on my training set with various learning rates ranging from 1e-3 to 1e-5 (see Table 4.8).

Learning Rate	CER	Accuracy
1e-3	22.1	60.9
5e-4	**20.7**	**61.8**
1e-4	23.0	59.2
5e-5	22.3	59.1
1e-5	24.0	57.9

Table 4.8: Comparison of CER values for different learning rates for the Kraken model [in %].

The best model was trained with a learning rate of 5e-4 (see Table 4.8). This training showed an optimal convergence curve and achieved the best CER values. In contrast, training with a higher LR (specifically 1e-3) was too fast. The other extreme was the LR value of 1e-5, where training took four times longer than at 5e-4. Despite such a long training period, the model achieved the worst CER results.

4.5.4.7 Prompt Engineering

Since I use two VLMs in my work, specifically Hunyuan and QWEN, it was important to find the optimal prompts. For both models, my goal was to find ideal prompts for both parcel numbers and for reading data from the written protocol tables.

Parcel Numbers

When creating these prompts, I emphasized that the input is a standalone number that may contain a slash or a hyphen. This resulted in a total of 13 prompts: some very simple, some stating that it is a parcel number, and others stating that the text is handwritten. I tested these prompts on a single image containing a total of 549 different annotations.

Prompt	Hunyuan	QWEN
OCR:	34.2	80.7
Read the text.	37.9	74.9
Identify the number.	34.3	**42.3**
Recognize the parcel number.	32.8	93.8
Recognize the number in the image. Digits only.	33.4	47.6
Identify the parcel number. It contains only digits, optionally a slash '/'.	32.1	155.3
Transcribe the handwritten number from the cadastral map.	28.8	52.2
Read the text. Format: 123 or 123/1.	44.1	50.1
Extract the parcel number. Examples: 123, 123/1.	38.1	99.2
What is the number written in the image?	**28.3**	49.8
Convert the image to text. The image contains a parcel number.	218.5	116.6
Recognize the handwritten digits and symbols.	39.1	58.8
Read the number. Ignore background noise.	29.7	54.2

Table 4.9: Results of suitable prompts for parcel numbers [in %].

Table 4.9 shows that the results for the Hunyuan model were very similar, except for one prompt, ranging from 28% to 44%. These results are already very good and

show that I am on the right track. For this model, I chose the prompt with the lowest CER: 'What is the number written in the image?'. For the QWEN model, the results were not as consistent and certainly not as accurate as for the Hunyuan model. The best prompt for QWEN was 'Identify the number.', which I chose as the final prompt. It is interesting to see that QWEN could not achieve the same accuracy in its base version as Hunyuan.

Written Protocols

Since the tables contain more than just numbers, it was necessary to devise more complex prompts that include additional information. It is important to add information to these prompts about how many columns the table has, what to do with empty columns, and what they contain. I compiled 10 prompts and tested them on 30 rows from the written protocols. The results can be seen in Table 4.10. For the Hunyuan model, I chose the fourth prompt, which reached a total CER of only 118.0%, which is an excellent result compared to the other prompts. It should be noted that the CER can mathematically exceed 100% when the model generates a significant number of insertions relativee to the number of characters in the ground truth. In the context of these experiments, values above 100% indicate that the model not only misrecognized the characters but also produced redundant strings, repeated words, or failed to adhere to the requested structural constraints. For the QWEN model, the results were more balanced. Even so, I selected the seventh prompt with the lowest overall CER of 223.1%. However, this value is much worse than that of the Hunyuan model.

Prompt	Hunyuan	QWEN
OCR Table Row: Transcribe this image into exactly 18 cells separated by '|'. Every cell must be accounted for. If a cell is empty, you MUST write '[EMPTY]'. Format: cell1 | cell2 | ... | cell18. Do not add any conversational text.	502.9	584.4
Convert the handwritten row to a pipe-separated string. Requirements: 1) Exactly 18 values separated by '|'. 2) Empty cells = [EMPTY]. 3) Preserve all numbers, German words, and Roman numerals. No preamble.	249.5	239.6

(table continues on the next page)

Table 4.10 (*continued from previous page*)

Prompt	**Hunyuan**	**QWEN**
Identify every cell in the row from left to right. There are 18 columns in total. Use '|' as a separator. If you cannot see any text in a column, label it as '[EMPTY]'. Output only the resulting string.	322.5	264.7
You are an expert in transcribing 19th-century land registry protocols. Read this row and output 18 pipe-separated cells. Handle the Kurrent-style German script and Roman numerals accurately. Use '[EMPTY]' for blank spaces.	**118.0**	411.1
Transcribe this historical document row. The content is a mix of 19th-century German, parcel numbers, and Roman numerals (e.g., I, II, III, IV). Use '|' to separate the 18 columns. Blank columns must be [EMPTY].	700.6	390.8
This image contains a handwritten table row from a historical protocol. There are exactly 18 cells separated by '|'. Recognize the content of each cell. If a cell is empty, write '[EMPTY]'. Use '|' as a separator. The row contains mixed German text, numbers, and Roman numerals.	675.8	**223.1**
Read the table row and ignore background noise, ink blots, or paper degradation. There are 18 columns. Focus only on the handwritten text and numbers. Format as 18 cells separated by '|'. Empty = [EMPTY].	288.1	268.9
Scan the image for handwritten data. Even if a column looks blank, you must include it in the 18-cell count as '[EMPTY]'. Use '|' as the delimiter. Ensure the final output has exactly 17 pipe symbols.	218.2	484.9

(*table continues on the next page*)

Table 4.10 (*continued from previous page*)

Prompt	Hunyuan	QWEN
Transcribe 18 cells using '|'. Example: Name | [EMPTY] | 120 | IV | ... | [EMPTY]. Follow this pattern for the provided image. Recognize German text and Roman numerals. Do not skip any column.	334.0	596.1
1. Count 18 columns. 2. Transcribe text/numbers. 3. Use '|' separator. 4. Use '[EMPTY]' for blanks. Output the resulting pipe-separated line now. Target content: Historical German script and measurements.	236.2	287.8

Table 4.10: CER results of prompts for written records [in %].

