FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Detection of text in historical maps
Student:	Adam Peňáz
Supervisor:	Mgr. Petr Šimánek
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

1) Survey literature describing the latest methods in object detection (e.g. PSENet [3], transformers [4]).
2) Analyze a dataset of historical maps [1,2].
3) Choose, explain and implement two methods in Pytorch. Both methods will be applied to a dataset of historical maps [1,2].
4) Apply more general methods trained on general (non-map) text detection datasets [5] to understand if pre-trained models can effectively be used on historical maps.
5) Analyze, compare, and discuss the results in detail.

[1] Nomenclature Dataset v 1.0: Dataset for Detection and Recognition of Handwritten Nomenclatures and toponyms from Historical Cadastral Maps, L. Lenc et al.
[2] Historical Map Toponym Extraction for Efficient Information Retrieval, L. Lenc et al.
[3] Shape Robust Text Detection with Progressive Scale Expansion Network, W. Wang et al.
[4] Visual Transformer for Object Detection, M. Yang et al.
[5] https://github.com/cs-chan/Total-Text-Dataset

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 20 October 2022 in Prague.

Bachelor's thesis

DETECTION OF TEXT IN HISTORICAL MAPS

Adam Peňáz

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Petr Šimánek
May 10, 2023

Citation of this thesis: Peňáz Adam. *Detection of text in historical maps.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

I would like to thank my supervisor, Mgr. Petr Šimánek, for his time, and effort, and for providing me with the topic and practical knowledge of neural networks. I would also like to thank my family, with a special mention of my girlfriend, and classmates from FIT CTU for their unwavering support throughout my academic journey.

Declaration

In Prague on May 10, 2023 . . . .

Abstract

Text Detection is a challenging task, especially when it comes to detecting specific types of text such as map Nomenclatures on historical maps. This thesis proposes two CNN-based networks, PSENet and TextPMs, implemented in PyTorch for detecting map Nomenclatures, and evaluates their performance using the Nomenclature dataset. Additionally, the thesis explores the effectiveness of pre-trained models on the TotalText dataset for detecting nomenclatures in historical maps. Although pre-trained models did not yield promising results, the proposed methods achieved f-measure scores of 88.8% and 91.1%, respectively, demonstrating their suitability for the task. Overall, the thesis contributes to the field of historical map analysis by introducing effective methods for text detection in this challenging domain.

Keywords text detection, historical maps, convolutional neural networks, PyTorch, PSENet, TextPMs, deep learning

Abstrakt

Detekce textu je náročný úkol, zejména pokud jde o detekci konkrétních typů textu, jako jsou například názvy míst na historických mapách. Tato práce představuje dvě metody založené na konvolučních neuronových sítích, PSENet a TextPMs, implementované v PyTorch knihovně pro detekci názvů míst na historických mapách a hodnotí jejich výkon pomocí Nomenclature datasetu. Kromě toho práce zkoumá účinnost předtrénovaných modelů na TotalText datasetu pro detekci nomenklatur v historických mapách. I když předtrénované modely nedosáhly slibných výsledků, navržené metody dosáhly f-skóre v hodnotě 88, 8 % a 91, 1 %, což dokazuje jejich vhodnost pro tuto úlohu. Celkově tato práce přináší přínos do oblasti analýzy historických map tím, že představuje účinné metody pro detekci textu v této náročné oblasti.

Klíčová slova detekce textu, historické mapy, konvoluční neuronové sítě, PyTorch, PSENet, TextPMs, hluboké učení

Acronyms

Adam	Adaptive Moment Estimation
BFS	Breadth First Search
BN	Batch Normalization
CC	Connected Component
CNN	Convolutional Neural Network
EMA	Exponential Moving Average
FC	Fully Connected
FN	False Negative
FP	False Positive
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
IM	Iterative Module
IoU	Intersection over Union
OHEM	Online Hard Example Mining
PSENet	Progressive Scale Expansion Network
ReLu	Rectified Linear Unit
ResNet	Residual Network
RGB	Red, Green, and Blue
SAF	Sigmoid Alpha Function
SGD	Stochastic Gradient Descent
TextPMs	Text Detection with Probability Maps
TP	True Positive

Chapter 1
Introduction

The problem of detecting text is an important topic in the field of computer vision. Historical maps contain a wealth of information about the past, including place names, landmarks, and other textual information that is often critical for research and analysis. There are many potential applications across various fields such as history, geography, cartography, and archaeology. However, the text in these maps can be difficult to detect and extract due to the complex nature of the maps and the text itself, which can vary in size, font, and orientation. As a result, developing effective methods for text detection in historical maps is essential for preserving and understanding our cultural heritage.

I chose this topic because I believe that improving text detection on historical maps can provide significant benefits. By accurately detecting text on these maps, researchers and enthusiasts can better understand historical events, study the evolution of place names, and create more accurate maps for modern use.

The main focus of this thesis is to implement and evaluate the effectiveness of two CNN-based networks, PSENet and TextPMs, for detecting nomenclatures on historical maps using the Nomenclature dataset. The thesis also explores the effectiveness of pre-trained models on the TotalText dataset.

The thesis is structured into five main chapters. The first chapter, Theoretical Background, explains the fundamental concepts underpinning the methodology chapter. The second chapter, Related Works, reviews existing literature and previous work on the topic. The third chapter, Datasets Analysis, describes and analyzes the Nomenclature and TotalText datasets. The fourth chapter, Methodology, explains the PSENet and TextPMs methods used in the study. The final chapter, Experiments, summarizes the experiments and presents their outcomes and conclusions.

Chapter 2
Objectives

The main objective of this thesis is to implement and apply two CNN-based methods to a dataset of historical maps for text detection. Both methods will be implemented in Pytorch.

The next objective is to describe and analyze a dataset of historical maps to gain insights into the challenges associated with detecting text in such maps.

The last objective is to apply models trained on the TotalText dataset to understand if pre-trained models can effectively be used on historical maps.

Detecting text on historical maps can help to understand the maps better and reveal what is depicted on them.

Chapter 3

Theoretical Background

This chapter provides an overview of the fundamental concepts and terminology utilized in this thesis. The chapter begins by introducing Convolutional Neural Networks (CNNs), which form the core building block for both methods employed. The second section focuses on Residual Network (ResNet) [1], a specific CNN architecture that has gained widespread usage in the image processing field. The final section of the chapter outlines the Feature Pyramid Network (FPN) [2], which serves as the backbone of both methods[1].

3.1 Convolutional Neural Networks

A type of neural network known as CNN is designed to handle data with a grid-like topology, such as images. Specifically, a digital image consists of a series of pixels organized in a grid-like manner, with each pixel assigned a value that represents its brightness and color. The CNN architecture is optimized for processing such data and has demonstrated exceptional performance in various computer vision tasks. This section is based on [3, 4].

3.1.1 Architecture Overview

A simple CNN comprises a series of layers that transform one set of activations to another using a differentiable function. The three main types of layers used to build CNN are the *Convolutional Layer*, *Pooling Layer*, and *Fully-Connected Layer*, which are also used in regular Neural Networks. These layers are combined to form a complete CNN architecture as shown in Figure 3.1.

3.1.2 Convolutional Layer

The convolutional layer is a fundamental part of the CNN and carries the main computational load. In the field of mathematics, convolution refers to a mathematical operation performed on two given functions f and g, in the continuous case defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

and in the discrete case as:

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

[1]PSENet and TextPMs

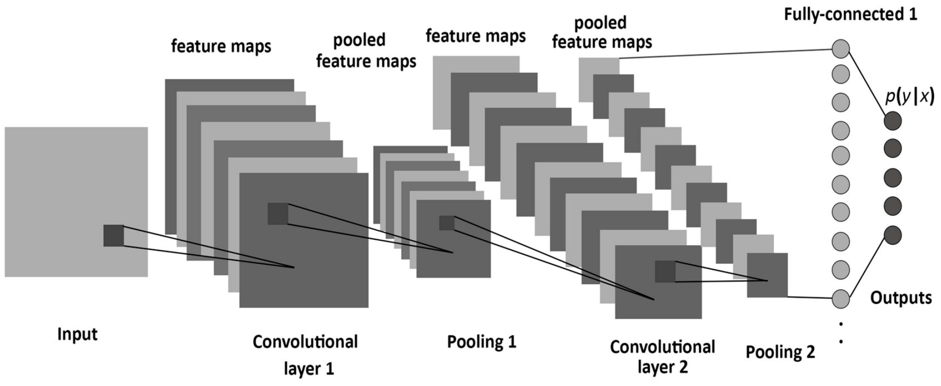


Figure 3.1 The structure of a CNN consists of three main layer types. [5]

The initial parameter f is referred to as the *input*, while the second parameter g is called the convolution *kernel*.

In the context of a network, f and g are represented as a *tensor*[2]. For simplicity, let us imagine the tensor as a *matrix*, then this layer performs a dot product between two matrices 3.2, where the kernel matrix is the set of learnable parameters. The other matrix represents the restricted part of the receptive field. The kernel is smaller than the image in terms of its spatial dimensions but is greater in depth. This means that if an image is composed of three channels (RGB), the kernel will have small spatial height and width dimensions, but the depth will extend to all three channels.