4.5.4.8 Expanding Tables with Separators

Once all models were trained, I considered ways to improve the accuracy of detecting the correct number of cells. For example, for the Hunyuan model, this was 91.3%. Since I had information on exactly where each cell begins and ends, I decided to expand the row crops with custom separators. I added a 10-pixel-thick green line where two cells were supposed to split. This increased the detection accuracy of the aforementioned model by 4% to 95.4%. A disadvantage of this approach could be the loss of context when splitting words that extend into an adjacent cell. However, since we were already splitting words with a separator during annotation, this should not be an issue.

4.5.4.9 Expanding the Training Set

Before the final training, I excluded all rows from the training dataset that contained either crossed-out text or text appearing in multiple lines within a cell. Models trained this way achieved very good results. However, in tests on written protocols containing this filtered data, the models were not as effective. I gradually added rows with crossed-out text to the training dataset and subsequently multi-line rows. After adding this complex data and retraining the models, although the overall CER error rate, Accuracy, and cell count precision were not so good, the models showed greater reliability in real-world use on written protocols. I also had to expand the prompts for the QWEN and Hunyuan models gradually. The resulting prompt for the QWEN model is:

> This image contains a handwritten table row from a historical protocol. It consists of exactly 18 cells separated by thick green vertical bars. Recognize the content of each cell and use '|' as a separator in your response. If a cell is empty, write '[EMPTY]'. If a word or number is crossed out, enclose it in double quotes (e.g., "text"). If a cell contains text on multiple lines, separate the lines with a semicolon (e.g., "hello world;I hear"). The row contains mixed German text, numbers, and Roman numerals.

And for the Hunyuan model:

> You are an expert in transcribing 19th-century land registry protocols. Read this row and output 18 pipe-separated cells. Cells are separated by thick green vertical bars. Handle the Kurrent-style German script and Roman numerals accurately. Use '[EMPTY]' for blank spaces. If a word or number is crossed out, enclose it in double quotes (e.g., "text"). If a cell contains text on multiple lines, separate the lines with a semicolon (e.g., "hello world;I hear").

4.6 Final Results and Comparison

This section presents the results of my experiments. It also describes the final comparison of both the detection models and the OCR models.

4.6.1 Detection Evaluation

Based on the comparison in Section 4.4.1.2, two distinct architectures, EAST and YOLO, were selected for the detection task. The detection results are summarized in Table 4.11. The table includes the previously mentioned metrics such as Precision, mAP50, and IoU. Additionally, it shows the time the model took to perform detection over the test set.

Metric	EAST	YOLO
Precision	**91.0**	86.4
Recall	**94.0**	90.0
F1 score	**92.5**	88.2
mAP50	85.9	**87.3**
IoU	75.0	**79.5**
Time	**140.8s**	176.9s

Table 4.11: Detection results [in %].

The measured values indicate that the YOLO and EAST models are quite evenly matched. The EAST model achieved a Recall of 94%, which in practice means it misses only 6% of the parcel numbers. The highest number of these undetected parcel numbers occurs in densely built-up locations. An example is shown in Figure xy, where the EAST model manages to detect parcel numbers in these locations very accurately but misses a few. In contrast, Figure xy shows detection on the same map performed by the YOLO model, which misses more detections. This is reflected in the 4% lower Recall value. Similarly, the YOLO model has a lower Precision value.

YOLO achieved higher mAP50 and IoU scores. However, the metric is highly skewed in this case because the EAST model cannot handle confidence scores for individual predictions and returns a value of 1 for every detection. The YOLO model thus handles the filtering of uncertain results much better, making this metric less informative for EAST. Similarly, the IoU metric may be skewed for the EAST model because it can return rotated boxes. This can result in the model delineating a number with a rotated box more accurately, but when compared to ground truth using a standard axis-aligned bounding box, the resulting IoU was low. Despite both metrics being stricter on the EAST model, it maintains values close to those of the YOLO model. The EAST model is also 25% faster than YOLO. Overall, the better model is the EAST, which excellently handles the detection of even less legible numbers.

4.6.2 OCR Evaluation

When evaluating the OCR models, I focused on three main aspects. The first was the overall CER and Accuracy, followed by the reading speed. Speed was calculated as the average time taken to read a single parcel number or a table row.

4.6.2.1 Parcel Numbers

The models were trained on the full dataset using the best parameters and optimal prompts.

Metric	Kraken	GOT	Hunyuan	QWEN
CER	20.7	17.0	**16.0**	20.4
Accuracy	61.4	66.7	**67.4**	64.2
Time	**0.1s**	0.7s	2.2s	4.3s

Table 4.12: Comparison of performance results of the OCR model for parcel numbers [in %].

Looking at the results in Table 4.12 from the perspective of CER and Accuracy, the values are quite interesting. As expected, the Kraken model achieved the lowest

performance. However, I was pleasantly surprised by the result. I expected its values to be much farther from those of the other models, but compared to the large VLM model QWEN, its CER was only 0.3% worse. A larger difference was seen in the Accuracy value, specifically 3%. A surprise is the GOT model, which ranked second, with results very similar to those of the Hunyuan model. Considering that GOT is built on a small QWEN model, I am surprised it even outperformed the large QWEN model. Comparing the two large VLM models, the search for the ideal prompt already indicated that the Hunyuan model is much better suited for this task than its competitor. While the models are very similar in CER and Accuracy, there are significant differences in speed. The reading speed results again aligned with expectations, with the less demanding models reading data much faster than the large VLM models. Inference time was measured on the specific hardware configuration described in Section 2. These times will vary significantly depending on the GPU architecture, VRAM capacity, and overall hardware performance.

For parcel numbers, I added another aspect to the evaluation: CER and Accuracy for specific cases, specifically, crossed-out numbers and numbers with slashes. In total, the test dataset contained 213 numbers with slashes and 121 crossed-out numbers. Here are the exact data for each model:

Category	**Kraken**	**GOT**	**Hunyuan**	**QWEN**
CER [%]				
Simple	14.4	14.5	**12.8**	20.0
Fraction	31.6	18.6	17.3	**15.0**
Crossed	**19.4**	24.8	29.0	40.0
Accuracy [%]				
Simple	74.2	74.4	**76.0**	69.7
Fraction	24.4	53.1	54.5	**59.6**
Crossed	**64.5**	51.2	46.3	43.8

Table 4.13: Comparison of OCR performance metrics: CER (lower is better) and Accuracy (higher is better).

From the results in Table 4.13, it can be observed that the large VLMs handled slashes in text better, while Kraken loses in this category. However, Kraken dominated in crossed-out text, handling it better than all other models, while the VLMs struggled with it. The most balanced model was GOT, which performed very well across all categories.

Based on all the preceding results, I selected GOT as the winning model. While this model does not dominate any single category, it is never the worst performer. GOT is very accurate and simultaneously very fast. In a scenario where time is not

a factor, I would choose the Hunyuan model. It is the most accurate model, but it is up to three times slower than GOT. Conversely, if speed were the primary consideration at the expense of accuracy, the Kraken model performed very well. The biggest disappointment was the QWEN model, which was expected to be one of the best models for this task, but even though its performance was good, it was not as good as expected.

4.6.2.2 Written Records

In addition to testing parcel numbers, I also evaluated the model's ability to precisely detect the correct number of columns, as this significantly impacts its accuracy. CER and Accuracy values were evaluated against the entire table row.

Model	CER	Accuracy	Time	Correct Column Count
Kraken	31.8	0.3	**0.2s**	48.6
GOT	21.2	0.6	6.9s	81.5
Hunyuan	**8.6**	**27.0**	7.6s	92.6
QWEN	11.8	11.4	14.9s	**95.2**

Table 4.14: Comparison of OCR results for tabular data [in %].

At first glance, it is clear from Table 4.14 that for written records, there is a larger difference in accuracy between the models. VLMs are much more accurate than the others. The reason is certainly their ability to detect the number of columns correctly. It can be seen that Kraken returns the correct number of columns in only 48.6% of cases. In contrast, the QWEN model provides a much clearer overview of where individual cells begin and end, achieving a very good 95.2% success rate. Consequently, Kraken accurately identified an entire row in only 0.3% of cases. The Hunyuan model clearly dominates this category, correctly reading more than a quarter of the rows without error and making the fewest overall mistakes. The GOT model is again an average between Kraken and the large VLMs; it is not the worst in any metric, but it does not dominate in any metric either. While the models are very similar in CER and Accuracy, there are significant differences in speed. The reading speed results again aligned with expectations, with the less demanding models reading data much faster than the large VLM models. It is important to note that the inference time was measured on the specific hardware configuration mentioned in Section 2. These times will vary significantly depending on the GPU architecture, VRAM capacity, and overall hardware performance.

A detailed evaluation at the individual-cell level was also prepared for written records (see Table 4.15).

Record Field	Kraken	GOT	Hunyuan	QWEN
1 - Sheet Number (Roman Number)	65.6	72.7	16.7	24.7
2 - Parcel Number	83.5	60.1	16.2	28.4
3 - Legal Property of the Plot I	530.0	102.5	30.0	32.5
4 - Legal Property of the Plot II	42.1	19.9	14.6	12.6
5 - Legal Property of the Plot III	0.0	0.0	0.0	0.0
6 - Legal Property of the Plot IV	22400.0	200.0	100.0	100.0
7 - House Number	211.8	69.8	20.7	15.8
8 - Owner Name	71.2	46.6	14.7	21.3
9 - Status/Occupation	76.8	41.2	16.9	21.3
10 - Residence	110.2	42.6	14.2	16.0
11 - Culture (at Surveying)	65.7	46.6	21.6	25.2
12 - Culture (at Classification)	288.6	196.4	50.0	65.0
13 - Area (Yoke)	605.8	202.9	75.4	73.9
14 - Area (Square Klafter)	106.9	52.9	18.3	23.1
15 - Yield Class	62.7	31.7	15.0	14.1
16 - Net Yield (Florin)	148.2	118.5	37.8	38.4
17 - Net Yield (Kreuzer)	100.6	79.6	28.1	41.6
18 - Remarks	100.1	78.1	50.5	65.9

Table 4.15: Comparison of CER per specific column within written records [in %].

The most important column in the entire written record is the second column, which contains the parcel number. There, the Hunyuan model clearly dominates, even over the QWEN model. Based on this, I have selected the Hunyuan model as the most successful for written records.

4.7 Discussion and Error Analysis

This section focuses on a deeper analysis of the results from the previous chapter and attempts to address specific problems encountered during the process.

4.7.1 Analysis of Model Limits and Errors

In this section, I would like to describe where the models reached their limits and where they erred most frequently.

1. **The problem of densely built-up areas**
   The most significant problem in parcel number detection is clearly the presence of densely built-up areas. Here, the models inaccurately detect parcel numbers, merge them into a single record, or fail to detect them at all. Both problems are shown in Figure 4.9, where the EAST model fails to detect the

number 16 and the crossed 74. It even merged the numbers 64 and 13 into a single number.



Figure 4.9: Example of detection errors in densely built-up areas.

2. **Boundary point numbers**
   Along the boundaries of the cadastre, there are boundary points (turning points) that are also numbered. The detection models sometimes confuse these numbers with parcel numbers, leading to false-positive detections of extra numbers. False detections are shown in Figure 4.10.

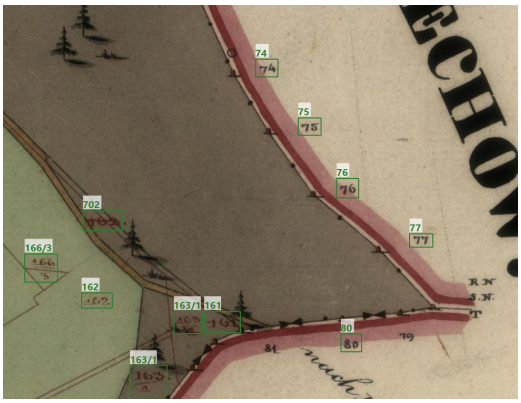


Figure 4.10: Example of detection errors along boundaries.