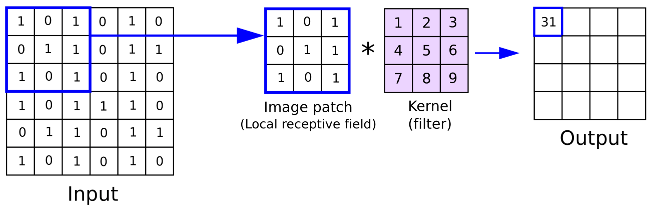


Figure 3.2 The dot product of multiplying two matrices that have been flattened. [6]

In the forward pass, the kernel steps over the height and width of the input image to produce a representation of that receptive region, resulting in a two-dimensional activation map that indicates the kernel response at each spatial position of the image. The size of the step is called a *stride*.

3.1.3 Activation Functions

After applying convolution to an image, non-linear activation functions are often added to introduce non-linearities to the activation map since images are far from linear. There are different types of activation functions, with popular ones including:

Sigmoid The sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$ is a commonly used activation function that maps real-valued numbers into a range between 0 and 1. This range is often interpreted as a probability measure. However, a significant disadvantage of using the sigmoid

[2]A tensor is a mathematical object that represents a multi-dimensional array of data.

function is that it suffers from *the vanishing gradient problem*, where the gradient becomes almost zero at the tails of the function.

Tanh The tanh function, defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ "squashes" real-valued numbers to the range between -1 and 1. It saturates like a sigmoid but is zero-centered, unlike sigmoid neurons.

ReLu The Rectified Linear Unit, defined as $f(x) = \max(0, x)$ is a popular activation function in recent years. It works by setting any negative input to zero, while positive values remain unchanged. This results in faster convergence compared to sigmoid and tanh, making ReLU a more reliable choice.

3.1.4 Pooling Layer

The pooling layer performs a summarization that replaces the output of the network at specific locations by computing a summary statistic of the nearby outputs. The key objective of this layer is to reduce the spatial size of the representation, resulting in a reduced computational load and weights. The pooling operation is performed individually on every slice of the representation.

There are several types of pooling functions, including the average of the rectangular neighborhood, the $L2$ norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. Nevertheless, the most commonly used method is max pooling, which returns the maximum output from the neighborhood as shown in Figure 3.3.

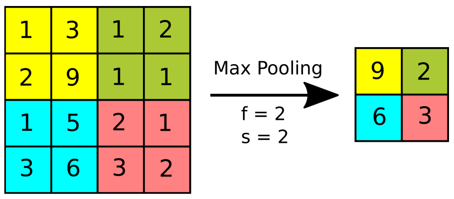


Figure 3.3 The max pooling operation applied by sliding a small matrix of size $f \times f$ across the input image with a stride s. [6]

3.1.5 Fully Connected Layer

The fully connected (FC) layer in a general neural network architecture is responsible for mapping the input representation to the output representation. It achieves this by establishing full connectivity between all the neurons in the preceding and succeeding layers, similar to a regular fully connected neural network.

Each neuron in the FC layer receives input from all the neurons in the preceding layer, and its output is then connected to every neuron in the succeeding layer. This means that the FC layer computes its output by performing a matrix multiplication between the input and weight matrices, followed by the addition of a bias term. [3, 4]

3.2 Residual Networks

This section introduces the issue with deep neural networks and presents a solution to this problem, which is based on the ResNet architecture. This section is based on [1, 7].

3.2.1 The Problem of Deep Neural Networks

The trend in neural networks has been to make them deeper, with the later models containing over a hundred layers, in comparison to the earlier models with only a few layers. This has allowed neural networks to represent increasingly complex functions.

However, a major problem associated with very deep neural networks is that the gradient signal, which is used to optimize the model during training, can vanish very quickly. This occurs during *backpropagation*[3], when we are multiplying by the weight matrix on each step. If the gradients are small, due to a large number of multiplications, the gradient can decrease exponentially quickly to zero, which makes gradient descent training extremely slow or even impossible.

3.2.2 Residual Block

The Residual Block is a solution to the problem of deep neural networks by allowing the network to fit the residual mapping, which is the difference between the original mapping and its approximation, rather than the original mapping itself.

The approach involves creating a *shortcut* or a *skip connection* that facilitates the flow of information between layers in a convolutional neural network. This is achieved by allowing data to bypass the normal flow of the CNN from one layer to the next layer after the immediate next.

There are two important things to keep in mind with residual blocks. First, adding more layers will not hurt the performance because unnecessary layers will be ignored during regularization. Second, if new layers do improve the model, even with regularization, the weights of those layers will be nonzero and the performance of the model may improve slightly.

ResNets use two types of blocks 3.4 depending on whether the input and output dimensions are the same or different. The *identity block* is used when the input and output activations have the same dimension, while the *convolutional block* is used when the dimensions do not match up, and it includes a convolutional layer in the shortcut path. [1, 7]

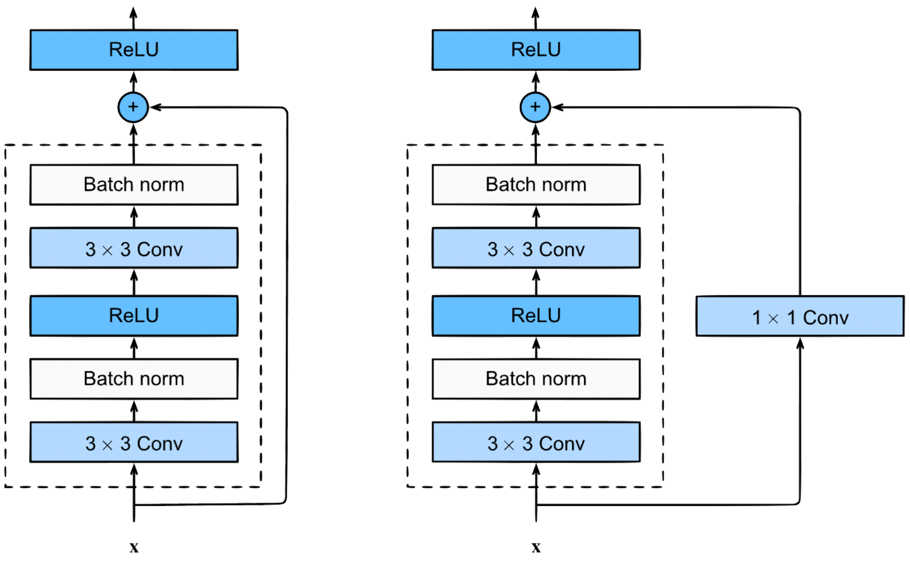


Figure 3.4 The identity block and the convolutional block are positioned respectively on the left and right sides. [8]

[3]Backpropagation is a common method used to train neural networks by computing the gradients of the loss function with respect to each parameter in the network.

3.3 Feature Pyramid Networks

This section provides an overview of the FPN approach for feature extraction in object detection. FPN is a feature extraction model that was developed with accuracy and speed as the main goals. It is based on the pyramid concept and is designed to generate multiple feature map layers, also known as *multi-scale feature maps*, that contain higher quality information compared to traditional feature pyramids for object detection. This improvement in feature extraction is crucial for accurate object detection, particularly for small objects, and has resulted in state-of-the-art performance on benchmark datasets. This section is based on [2, 9].

3.3.1 Data Flow

The Feature Pyramid Network is made up of two pathways, one that goes from *bottom-up* and one that goes from *top-down* as shown in Figure 3.5. The bottom-up pathway is a standard convolutional network 3.1 used for feature extraction, where the spatial resolution decreases as we move upwards, and higher-level structures are detected, leading to an increase in the *semantic value*[4] of each layer.

The purpose of the top-down pathway is to create higher-resolution layers from a semantic-rich layer. The reconstructed layers may not have precise object locations because of all that downsampling and upsampling, so lateral connections are added between reconstructed layers and corresponding feature maps to improve detection accuracy. These connections also serve as skip connections, which simplify the training process similar to ResNet 3.2.2.

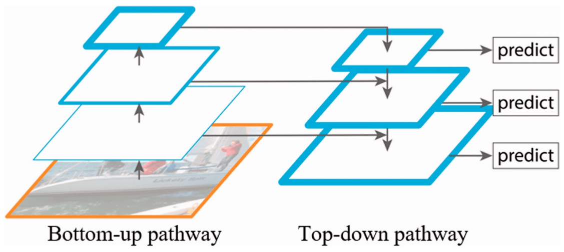


Figure 3.5 The Feature Pyramid Network architecture overview. [10]

3.3.2 Bottom-up Pathway

The bottom-up pathway is built with ResNet 3.2, and it consists of several convolution modules, each containing multiple convolution layers. While progressing upwards, the spatial dimension is decreased by half. The top-down pathway uses the output of each convolution module.

3.3.3 Top-down Pathway

To create the initial feature map layer for object prediction, we use a 1×1 convolution filter to reduce the channel depth of $C5$ to 256-d, resulting in $M5$, referring to the Figure 3.6. In the top-down path, we upsample the previous layer by 2 via nearest neighbors upsampling and then use a 1×1 convolution on the corresponding feature maps in the bottom-up path before adding them element-wise. Finally, we apply a 3×3 convolution to all merged layers. [2, 9]

[4]Level of abstract information about the detected objects.