3. **Crossed-out numbers**
   Crossed-out parcel numbers posed the greatest challenge for OCR methods. The models "hallucinated" the most in these cases. A surprise in this category was the Kraken model, which performed the best among all tested models. In Figure 4.10, it can be seen that the model falsely detected the crossed-out number 162 as 702.

4. **OCR models and cell detection**
   Regarding written records, large VLM models clearly dominated this category. This is mainly due to their ability to maintain a broader context across the entire image and their superior ability to detect individual cells. An advantage of these models is the fact that they accept a prompt as input, in which the sought objects in the image can be precisely described. In contrast, the classical Kraken model reached its limits in cell detection.

5. **Errors in empty cells**
   The models occasionally struggled with empty cells in the tables. An example is the sixth column, which contains no value and is always empty. As seen in Table 4.14, all models hallucinated and added an extra character. The Kraken model hallucinated the most here, reaching an enormous value of 22400% CER. Such a high value could result from the model's failure to correctly detect the number of cells, potentially inserting data from neighboring cells into this one.

4.7.2 Possible Improvements

Based on the analysis of the entire project, I see several clear areas for improvement.

1. **Training on rotated bounding boxes**
   Given that the EAST model outperformed the YOLO model, I see a definite improvement in transitioning from ordinary bounding boxes to the rotated bounding boxes (OBB) that the EAST architecture uses by default. This would require adjusting the annotation application and retraining the models, but it would eliminate the problem of OCR models reading text that is not horizontal.

2. **Linking cadastral maps and written records**
   This is the goal toward which the entire project is heading. Now that it is easy to obtain data from both parcel protocols and written records, there is no problem linking these datasets using parcel numbers.

3. **Obtaining information on parcel type**
   In its current state, the project can find and read parcel numbers and extract data from tables. However, it is unclear whether a read number belongs to a building or a land parcel. This extension would be crucial for linking the datasets mentioned above.

Conclusion 5

The objective of this thesis was to design and implement a comprehensive system for automating data processing from the Stable Cadastre. Overall, I implemented two methods for parcel number detection and four methods for text recognition. Furthermore, I developed two annotation applications for cadastral maps and written records, integrating the aforementioned models. In this work, I also performed a complete analysis of the results and proposed future directions for the system's development. Thus, I have fulfilled all the requirements of the thesis specifications.

Regarding detection itself, the comparison revealed differences between general object detection and specialized text localization. The EAST model proved to be the most suitable for the cadastral maps, achieving excellent results. While the YOLO-based approach provided high speed and robustness in identifying standard labels, the EAST detector excelled in handling the varied orientations and aspect ratios characteristic of historical map annotations. It faces challenges only in densely built-up areas and at boundary points where the visual congestion of features is extreme.

Concerning text recognition methods, I reached several interesting findings. Classical models, such as Kraken, are capable of competing with VLM models in pure digit recognition without difficulty; however, they struggle with more complex operations, such as table cell detection. In contrast, modern VLMs can handle diverse and complex tasks and can even replace specialized OCR tools. The GOT model proved to be the most suitable for parcel number recognition, while the Hunyuan model performed best for the text in written records.

Despite numerous challenges with historical documents, I achieved high-quality results. These results demonstrate that modern neural networks are capable of replacing manual labor in processing the Stable Cadastre, albeit with a certain margin of error. I hope to build upon this work in the future and to further improve and expand it into a form that truly facilitates human labor.

Bibliography

[Ala24] ALAMMAR, Jay. *The Illustrated Transformer*. 2024. Available also from: https://jalammar.github.io/illustrated-transformer/. [Online; accessed 2026-04-29].

[Ama24] AMAZON WEB SERVICES. *What is a Neural Network?* 2024. Available also from: https://aws.amazon.com/what-is/neural-network/. [Online; accessed 2026-04-28].

[APX26] APXML. *Introduction to Neural Networks: Early Stopping* [https://apxml.com/courses/introduction-to-neural-networks/chapter-6-improving-network-performance/early-stopping]. 2026. [Online; accessed 2026-04-29].

[Bri26] BRILLIANT.ORG. *Backpropagation*. 2026. Available also from: https://brilliant.org/wiki/backpropagation/. [Online; accessed 2026-04-28].

[Čes24] ČESKÝ ÚŘAD ZEMĚMĚŘICKÝ A KATASTRÁLNÍ. *Historie pozemkových evidencí*. 2024. Available also from: https://www.cuzk.cz/Katastr-nemovitosti/O-katastru-nemovitosti/Historie-pozemkovych-evidenci.aspx. [Online; accessed 2026-04-28].

[Čes26a] ČESKÝ ÚŘAD ZEMĚMĚŘICKÝ A KATASTRÁLNÍ. *Český úřad zeměměřický a katastrální* [online]. 2026. [visited on 2026-05-06]. Available from: https://cuzk.gov.cz/.

[Čes26b] ČESKÝ ÚŘAD ZEMĚMĚŘICKÝ A KATASTRÁLNÍ. *Legenda stabilního katastru*. 2026. Available also from: https://geoportal.cuzk.cz/(S(5lvpjrqfqxkfzg5h3j4uhdtk))/Dokumenty/CIO_Legenda.ZIP. [Online; accessed 2026-04-28].

[Che+21] CHENG, Bowen; MISRA, Ishan; SCHWING, Alexander G.; KIRILLOV, Alexander; GIRDHAR, Rohit. Masked-attention Mask Transformer for Universal Image Segmentation. *arXiv preprint arXiv:2112.01527* [https://arxiv.org/abs/2112.01527]. 2021. [Online; accessed 2026-04-29].

[Dee25] DEEPSEEK-AI. DeepSeek-V3 Technical Report. 2025. Available also from: https://arxiv.org/abs/2412.19437. DeepSeek-AI Technical Report. [Online; accessed 2026-04-29].

[Gal24] GALILEO AI. *Character Error Rate (CER) Metric*. 2024. Available also from: https://galileo.ai/blog/character-error-rate-cer-metric. [Online; accessed 2026-04-29].

[Gee24a] GEEKSFORGEEKS. *Impact of Learning Rate on a Machine Learning Model*. 2024. Available also from: https://www.geeksforgeeks.org/machine-learning/impact-of-learning-rate-on-a-model/. [online; accessed 2026-05-06].