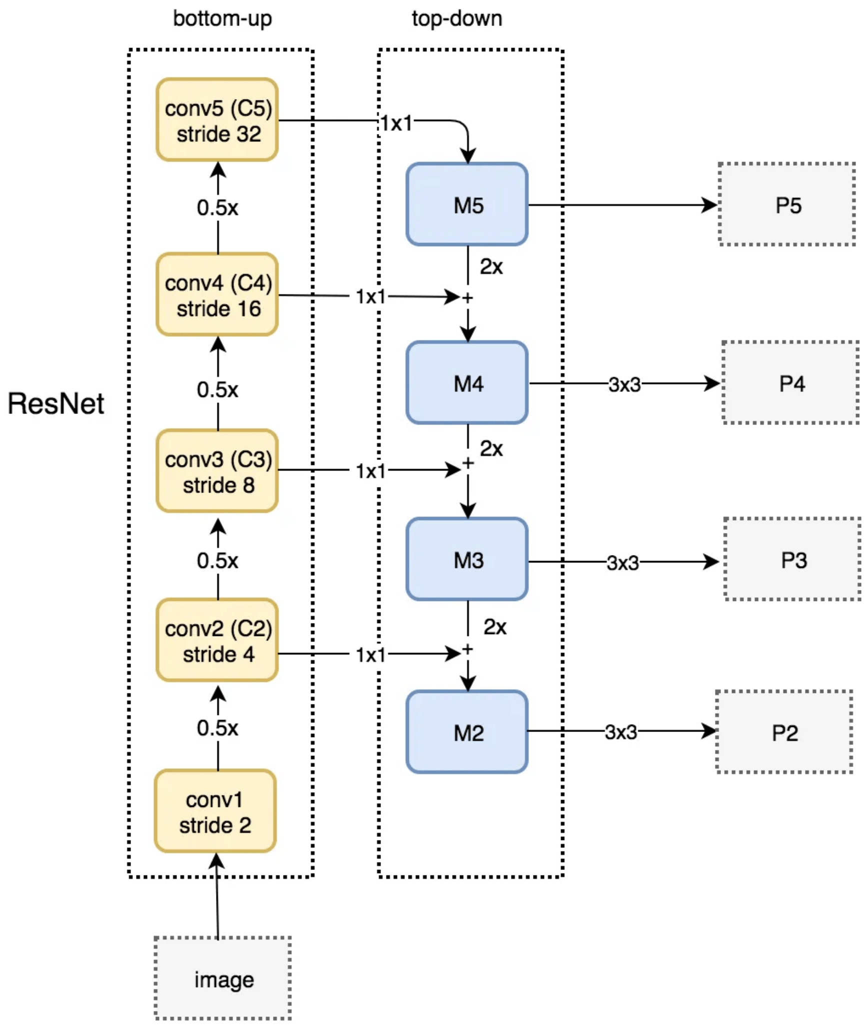


Figure 3.6 The bottom-up and top-down pathway operations of Feature Pyramid Network with lateral connections. [9]

Chapter 4
Related Work

Scene text detection has made significant advancements in recent years due to the progress made in deep learning techniques. Numerous methods have been proposed to address this challenge, each with a different approach focusing on different aspects of the problem. This chapter examines existing research on text detection techniques, with a particular focus on three general categories. The primary categories of deep learning-based methods are the *segmentation-based* methods and the *regression-based* methods. *Connected component (CC)* based methods, on the other hand, are commonly categorized as traditional computer vision methods. These categories represent different strategies for localizing text regions in an image, and each has its strengths and weaknesses.

4.1 Segmentation-based Methods

Segmentation-based methods generally aim to predict a pixel-wise segmentation map that identifies the locations of text instances in an image. The segmentation map is generated by analyzing the features extracted from the input image using a CNN. These methods can effectively handle variations in text size, style, orientation, and background, which makes them suitable for text detection in complex scenes. For instance, they can accurately segment text instances even in scenes with overlapping or curved text, or when text is partially occluded by other objects. They are also robust to changes in lighting, resolution, and distortion.

The PixelLink [11] algorithm directly predicts the pixel-level links between adjacent text regions in an image. The method works by dividing the text detection problem into two subtasks, *text region proposal* and *text region linking.* In the first step, the method generates a set of text region proposals using a simple thresholding technique on the output of the network. Then, in the second step, the method links these proposals using the predicted pixel-level links.

TextSnake [12] proposes a new representation for curved text instances in natural scenes. Instead of rectangular bounding boxes or quadrilaterals, it represents text instances as *snakes*, which are defined by a series of straight line segments and corresponding curvature parameters. After predicting the coordinates of the snake points, it optimizes the snake representation using a differentiable polygon rendering algorithm. The method also uses a set of post-processing steps same as PixelLink.

TextFuseNet [13] uses a feature fusion strategy to combine multi-level features from different layers of the network, which enables it to capture both fine-grained and high-level information for improved text detection performance. Additionally, it includes a post-processing step to refine the detection results and remove false positives.

The TextField [14] text detector utilizes an adversarial loss function at the instance level to improve the accuracy of predicted bounding boxes. The method further employs a multi-scale testing strategy to handle a text of different scales and resolutions.

DBNet++ [15] extends upon the original DBNet architecture by incorporating a recurrent neural network to refine the segmentation results and improve the recognition accuracy. It also includes a bounding box refinement module to further improve the localization of text regions.

4.2 Regression-based Methods

In regression-based methods, the text detection task is formulated as a regression problem where the aim is to predict the bounding box coordinates and attributes directly from the input image. In contrast to segmentation-based methods, which typically require additional post-processing steps to obtain the final bounding boxes, regression-based methods directly predict the bounding box coordinates in a single step. This makes them faster than segmentation-based methods, and more efficient in detecting regular text with clear boundaries, but they may struggle with detecting irregular or curved text due to their reliance on predefined bounding box shapes.

TextBoxes++ [16] incorporates an FPN-like architecture that aggregates feature maps across different resolutions to handle text instances of varying scales and aspect ratios. The method employs a novel aspect ratio refinement module that helps to refine the aspect ratios of text boxes and improve the detection performance of the network. In addition, Textboxes++ utilizes a positive-negative ratio balancing strategy during training to address the issue of class imbalance, resulting in better detection performance on small text instances.

The EAST [17] network consists of a feature extraction stage followed by a two-branch sub-network that outputs the *score map* and *geometry map* for each position in the input image. The score map represents the probability of a pixel belonging to the text region, while the geometry map encodes the four coordinates of the bounding box and the angle of rotation.

LOMO [18] presents a module for refining bounding box proposals for long texts in an iterative manner. The proposed module predicts center line, text region, and border offsets for rebuilding the text instance.

4.3 CC-based Methods

Connected Components-based methods segment text by grouping together pixels that share the same color or intensity values. These pixels belong to the same connected component. This is achieved by thresholding the image to binarize it and then identifying connected components using techniques such as flood filling[1]. One of the main limitations of CC-based methods is that they can be sensitive to noise and other artifacts in the image, leading to false positive detections.

SegLink [19] decomposes text into *segments* and *links*, where the link joins neighboring segments that form a single word. The upgraded verison SegLink++ [20] replaces the original backbone with an improved version of ResNet-50, and incorporates several modifications to enhance text detection accuracy.

CRAFT [21] use a single-shot detector that predicts character regions rather than text regions. The network is designed to recognize each character separately and then assemble them into words or text lines.

[1]Process of coloring connected pixels with the same color starting from a seed pixel and expanding to adjacent pixels.

Chapter 5
Datasets Analysis

This chapter first describes the Nomenclature dataset [22] and presents the challenges associated with this dataset. In the second part, the widely used TotalText dataset [23] is briefly introduced.

5.1 Nomenclature Dataset

The Nomenclature Dataset is designed to identify and recognize handwritten nomenclatures on historical cadastral maps. A nomenclature is a text that provides information about the position of an individual map sheet in a grid that covers a larger region. The dataset comprises a total of 800 map sheets which have been split into three parts for *training*, *testing*, and *validation*. The training set includes 650 sheets, while the testing set has 100 sheets, and the validation set includes 50 sheets. [24]

5.1.1 Data Description

The dataset includes files in two formats, namely `.txt` and `.jpg`. The `.jpg` files represent the input images, with a typical map image being shown in Figure 5.1. The `.txt` files provide annotations that are considered as the ground-truth. These annotations are tab-separated values that typically have the following format:

$$x_{min} \quad y_{min} \quad x_{max} \quad y_{max} \quad label \quad type$$

where the first four values specify the coordinates of an axis-aligned bounding box. The text enclosed within this bounding box is captured as the *label*, including any new line characters, which can complicate the parsing process. The *type* indicates the type of text. There are three types of text labels that can be identified. The first is *Handwritten (H)* and includes general toponyms such as forests, roads, hills, swamps, and rivers. These are shown in the bottom-right bounding box of Figure 5.1. The second type is *Printed (P)* and includes municipal toponyms like cities, villages, and significant buildings. These are shown in the bottom-left bounding box of Figure 5.1. The last type is *Nomenclature (N)* which is always located in the top-right corner of the map sheet and indicates the location of the map in the coordinate system. This is also shown in Figure 5.1. [22]

There are three ground-truth representations used in this thesis.

1. The *axis-aligned bounding box* representation is composed of four values. Usually, these values indicate the coordinates of the top-left and bottom-right corners of a rectangle that is aligned with the x and y axes. The Nomenclature dataset uses this representation.

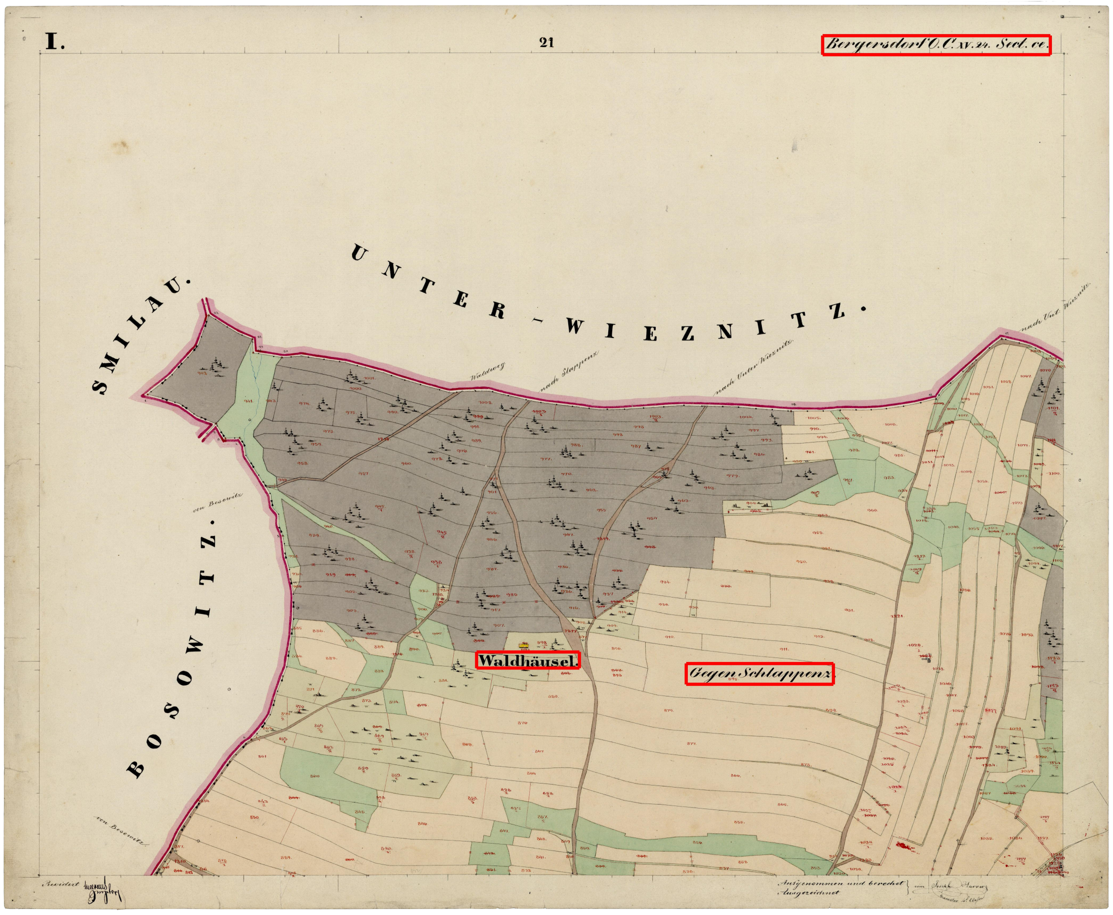


Figure 5.1 This is an illustration of a historical map that showcases all three label types. The red bounding boxes indicate the ground-truth regions. [24]

2. The *oriented bounding box* representation is comprised of eight values. Each value denotes one corner of a rectangle that can be rotated with respect to the coordinate axes.

3. The *polygon* representation is composed of coordinates for the vertices, typically listed in an array. The order in which these vertices are listed determines the sequence of the edges of the polygon. This representation provides more detailed and accurate shape information than bounding box representations, especially for complex and irregularly-shaped objects, such as text instances.

5.1.2 Toponym Box Limitations

The most challenging aspect is accurately distinguishing between place names and non-place name text on the map. It should be noted that even in Figure 5.1, some words do not qualify as place names. In addition, the labeling in the Nomenclature dataset lacks precision, containing a greater number of non-textual pixels than necessary. This inaccuracy can be partly attributed to the use of an axis-aligned representation for the ground-truth. Here are some examples of these inaccuracies:

- Certain text instances on the map may not be horizontally aligned, which poses a challenge when describing them using an axis-aligned bounding box. This results in the text instance being confined to the diagonal section of the rectangle, which results in a significant number of non-textual pixels. This is evident from Figure 5.2.

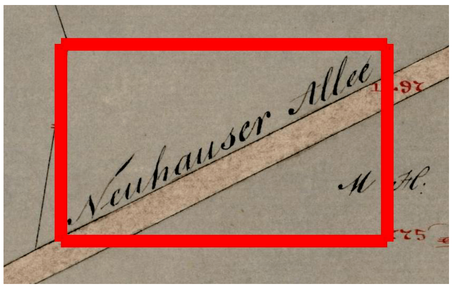


Figure 5.2 This figure refers to the inaccuracy of representing oriented text instances through the use of an axis-aligned bounding box.

- Typically, written text features characters that are vertically aligned, with certain characters extending below the baseline. However, since the axis-aligned bounding box must be rectangular in shape, the presence of a single descender character can significantly expand the entire ground-truth. This expansion leads to a significant number of non-textual pixels being included in the ground-truth. This is illustrated in Figure 5.3

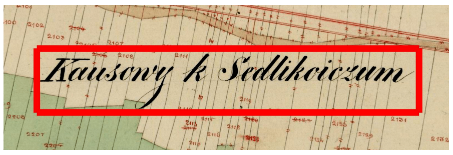


Figure 5.3 This figure refers to the extension of the entire ground-truth due to the inclusion of a descender character “y”.

Figures 5.2 and 5.3 show inaccuracies that could be resolved by using a polygon representation. However, it is important to note that even the ground truth data itself may contain inaccuracies, as demonstrated in Figure 5.4, where the middle three instances exhibit significant imprecision.

5.2 TotalText Dataset

The TotalText dataset [23] is a scene text dataset that features a diverse range of natural scene images with annotations for text detection and recognition tasks. One of the key characteristics of this dataset is its inclusion of multi-oriented and curved text instances that can appear in any direction and shape.

The TotalText dataset contains a total of 1 555 images of natural scenes, including street views, shop windows, indoor environments, and more. The images are collected from various sources, such as the Google Street View and other publicly available repositories. Each image is annotated with ground truth information.

The presence of multi-oriented and curved text instances in the TotalText dataset also reflects the real-world scenarios where text can appear in a variety of shapes and orientations, such as on billboards, road signs, and product packaging.

To handle the variability in text appearance, the TotalText dataset provides annotations in two forms: *oriented bounding boxes* and *polygons*, as described in Section 5.1.1. The oriented bounding boxes are used to represent the text instances with a minimum rotated rectangle that tightly encloses the text. The polygons are used to represent the text instances more accurately,

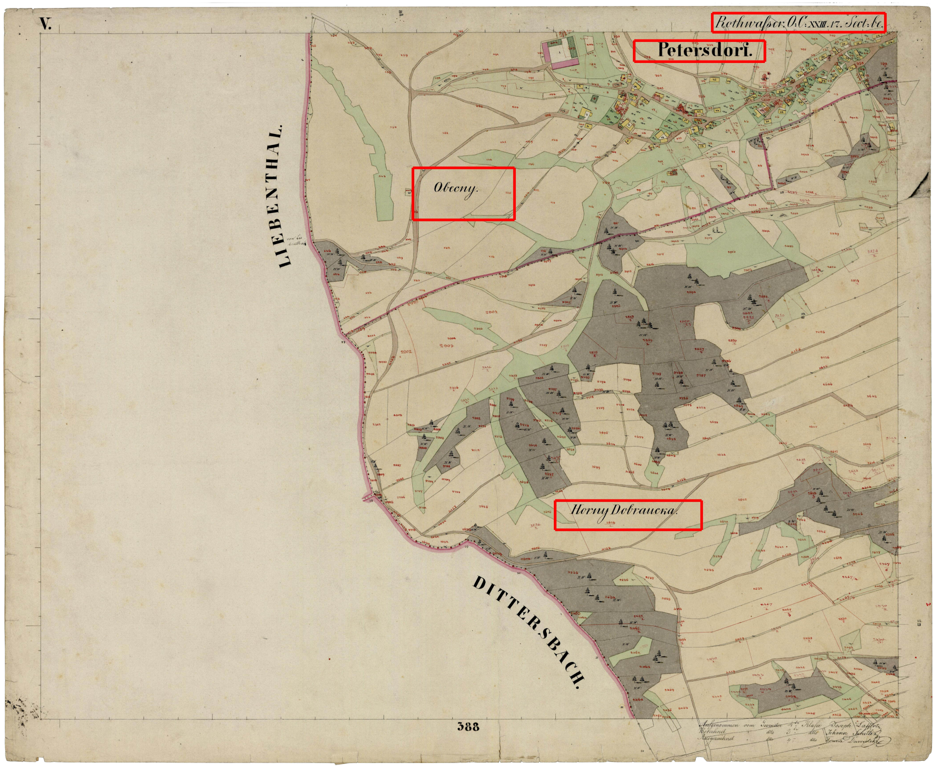


Figure 5.4 Illustration of highly imprecise ground truth annotations.

especially in cases where the text is curved or has irregular shapes. Figure 5.5 illustrates a typical example of an image within this dataset that features both representations.