[Gee24b] GEEKSFORGEEKS. *Introduction to Pooling Layer*. 2024. Available also from: https://www.geeksforgeeks.org/deep-learning/introduction-to-pooling-layer-cnn/. [Online; accessed 2026-04-29].

[Gee24c] GEEKSFORGEEKS. *Object Detection Models: Faster R-CNN*. 2024. Available also from: https://www.geeksforgeeks.org/computer-vision/object-detection-models/#4faster-rcnn. [Online; accessed 2026-04-29].

[Gee24d] GEEKSFORGEEKS. *R-CNN (Region-based CNNs)*. 2024. Available also from: https://www.geeksforgeeks.org/machine-learning/r-cnn-region-based-cnns/. [Online; accessed 2026-04-29].

[Goo24a] GOOGLE DEVELOPERS. *Machine Learning Crash Course: Reducing Loss*. 2024. Available also from: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent. [Online; accessed 2026-04-28].

[Goo24b] GOOGLE DEVELOPERS. *Machine Learning Glossary*. 2024. Available also from: https://developers.google.com/machine-learning/glossary. [Online; accessed 2026-04-28].

[Goo24c] GOOGLE DEVELOPERS. *Machine Learning Glossary: Batch*. 2024. Available also from: https://developers.google.com/machine-learning/glossary#batch. [online; accessed 2026-05-06].

[Goo24d] GOOGLE DEVELOPERS. *Machine Learning Glossary: Epoch*. 2024. Available also from: https://developers.google.com/machine-learning/glossary#epoch. [online; accessed 2026-05-06].

[Goo24e] GOOGLE DEVELOPERS. *Machine Learning Glossary: F1 Score*. 2024. Available also from: https://developers.google.com/machine-learning/glossary#f1. [online; accessed 2026-05-06].

[Goo24f] GOOGLE DEVELOPERS. *Machine Learning Glossary: Precision*. 2024. Available also from: https://developers.google.com/machine-learning/glossary#precision. [online; accessed 2026-05-06].

[Goo24g] GOOGLE DEVELOPERS. *Machine Learning Glossary: Recall*. 2024. Available also from: https://developers.google.com/machine-learning/glossary#recall. [online; accessed 2026-05-06].

[Goo26] GOOGLE DEVELOPERS. *Machine Learning Crash Course: Dividing Datasets*. 2026. Available also from: https://developers.google.com/machine-learning/crash-course/overfitting/dividing-datasets. [Online; accessed 2026-04-28].

[Hak+11] HAKIM, Seyed Jamalaldin Seyed; NOORZAEI, Jamaloddin; JAAFAR, M. S.; JAMEEL, Mohammed; MOHAMMADHASSANI, Mohammad. Application of artificial neural networks to predict compressive strength of high strength concrete. *International Journal of the Physical Sciences*. 2011, vol. 6, no. 5, pp. 975–981. Available also from: https://www.researchgate.net/publication/268380188_Application_of_artificial_neural_networks_to_predict_compressive_strength_of_high_strength_concrete. Figure 1: Architecture of a typical multilayer feed forward neural network [Online; accessed 2026-04-28].

[He+17] HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* [https://arxiv.org/abs/1703.06870]. 2017, pp. 2961–2969. [Online; accessed 2026-05-12].

[Hu+21] HU, Edward J. et al. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*. 2021. Available also from: https://arxiv.org/abs/2106.09685.

[Hug26a] HUGGING FACE. *Connectionist Temporal Classification (CTC)* [https://huggingface.co/learn/audio-course/chapter3/ctc]. 2026. Hugging Face Audio Course. [Online; accessed 2026-04-29].

[Hug26b] HUGGING FACE. *Hugging Face – The AI community building the future* [https://huggingface.co/]. 2026. [Online; accessed 2026-05-11].

[IBM24a] IBM. *What are neural networks?* 2024. Available also from: https://www.ibm.com/think/topics/neural-networks. [Online; accessed 2026-04-28].

[IBM24b] IBM. *What is a Loss Function?* 2024. Available also from: https://www.ibm.com/think/topics/loss-function. [Online; accessed 2026-04-28].

[IBM24c] IBM. *What is Hyperparameter Tuning?* 2024. Available also from: https://www.ibm.com/topics/hyperparameter-tuning. [Online; accessed 2026-04-28].

[IBM24d] IBM. *What is Optical Character Recognition (OCR)?* 2024. Available also from: https://www.ibm.com/topics/optical-character-recognition. [online; accessed 2026-05-06].

[Jor18] JORDAN, Jeremy. *Object detection: one-stage methods*. 2018. Available also from: https://www.jeremyjordan.me/object-detection-one-stage/. [Online; accessed 2026-04-29].

[Kie24] KIESSLING, Benjamin. *Introduction to ATR with kraken*. 2024. Available also from: https://kraken.re/7.0/introduction_to_atr.html. [Online; accessed 2026-04-29]. Kraken documentation.

[Lab24] LABELLERR. *Semantic vs Instance vs Panoptic: Which Image Segmentation Technique to Choose?* 2024. Available also from: https://www.labellerr.com/blog/semantic-vs-instance-vs-panoptic-which-image-segmentation-technique-to-choose/. [Online; accessed 2026-04-29].

[Lee+20] LEE, Junyeop et al. On Recognizing Texts of Arbitrary Shapes with 2D Self-Attention. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [https://arxiv.org/abs/1910.04396]. 2020, pp. 5469–5478. [Online; accessed 2026-05-12].

[LJY24] LI, Fei-Fei; JOHNSON, Justin; YEUNG, Serena. *CS231n: Convolutional Neural Networks for Visual Recognition – Module 1: Neural Networks Part 1*. Stanford University, 2024. Available also from: https://cs231n.github.io/neural-networks-1/. [Online; accessed 2026-04-28].

[Li+21] LI, Minghao et al. TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models. 2021. Available also from: https://arxiv.org/abs/2109.10282.

[Liu+16] LIU, Wei et al. SSD: Single Shot MultiBox Detector. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016, pp. 21–37. Available also from: https://arxiv.org/abs/1512.02325.