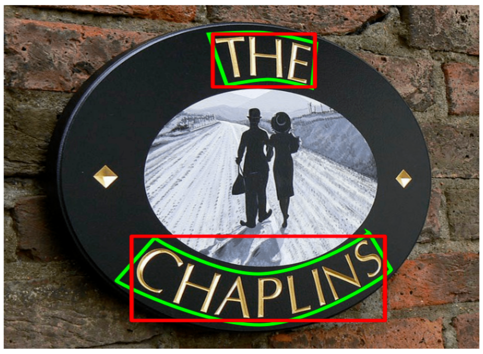


Figure 5.5 Illustrative example of the TotalText dataset. The green contours correspond to the *Polygon* representation and the red contours represent the *Oriented Bounding Box* representation. [23]

Chapter 6

Methodology

This chapter introduces two text detection methods that were chosen for implementation and application to the Nomenclature Dataset[1]. The *Progressive Scale Expansion Network (PSENet)*[25] is a recently proposed and widely used method that was selected for its robustness and stability. The *Text Detection with Probability Maps (TextPMs)* [26] is a novel approach that has shown promising performance and potential. These methods achieved an F-measure of 80.9% and 88.79% on the TotalText dataset[2], respectively. In addition, this chapter details the training process and evaluation used in the Experiments 7 chapter.

6.1 Progressive Scale Expansion Network

PSENet is a deep learning-based text detection algorithm that was proposed in 2019 by Wang et al. The algorithm is designed to handle text in various shapes and orientations, making it robust to the challenges posed by natural scene images.

PSENet uses a *progressive scale expansion* strategy to gradually expand the detected text regions. The network generates a set of text region proposals at each scale, which are then merged and refined in a post-processing step to obtain the final detection results. This section is based on [25].

6.1.1 Network Design

Figure 6.1 presents an overview of the PSENet model, which employs ResNet 3.2 as its backbone. The model combines low-level texture features with high-level semantic features, which are fused into feature map F to encode information from various receptive fields. Subsequently, F is projected into n branches to generate several segmentation outcomes, each corresponding to a segmentation mask for all text instances at a specific scale. The scale of each segmentation mask is determined by hyper-parameters. S_1 indicates the segmentation result for the text instances with the smallest scales, while S_n denotes the original segmentation mask with the largest kernels. The final detection results R are produced by applying the *progressive scale expansion algorithm* 6.1.3, which gradually expands the kernels of all instances from S_1 to their full shapes in S_n.

[1]https://corpora.kiv.zcu.cz/nomenclature/
[2]https://github.com/cs-chan/Total-Text-Dataset

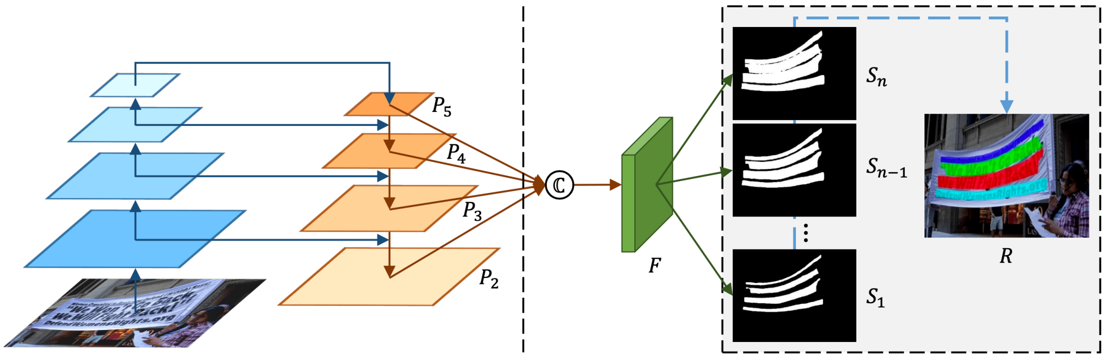


Figure 6.1 The PSENet overall pipeline, with the FPN on the left and the feature fusion followed by Progressive Scale Expansion Algorithm 6.1.3 on the right. [25]

6.1.2 Backbone

The PSENet model is based on the FPN architecture discussed in Section 3.3. Initially, four feature maps (P_2, P_3, P_4, and P_5) are generated by the FPN backbone, each with different channel sizes of 128, 256, 512, and 1024, respectively. These feature maps are merged to create a new feature map F, with 1024 channels that combine semantic features from low to high levels. To achieve this, a function $\mathbb{C}(\cdot)$ is used to combine the four feature maps as follows:

$$F = \mathbb{C}\left(P_2, P_3, P_4, P_5\right) = P_2 \parallel \mathrm{Up}_{\times 2}(P_3) \parallel \mathrm{Up}_{\times 4}(P_4) \parallel \mathrm{Up}_{\times 8}(P_5), \tag{6.1}$$

where "$\parallel$" means concatenation and $\mathrm{Up}_{\times 2}(\cdot)$, $\mathrm{Up}_{\times 4}(\cdot)$, and $\mathrm{Up}_{\times 8}(\cdot)$ indicate 2, 4, and 8 times upsampling, respectively. Afterwards, the input F goes through Convolution-BN-ReLu layers with a 3×3 filter, which reduces it to 256 channels. Then, it undergoes n layers of Convolution with a 1×1 filter, followed by upsampling and Sigmoid activation, resulting in n segmentation outputs labeled S_1, S_2, $\ldots$, S_n.

6.1.3 Progressive Scale Expansion Algorithm

It is often difficult for segmentation-based methods to accurately separate text instances that are located closely together. To address this challenge, a progressive scale expansion algorithm has been developed as a solution.

In Figure 6.2, a clear example is provided to illustrate the progressive scale expansion algorithm, which is derived from the *Breadth-First-Search (BFS)* algorithm. In the given example, three segmentation results $S = \{S_1, S_2, S_3\}$ are presented in Figures 6.2 (a), (e), and (f). Initially, the minimal kernels' map S_1 is used to obtain four distinct connected components $C = \{c_1, c_2, c_3, c_4\}$ as initializations. These connected components are represented by different colors in Figure 6.2 (b). By doing so, we can detect all the central parts of the text instances, i.e., the minimal kernels. Next, we gradually expand the detected kernels by merging the pixels in S_2 and then in S_3. The scale expansions' results are displayed in Figures 6.2 (c) and (d), respectively. Finally, the connected components marked with different colors in Figure 6.2 (d) are extracted as the text instances' final predictions.

The scale expansion process is demonstrated in Figure 6.2 (g). The algorithm starts from the pixels of multiple kernels and combines adjacent text pixels in iterations, similar to BFS. Some pixels may be in conflict during expansion, as highlighted by the red box in Figure 6.2 (g). In practice, the principle for dealing with these conflicts is to merge the conflicting pixel with only one single kernel based on a first-come-first-served basis. Thanks to the "progressive" nature of the expansion procedure, these boundary conflicts will not significantly affect the final detection results or the performance.

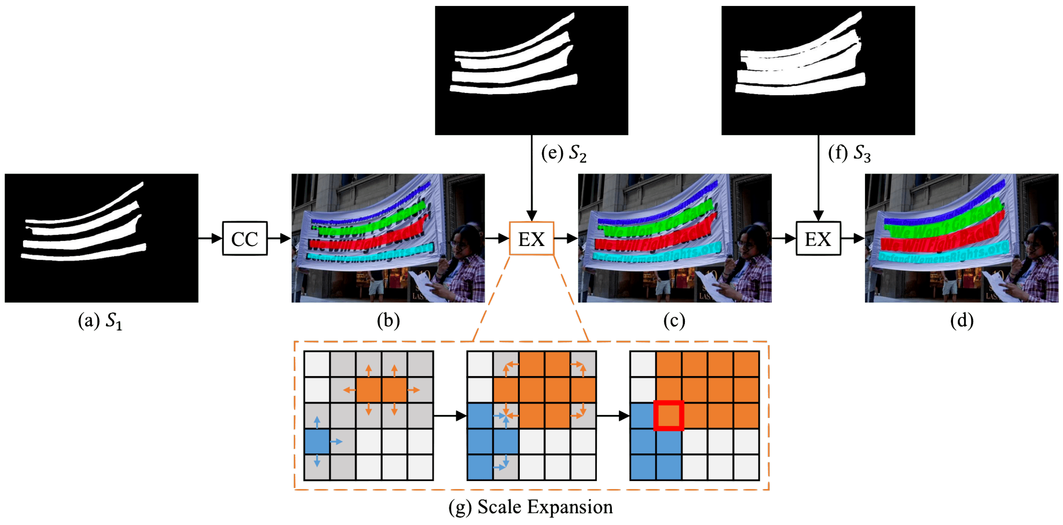


Figure 6.2 The figure depicts the progressive scale expansion algorithm, which includes two functions: CC and EX. CC is used to find the connected components, while EX is responsible for the scale expansion. [25]

6.1.4 Kernel Scales

As depicted in Figure 6.1, PSENet generates various segmentation outcomes, such as S_1, S_2, ..., S_n utilizing multiple kernel scales. Therefore, it requires ground-truth labels corresponding to different kernel scales for successful training. In practical scenarios, these labels can be obtained easily and effectively by shrinking the original text instance. In Figure 6.3 (b), the original text instance is represented by a polygon with a blue border, which corresponds to the largest segmentation label mask as shown in the rightmost map of Figure 6.3 (c). To obtain the shrunk masks presented sequentially in Figure 6.3 (c), the Vatti clipping algorithm [27] is used to shrink the original polygon p_n by d_i pixels and obtain the shrunk polygon p_i as shown in Figure 6.3 (a).

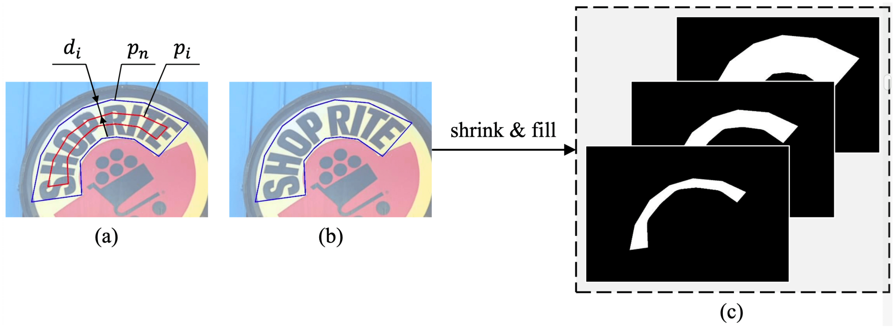


Figure 6.3 The illustration of how labels are created. [25]

After obtaining the shrunk polygon p_i, it is converted into a binary mask for the segmentation label ground-truth. The resulting ground-truth maps are denoted as $G_1, G_2, \ldots, G_n$ respectively. Mathematically, if we consider the scale ratio as r_i, the margin d_i between p_n and p_i can be calculated as:

$$d_i = \frac{\mathrm{Area}\,(p_n) \times \left(1 - r_i^2\right)}{\mathrm{Perimeter}\,(p_n)} \tag{6.2}$$

Moreover, the ground-truth map G_i is related to a scale ratio r_i, which is defined as:

$$r_i = 1 - \frac{(1-m)\times(n-i)}{n-1}, \tag{6.3}$$

where m is the minimal scale ratio within the range of (0, 1]. Equation 6.3 defines the scale ratios $r_1, r_2, \ldots, r_n$, which are determined by two hyper-parameters n and m. These scale ratios linearly increase from m to 1.

6.1.5 Loss Function

To train PSENet, the loss function L can be expressed as follows:

$$L = \lambda L_c + (1-\lambda)L_s, \tag{6.4}$$

where L_c denotes the loss for complete text instances, L_s denotes the loss for shrunk instances, and λ is a balancing parameter that determines the relative significance of L_c and L_s.

The authors of PSENet decided to employ the dice coefficient in their experiments. The dice coefficient, denoted as $D(S_i, G_i)$, is defined as follows:

$$D\left(S_i, G_i\right) = \frac{2\sum_{x,y}\left(S_{i,x,y}\times G_{i,x,y}\right)}{\sum_{x,y} S_{i,x,y}^2 + \sum_{x,y} G_{i,x,y}^2} \tag{6.5}$$

Equation 6.5 computes the dice coefficient, which is derived from the values of pixels (x, y) in the segmentation result, $S_{i,x,y}$, and the corresponding ground-truth, $G_{i,x,y}$. Additionally, there are various structures that resemble text strokes, such as fences and lattices. In order to better discriminate between these structures, the authors utilized *Online Hard Example Mining (OHEM)* [28] during training to improve the performance of L_c. The primary goal of L_c is to segment the text and non-text areas. Specifically, if the training mask generated by OHEM is denoted as M, then L_c is formulated according to Equation 6.6.

$$L_c = 1 - D\left(S_n \cdot M, G_n \cdot M\right) \tag{6.6}$$

The loss function for the shrunk text instances denoted as L_s, excludes the pixels of the non-text region in the segmentation result S_n to avoid redundancy. This is because the shrunk instances are within the original areas of the complete text instances. The formulation of L_s 6.7 takes into account a mask, W, that disregards the non-text region pixels in S_n. $S_{n,x,y}$ refers to the value of the pixel (x, y) in S_n. [25]

$$L_s = 1 - \frac{\sum_{i=1}^{n-1} D\left(S_i \cdot W, G_i \cdot W\right)}{n-1}; \quad W_{x,y} = \begin{cases} 1, & \textit{if } S_{n,x,y} \geq 0.5 \\ 0, & \textit{otherwise} \end{cases} \tag{6.7}$$

6.1.6 Hyperparameters

Since PSENet uses a neural network, there are some common hyperparameters that are shared among them:

Learning rate Determines the step size at each iteration while moving toward a minimum of a loss function during training.

Momentum Controls the contribution of the previous updates to the current update of the model weights.

Weight decay Regularization technique that reduces the magnitude of weights during the training process to prevent overfitting.

Number of epochs Number of times the entire dataset is passed through the model. In other words, one epoch is a complete pass via the entire dataset during training.

Optimizer Algorithm that adjusts the parameters of the model during training to minimize the loss function. There are various optimizers, such as *Stochastic Gradient Descent (SGD)* and *Adaptive Moment Estimation (Adam)*, which differ in how they update the weights. SGD calculates the gradient of the loss function and updates the weights using a fixed learning rate, while Adam adjusts the learning rate adaptively according to the history of gradients.

Batch size Number of samples from the dataset used in one iteration of the training process.

Furthermore, there exist multiple hyperparameters which are specific to PSENet:

Lambda parameter in loss function (λ) The balancing parameter, as discussed in 6.4.

Number of kernels (n) Number of segmentation results denoted as $S_1, S_2, \ldots, S_n$ in Figure 6.1.

Minimal kernel scale (m) The minimal scale ratio employed in Equation 6.3.

Bounding Box Type Whether to use an *oriented bounding box* or a *polygon* representation for the final output. Both representations are explained in Section 5.1.1.

6.2 Text Detection with Probability Maps

The TextPMs method, proposed by Shi-Xue Zhang et al. in 2022, is a novel segmentation-based text detection method that uses a group of probability maps to accurately detect text instances by describing possible probability distributions. It uses a *Sigmoid Alpha Function (SAF)* to transfer the distances between boundaries and their inside pixels to a probability map and adopts an iterative model to predict and assimilate probability maps to reconstruct text instances. This section is based on [26].

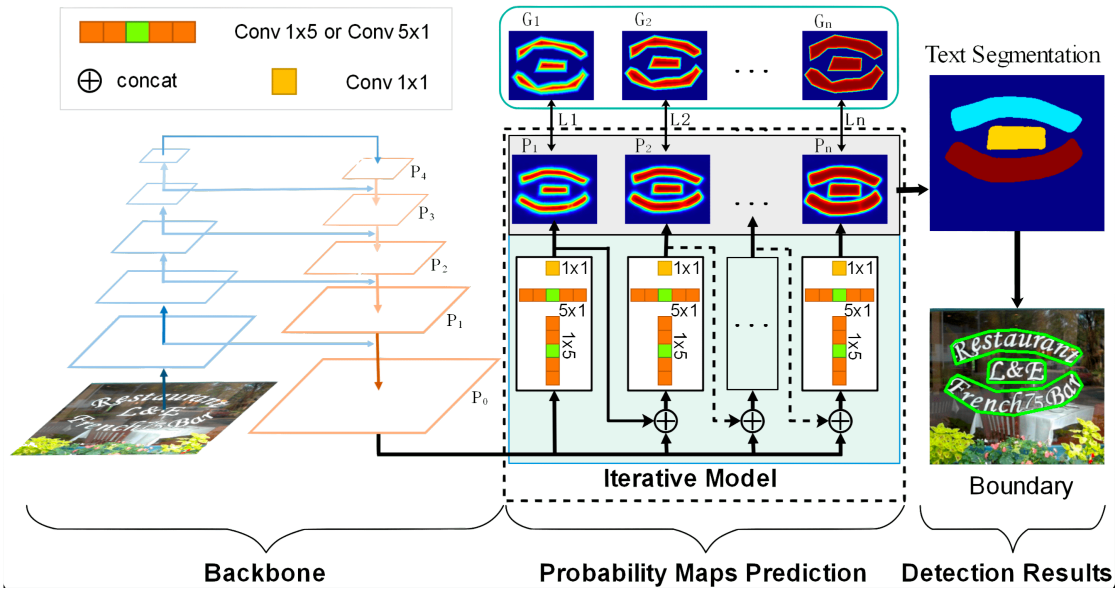


Figure 6.4 Overall TextPMs architecture with FPN on the left followed by Iterative Module on the right. [26]

6.2.1 Architecture Overview

The TextPMs method uses a ResNet-50 model to extract image features and a multi-level feature fusion strategy to preserve spatial resolution and capture multi-level information. The backbone of this architecture is the *Feature Pyramid Network (FPN)* as described in Section 3.3. An *iterative module* is then used to predict probability maps that describe the possible probability distributions of text pixels within a boundary, which provide enough information for reconstructing text instances. Different region-growing algorithms are used to aggregate the computed probability distribution maps into candidate text instances, and an adaptive voting filtering algorithm is used to filter out false-positive text instances. An overview of the architecture is shown in Figure 6.4.

6.2.2 Probability Maps

Many scene text detection datasets only provide rough annotations of text boundaries due to the high cost of pixel-level annotations. These annotations are then used to create pixel-level binary supervision masks for text instance segmentation in most segmentation-based methods. However, the quality of these masks is often poor, with many background pixels being incorrectly categorized as text pixels due to inaccurate annotations around text boundaries. This results in false-positive text instances in the trained detectors. Some segmentation-based methods attempt to reduce or avoid this issue through distance or direction fields, but the inaccurate annotations still significantly degrade the detection performance. This problem is illustrated in Figure 6.5 (c).