[Mod24] MODELSCOPE. *MS-Swift: A Multi-modal Large Model Training and Deployment Framework* [https://github.com/modelscope/ms-swift]. 2024. GitHub repository. [Online; accessed 2026-05-11].

[Ngu23] NGUYEN, Linh. *Rectified Linear Unit (ReLU) Activation Function: Guide & Examples*. 2023. Available also from: https://www.datacamp.com/blog/rectified-linear-unit-relu. [online; accessed 2026-05-06].

[Nit+21] NITHILAKSHKUMAR, S.; SIBI CHAKRAWARTHY, S.; SIVARAMAN, R.; NANDHINI, K.; HEMALATHA, S. Performance Evaluation and Analysis of Drone-based Vehicle Detection Techniques From Deep Learning Perspective. *IEEE Access / ResearchGate*. 2021. Available from DOI: 10.13140/RG.2.2.14652.16000. [Online; accessed 2026-04-29].

[Nit24] NITIN. *What are the differences in Image Classification, Detection, Segmentation?* 2024. Available also from: https://medium.com/@Nitin_Indian/what-are-the-differences-in-image-classification-detection-segmentation-d2f1e9db9d17. [Online; accessed 2026-04-29].

[NVI24] NVIDIA. *What are Vision-Language Models?* 2024. Available also from: https://www.nvidia.com/en-us/glossary/vision-language-models/. [Online; accessed 2026-04-29].

[ON15] O'SHEA, Keiron; NASH, Ryan. An Introduction to Convolutional Neural Networks. *arXiv preprint arXiv:1511.08458*. 2015. Available also from: https://www.researchgate.net/publication/333593451_An_Introduction_to_Convolutional_Neural_Networks. Figure 2: Illustration of Max Pooling and Average Pooling [Online; accessed 2026-04-29].

[Pil24] PILÁT, Martin. *Neuronové sítě: Konvoluční sítě a zpracování obrazu*. 2024. Available also from: https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/neuronove-site-konvolucni-site-zpracovani-obrazu. [Online; accessed 2026-04-29].

[Pne26] PNEUMA, Fabien (milesial). *PyTorch U-Net: Semantic segmentation with PyTorch* [https://github.com/milesial/pytorch-unet]. 2026. [Online; accessed 2026-05-12].

[Pyt26] PYTHON SOFTWARE FOUNDATION. *Python Programming Language* [https://www.python.org/]. 2026. [Online; accessed 2026-05-01].

[Qwe25] QWEN TEAM. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*. 2025. Available also from: https://arxiv.org/abs/2505.09388. [Online; accessed 2026-04-29].

[AAA17] AL-RIKABI, Shumoos J.; AL-HADI, Israa A.; AHMED, Omar S. Classification of SURF Image Features by Selected Machine Learning Algorithms. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2017, vol. 7, no. 7, pp. 235–240. Available from DOI: 10.23956/ijarcsse/V7I7/01111. [Online; accessed 2026-04-29].

[Rob24] ROBERTS, Eric. *The History of Computing: Neural Networks*. Stanford University, 2024. Available also from: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html. [Online; accessed 2026-04-28].

[Ros16] ROSEBROCK, Adrian. *Intersection over Union (IoU) for object detection*. 2016. Available also from: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Online; accessed 2026-04-29].

[Ros22] ROSEBROCK, Adrian. *Semantic vs. Instance vs. Panoptic Segmentation*. 2022. Available also from: https://pyimagesearch.com/2022/06/29/semantic-vs-instance-vs-panoptic-segmentation/. [Online; accessed 2026-04-29].

[Sak17] SAKURARIVEN. *EAST: An Efficient and Accurate Scene Text Detector*. 2017. Available also from: https://github.com/SakuraRiven/EAST. [Online; accessed 2026-04-29]. GitHub repository.

[sci26a] SCIKIT-IMAGE DEVELOPMENT TEAM. *Canny edge detector — scikit-image documentation* [https://scikit-image.org/docs/stable/auto_examples/edges/plot_canny.html]. 2026. [Online; accessed 2026-05-11].

[sci26b] SCIKIT-LEARN DEVELOPERS. *sklearn.cluster.DBSCAN — scikit-learn documentation* [https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html]. 2026. [Online; accessed 2026-05-11].

[Sci26] SCIPY DEVELOPERS. *scipy.ndimage.gaussian_filter — SciPy documentation* [https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html]. 2026. [Online; accessed 2026-05-11].

[Sha+18] SHAFI, Urwa et al. Classification of Plant Leaf Diseases Using Multi-Level Features and Convolutional Neural Network. *Proceedings of the 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2018. Available from DOI: 10.1109/ICOMET.2018.8346437. Figure 1: Convolution between a 3x3 kernel and a 6x6 feature map [Online; accessed 2026-04-29].

[SBY15] SHI, Baoguang; BAI, Xiang; YAO, Cong. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2015. Available also from: https://arxiv.org/abs/1507.05717. [Online; accessed 2026-04-29].

[Ten25] TENCENT HUNYUAN TEAM. *Hunyuan-OCR: End-to-End OCR Expert Vision-Language Model*. 2025. Available also from: `https://vision.hunyuan.tencent.com/`. [Online; accessed 2026-04-29].

[The26] THE QT COMPANY. *PySide6: Official Python bindings for the Qt cross-platform application and UI framework* [`https://pypi.org/project/PySide6/`]. 2026. PyPI project. [Online; accessed 2026-05-11].

[Ult24a] ULTRALYTICS. *What is a Backbone? - Ultralytics YOLO Glossary*. 2024. Available also from: `https://www.ultralytics.com/glossary/backbone`. [Online; accessed 2026-04-29].

[Ult24b] ULTRALYTICS. *YOLO11 Documentation: Real-Time Object Detection*. 2024. Available also from: `https://docs.ultralytics.com/models/yolo11/`. [Online; accessed 2026-04-29].

[Ult26] ULTRALYTICS. *Ultralytics: State-of-the-art AI and YOLO models* [`https://www.ultralytics.com/`]. 2026. [Online; accessed 2026-05-11].

[Wan+24] WANG, Jingxu et al. From Convolutional to Generative: A Survey on Handwritten Text Recognition. *arXiv preprint arXiv:2412.11657*. 2024. Available also from: `https://arxiv.org/abs/2412.11657`. [Online; accessed 2026-04-29].

[Wei+24] WEI, Haoran et al. General OCR Theory: Towards OCR-2.0 via a Unified End-to-end Model. *arXiv preprint arXiv:2409.01704*. 2024. Available also from: `https://arxiv.org/abs/2409.01704`. [Online; accessed 2026-04-29].

[Wei+22] WEISS, Romano; KARIMIJAFARBIGLOO, Sanaz; ROGGENBUCK, Dirk; RÖDIGER, Stefan. Applications of Neural Networks in Biomedical Data Analysis. *Biomedicines*. 2022, vol. 10, no. 7, p. 1469. Available from DOI: `10.3390/biomedicines10071469`. Obrázek 1: Structure of an artificial neuron [Online; accessed 2026-04-28].

[Zho+17] ZHOU, Xinyu et al. EAST: An Efficient and Accurate Scene Text Detector. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5551–5560. Available also from: `https://arxiv.org/abs/1704.03155`.

List of Figures

2.3 Visualization of different parcel number types. Building parcels are highlighted in black boxes, while land parcels are highlighted in red boxes. 3
2.1 Color coding legend for the surveying record. Adapted from [Čes26b]. 4
2.2 Example of a surveying record from the dataset. 4
2.4 Example of a land written record. 6
2.5 Example of a building written record. 6

3.1 Diagram of an artificial neuron. Adapted from [Wei+22]. 8
3.2 Basic layers of a neural network. Adapted from [Hak+11]. 9
3.3 Convolutional filter. Adapted from [Sha+18]. 13
3.4 Pooling methods, Max Pooling (right) and Average Pooling (left). Adapted from [ON15]. 14
3.5 Semantic, Instance, and Panoptic segmentation. Adapted from [Lab24]. 17
3.6 Architecture diagram of a multimodal model combining visual and text inputs to generate responses. Adapted from [NVI24]. 20

4.1 Diagram of the system architecture. 24
4.2 Annotation application for cadastral maps. 27
4.3 Annotation application for written records. 28
4.4 Diagram of the grid detection algorithm. 29
4.5 Difference in color of written records. 30
4.6 Detected points on a written record. 31
4.7 Example of successful grid detection of a written record, where the blue highlighting represents the detected grid and the yellow highlighting shows the selected row. 33
4.8 Confusion Matrix. Adapted from [AAA17]. 36
4.9 Example of detection errors in densely built-up areas. 55
4.10 Example of detection errors along boundaries. 55

1 Creating a new annotation in the map. 70
2 Editing an existing annotation. 70

3 Written records application with populated values. 71
4 Adjusting the table grid. 72
5 Editing a table row. 72

List of Tables

4.1 Statistics of annotated parcel numbers from cadastral maps. 34
4.2 Overview of annotated written protocols. 34
4.3 Dataset split into training, testing, and validation sets. 35
4.4 Comparison of detection models (more ★ indicates better performance). 40
4.5 Comparison of OCR models [in %]. 41
4.6 Comparison of YOLO model sizes [in %]. 42
4.7 Comparison of Kraken models [in %]. 45
4.8 Comparison of CER values for different learning rates for the Kraken model [in %]. 45
4.9 Results of suitable prompts for parcel numbers [in %]. 46
4.10 CER results of prompts for written records [in %]. 49
4.11 Detection results [in %]. 50
4.12 Comparison of performance results of the OCR model for parcel numbers [in %]. 51
4.13 Comparison of OCR performance metrics: CER (lower is better) and Accuracy (higher is better). 52
4.14 Comparison of OCR results for tabular data [in %]. 53
4.15 Comparison of CER per specific column within written records [in %]. 54

List of Abbreviations

CER Character Error Rate

CNN Convolutional Neural Network

CRNN Convolutional Recurrent Neural Network

CTC Connectionist Temporal Classification

EAST Efficient and Accurate Scene Text detector

GUI Graphical User Interface

HTR Handwritten Text Recognition

IoU Intersection over Union

LLM Large Language Model

mAP Mean Average Precision

OCR Optical Character Recognition

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SSD Single Shot MultiBox Detector

VE Vision Encoder

VLM Vision-Language Models

YOLO You Only Look Once

ČÚZK Český úřad zeměměřický a katastrální

User Manual

This chapter describes how to operate the annotation applications designed for cadastral maps and written records. Both applications are designed to be straightforward and user-friendly.

Cadastral Maps

After launching the application (illustrated in Figure 4.2), the first step is to use the 'Otevřít složku' button in the top-left corner to select the folder containing the map images. Once the folder is loaded, the first map is automatically displayed on the canvas. You can pan across the map by holding the Ctrl key and the left mouse button. To zoom in or out, hold Ctrl and use the mouse wheel. To initiate parcel number detection, recognition, or both, use the 'Detekce', 'OCR', and 'Detekce + OCR' buttons in the top toolbar. Before running these processes, you can select specific models for both detection and OCR.

The maps can also be annotated manually. By holding the left mouse button, you can draw a bounding box and fill in the corresponding value, as shown in Figure 1. Existing bounding boxes can be modified by right-clicking on them; this opens a window (see Figure 2) where you can update the value or delete the box. To streamline the workflow, the application includes several keyboard shortcuts:

- Right Arrow - move to the next image in the folder
- Left Arrow - move to the previous image in the folder
- Ctrl + Delete - delete all bounding boxes on the current map
- Ctrl + S - save the current file
- Ctrl + Z - undo/delete the last drawn bounding box

The application automatically saves a JSON file with annotations whenever you navigate between images or close the program. If you open a file that already contains annotations, they will be loaded and displayed on the canvas automatically.