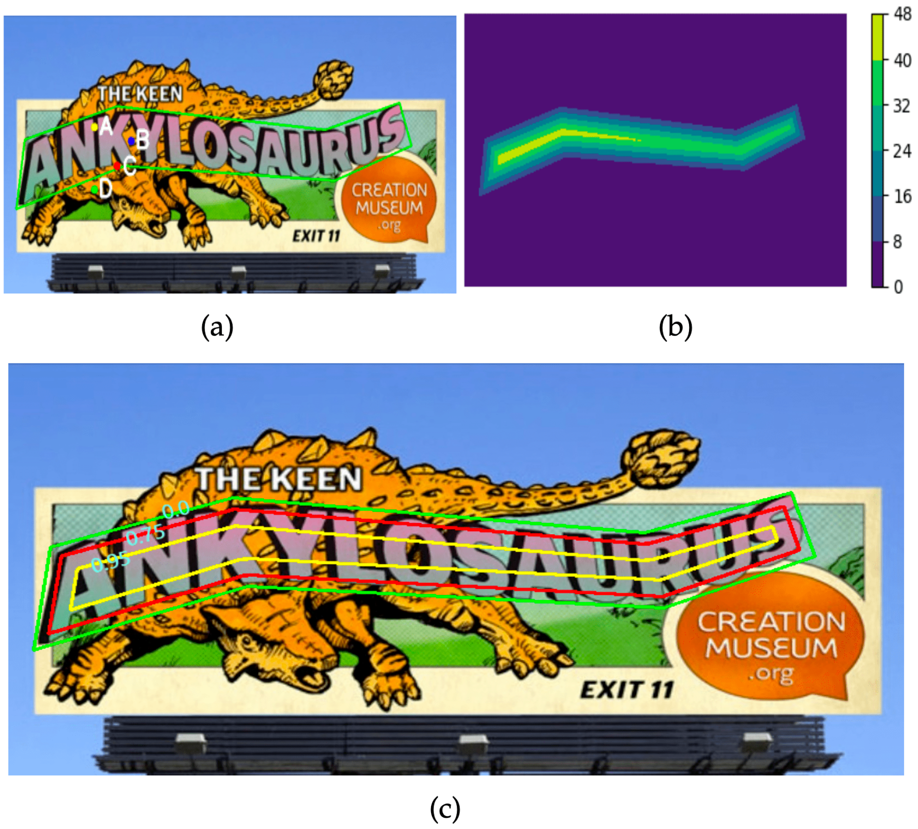


Figure 6.5 This figure demonstrates the correlation between the distance to the boundary and the probability distribution. [26]

In this method, probability maps are proposed to address the ambiguity in text annotations explicitly. Generally, the likelihood of a pixel being a text pixel is determined by its distance to the boundary. As depicted in Figure 6.5 (a), the yellow point (A) located farther away from the text boundary has a higher probability of being a text pixel than the red point (C) located close to the boundary. On the other hand, even though the blue point (B) may resemble a background pixel based on color and texture features, the probability of it being predicted as a part of a text instance is still high in existing text detection methods. This is because point B is surrounded by text pixels and located far away from the text boundary.

The primary objective of this method is to determine text boundaries by considering the probability of the entire boundary instead of a single pixel. In this approach, it is assumed that pixels on the same contour have the same probability, and contours closer to the annotation boundary have smaller probability values. Figure 6.5 (c) shows a green annotation boundary, mainly located in the background region, with background noise inside the boundary, especially near the boundary (point C in Figure 6.5 (a)). The red and yellow contours, shown in Figure 6.5 (a), have the same distance to the text boundary (green contour), and pixels on the red contour are more likely to be a text instance than those on the green contour, whereas pixels on the yellow contour are more likely to be text than those on the red and green contours. By computing the distance to the annotation boundary, a series of contours with pixels at the same distance to the annotation boundary can be obtained, as shown in Figure 6.5 (b). Typically, the closer a contour is to the boundary, the lower the proportion of text pixels. Assuming the proportion of text pixels on a contour represents pixel probability, pixels on the same contour have the same probabilities, enabling the establishment of the relationship between the probability of a pixel and its distance to the boundary. The objective of the text detection task is to identify a contour where the probability of pixels has a minimum value.

6.2.3 Sigmoid Alpha Function

The probability map is constructed based on the distance between pixels and the annotation boundary to represent the likelihood of a pixel belonging to a text region. To establish a relationship between pixel probability and distance to the boundary, a *Sigmoid Alpha Function (SAF)* is proposed, which is based on the *Sigmoid Function* [29]. The SAF is beneficial as it maps distance values into the range [0,1], which can be interpreted as a pixel probability. As a result, the SAF is applied to map a distance map (D) into a probability map. The definition of the Sigmoid Alpha Function is:

$$SAF_{(i,j)} = C * \left(\frac{2}{1 + e^{\frac{-\alpha D_{(i,j)}}{L}}} - 1 \right); \quad C = \frac{1 + e^{-\alpha}}{1 - e^{-\alpha}}; \quad \alpha \in (0, \infty) \tag{6.8}$$

where parameter α can be used to control the shape of the probability distribution maps, while the constant C is responsible for keeping the output values within the range of [0, 1]. The indices (i, j) represent the coordinates of a particular pixel in the map, and $D_{(i,j)}$ represents the shortest distance from that pixel to the boundary. The distance map D is the set of all these shortest distances $D_{(i,j)}$. Lastly, the variable L denotes the scale of the text instance, and it is defined as:

$$L = max\left(D_{(i,j)}\right); \quad i, j \in T \tag{6.9}$$

where T denotes a text instance. By incorporating information about the size of text instances into probability maps through the parameter L in the SAF, the model is better equipped to detect text of different sizes, increasing its adaptability.

Due to the uncertainty and inaccuracy of annotation boundaries, a single probability map cannot accurately describe the probability distributions of text pixels inside a boundary. To address this, a series of probability distribution maps are employed as supervision to segment text instances. Based on observations and experiments, a group of probability maps can account for possible probability distributions that cover annotation errors and provide sufficient information for reconstructing text instances. Multiple probability maps improve the robustness of the model for final detections as other probability maps can provide supplementary context predictions if a local prediction of one probability map is inaccurate. Using Equation 6.8, a group of probability distribution maps can be obtained by controlling the hyperparameter α in SAF.

6.2.4 Generation of Probability Maps

The process of creating probability maps depends on the distance map D and SAF. For a given text image, a polygon representation is used to describe the region of each text instance. To generate labels for each instance, the shortest distance from each pixel (i, j) to the boundary is computed and stored as a set of $D_{(i,j)}$ values, that form the distance map for the text instance. As a result, the distance map D may be defined as follows:

$$D = \left\{D_{(i,j)}\right\}; \quad i, j \in T, \tag{6.10}$$

where T represents the text instance.

Once the distance map D is obtained, the sigmoid alpha function is applied with a series of α values $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ to convert the distance into probability. This procedure, as shown in Figure 6.6, results in a set of probability distribution maps. These probability distribution maps $(G_1, G_2, \ldots, G_n)$ can be described as:

$$G = \{G_k\}_{k=1}^{n} = \{SAF\,(D, \alpha_i) \mid i \in (1, 2, \ldots, n)\}, \tag{6.11}$$

where n corresponds to the number of α values. The set of α values $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is determined by a hyper-parameter. The probability maps G will act as the final ground-truths.

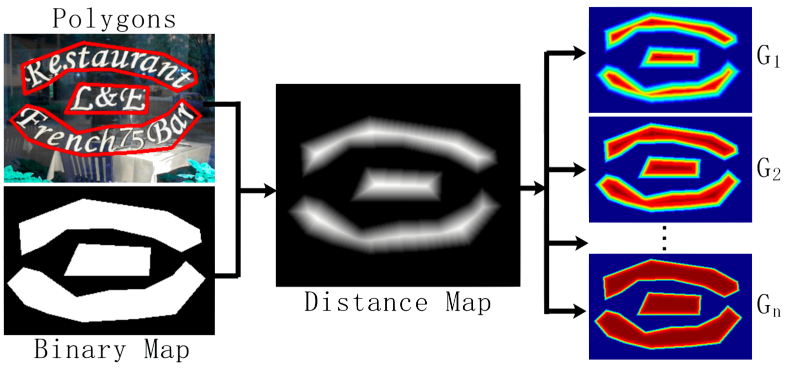


Figure 6.6 The process of generating probability maps. First, a distance map is created from text instances. Second, the distance map is fed into the SAF to generate a set of probability maps. [26]

6.2.5 Iterative Module

The paper presents a new approach called *iterative module (IM)*, which is designed to improve the accuracy of the probability maps generated in the earlier steps. The IM employs large-scale asymmetric convolution kernels based on text attributes. The structure of the module is shown in Figure 6.7. The IM can explore the mapping relationship between probability distributions to optimize the predictions of the probability maps. The information from low-level iterations can be fully utilized to enhance the accuracy of predictions at high-level iterations. Back-propagation gradients can be used to optimize the parameters in high-level iteration layers. Finally, the predictions of the IM serve as a series of probability maps $P_1, P_2, \ldots, P_n$.