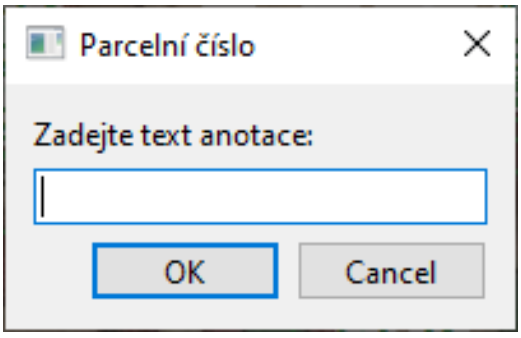


Figure 1: Creating a new annotation in the map.

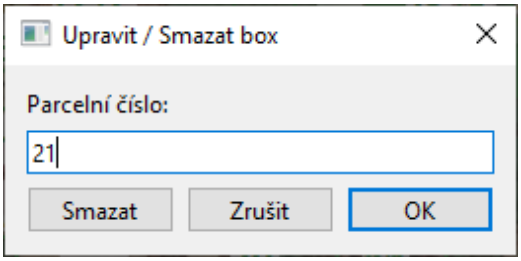


Figure 2: Editing an existing annotation.

Written Records

The application for processing written records is shown in Figure 4.3. Similar to the previous tool, you must first open the folder containing the records. Upon opening, the application automatically attempts to detect the table grid, which is then rendered directly onto the canvas. Text recognition can then be started using the 'Spustit OCR' button, with the option to select a specific model.

Rows that have already been annotated and contain text are highlighted in green (see Figure 3). Individual rows can be edited manually by left-clicking on them, which opens the editing window shown in Figure 5. Rows are edited separately for the left and right sides of the table. To speed up the process, you can move to the next row directly from this window by pressing Enter. Once text recognition is complete, the application automatically performs a sequence correction for the parcel numbers. The application also supports the following keyboard shortcuts:

- Right Arrow - move to the next image in the folder
- Left Arrow - move to the previous image in the folder
- Ctrl + S - save the file

- Ctrl + E - enter table grid editing mode (see Figure 4), allowing you to manually move the points of the detected grid
- Ctrl + R - restart the table grid detection
- Ctrl + F - fix the sequence of parcel numbers
- Ctrl + V - run validation on the detected grid

As with the map application, annotations are automatically saved to a JSON file when navigating between images or exiting. If a file with existing annotations is opened, the values are automatically populated into the respective rows.

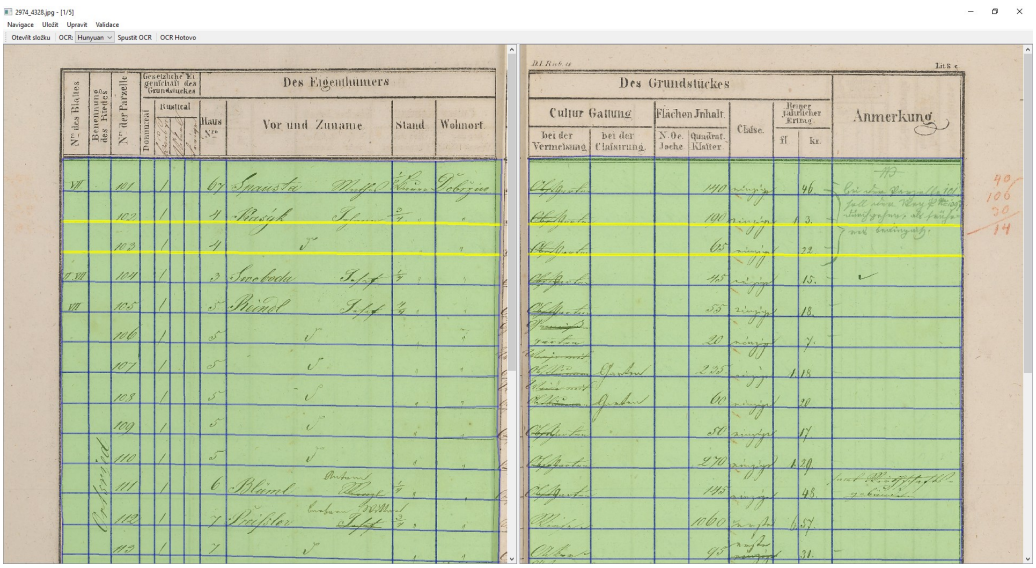


Figure 3: Written records application with populated values.

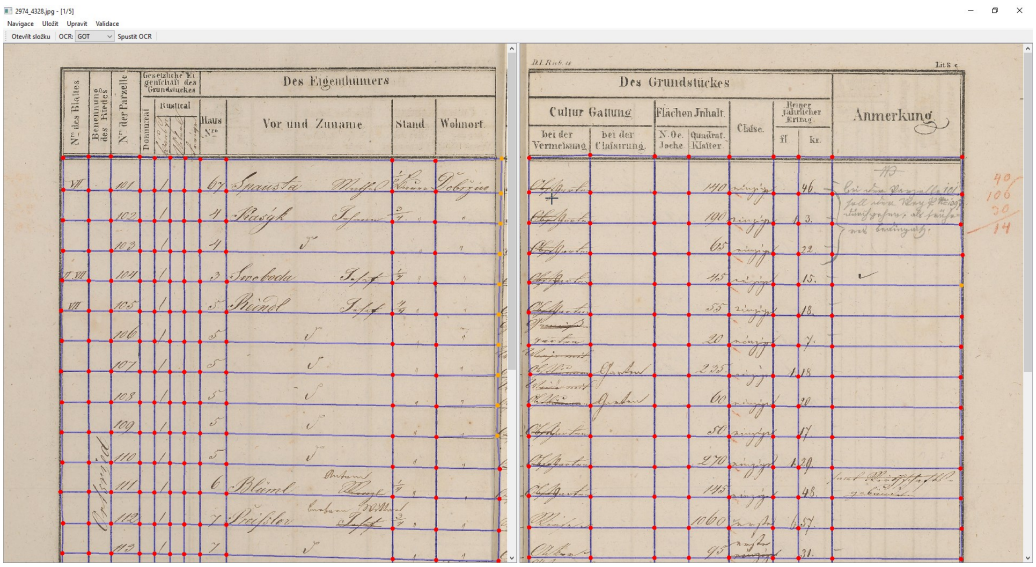


Figure 4: Adjusting the table grid.



Figure 5: Editing a table row.