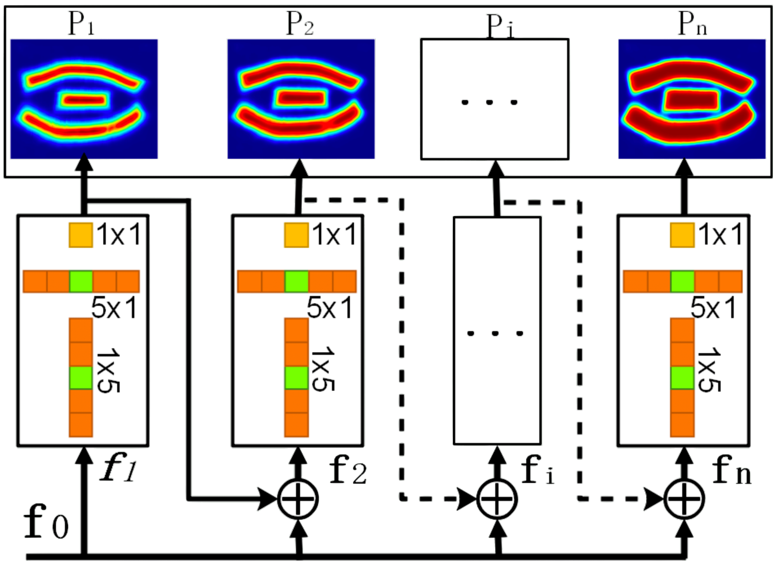


Figure 6.7 Detailed structure of Iterative Module. [26]

Scene text instances are different from generic object instances as they have varying sizes and aspect ratios and are often distributed in strips, particularly for long and line-level text instances. To tackle the challenge of significant variability in size and aspect ratio in scene text detection, two different approaches can be utilized. Firstly, text size information is encoded into probability maps using SAF. Secondly, an iterative model with a group of asymmetric convolutions is employed to improve the accuracy of detecting long and line-level text. Using these two approaches together, it is possible to accurately detect long text instances. In mathematical terms, the IM can be expressed as:

$$f_1 = \mathit{ConvBlock}\left(f_0\right); \quad P_1 = \sigma\left(f_1\right) \tag{6.12}$$

$$f_i = \mathit{ConvBlock}\left(f_0 \oplus f_{i-1}\right); \quad P_i = \sigma\left(f_i\right); \quad i \in [2, n] \tag{6.13}$$

The Equations 6.12, 6.13 utilizes the concatenation operation denoted by $\oplus$ and the *Sigmoid* activation function, discussed in Section 3.1.3, represented by $\sigma(\cdot)$. The symbol n represents the total number of iteration steps, which is determined by the number of alpha values set by the hyperparameter, the same as in Equation 6.11. This guarantees that the probability maps P are matched with the ground-truth maps G for the purpose of the loss function.

In summary, the process begins by obtaining fusion feature maps f_0 from the backbone with 16 channels. These feature maps f_0 are then fed into the first iterative layer, given by Equation 6.12, of the iterative module to produce a new feature map f_1, which is used to generate a probability map P_1 through the Sigmoid function. This probability map P_1 is then used as prior information for producing the second probability map P_2 in the next iteration layer, described by Equation 6.13. This process continues for several iterations to obtain multiple probability map predictions $P_1, P_2, \ldots, P_n$. After obtaining these probability distribution maps P from the trained network, region growth algorithms such as the *Watershed Algorithm* or *Progressive Scale Expansion Algorithm* can be applied to combine these probability maps into complete text instances.

6.2.6 Loss Function

Once the ground-truth probability maps G and predicted probability maps P have been obtained, as described in Sections 6.2.4 and 6.2.5, respectively, the loss for each pixel on the image domain

Ω can be calculated using the *Mean Squared Error (MSE)* as:

$$loss_i = \frac{1}{\Omega} \sum_{p \in \Omega} \|G_i(p), P_i(p)\|_2 \tag{6.14}$$

The Equation 6.14 calculates the MSE for each pixel in the image domain Ω by comparing the i-th ground-truth probability map G_i and the i-th predicted probability map P_i. Here, p represents a pixel in the image domain Ω. The network training focuses on distinguishing hard-to-classify pixels. To achieve this, the concept of *Online Hard Negative Mining (OHEM)* is utilized. Specifically, only pixels with a probability greater than 0.001 are considered, while those with "zero" probability are not taken into account.

The total loss L is obtained by a sum of all the individual losses $loss_i$ as:

$$L = \sum_{i=1}^{n} loss_i, \tag{6.15}$$

where i denotes the probability map number, and n is the total number of probability maps. [26]

6.2.7 Hyperparameters

This method shares common hyperparameters with neural networks, as outlined in Section 6.1.6. Additionally, it contains specific hyperparameters:

Set of α values The set of $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is used to construct different distributions of probability maps, as described in Section 6.2.4, Equation 6.11, and Section 6.2.5 Equation 6.13.

Recovery Algorithm Whether to use *Watershed Algorithm* or *Progressive Scale Expansion Algorithm* to recover complete text instances from probability maps.

6.3 Training Process

Both methods were implemented using the PyTorch [30] deep learning framework. The code was partially adapted from the official implementations of the PSENet[3] and TextPMs[4]. The models were trained using *stochastic gradient descent (SGD)* optimizer on a single NVIDIA A100-SXM4-40GB GPU.

The PSENet hyperparameters were chosen following the guidelines provided in the official PSENet paper. The initial learning rate was set at 0.001 and is divided by 10 every third of the training process, with a momentum of 0.99 and a weight decay of 5×10^{-4}. While the number of epochs varied depending on the experiment, it typically ranged around 500. All images were resized to 640×640, and the batch size was set to 8 due to GPU memory constraints, although the paper suggests a batch size of 16. In the loss function, the balancing parameter (λ) was set to 0.7. The PSENet model utilized 7 kernels, with a minimal kernel scale of 0.7. To ensure a better consistency of the final output with the annotations in the Nomenclature dataset, oriented bounding boxes were chosen as the output format. [25]

The hyperparameters for TextPMs were also selected based on the official TextPMs paper. The training process consists of a pre-training phase of two epochs on the SynthText [31] dataset, where the Adam optimizer is used with a fixed learning rate of 0.001 and a batch size of 10. After the pre-training phase, the training process is divided into two stages: the first stage resizes images to 640×640 with a batch size of 6, while the second stage resizes images to 800×800 with a batch size of 4. Both stages adopt the SGD optimizer with an initial learning rate of

[3]`https://github.com/whai362/PSENet`
[4]`https://github.com/GXYM/TextPMs`

0.01, which is multiplied by 0.9 after every 100 epochs, with a momentum of 0.9. The number of epochs is set to 300 for each stage. [26]

It is advisable to consider dataset augmentation when dealing with a small amount of data like in this case. Dataset augmentation is a technique that involves creating additional data samples by applying various transformations or modifications to the existing training data. The goal is to increase the size of the training dataset and improve the model's ability to generalize. By introducing variations to the training data, the model becomes more robust and capable of handling different types of input data. In the thesis, the transformation includes random cropping, flipping, changes in brightness and contrast, and random rotation with an angle sampled from a Gaussian distribution between -60 and 60 degrees.

6.4 Evaluation

This section aims to introduce the standard approach for evaluating object detection tasks, followed by an explanation of the TotalText evaluation method utilized in this thesis.

6.4.1 Object Detection Evaluation

To evaluate the effectiveness of an object detection model, it is necessary to measure the degree of similarity between the predicted bounding box and the actual location of the object in the image. This is typically done by calculating the *Intersection over Union (IoU)*. [32]

The IoU is a metric used to measure how closely the predicted bounding box aligns with the ground truth bounding box. The value of IoU ranges between 0 and 1. When the two boxes overlap completely, the predicted bounding box is considered perfect, and the IoU is 1. Conversely, if the two boxes do not overlap at all, the IoU is 0. The IoU is calculated by dividing the area of intersection between the two bounding boxes by the area of their union:

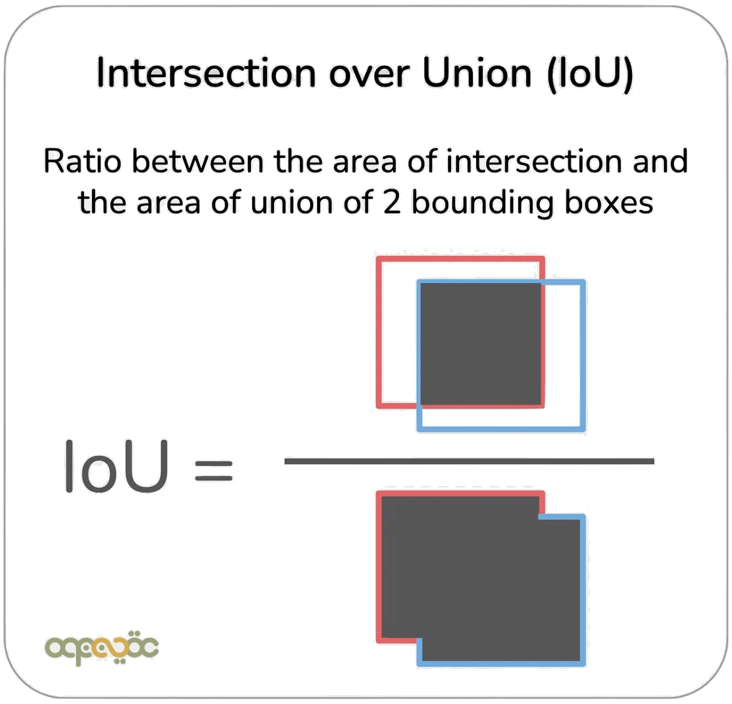

Intersection over Union (IoU)
Ratio between the area of intersection and the area of union of 2 bounding boxes
IoU =

Figure 6.8 This figure provides an illustration of the IoU calculation. [32]

After calculating the IoU between the predicted bounding box and the ground truth bounding box, the resulting value is compared to a specified threshold value. If the IoU value exceeds the threshold value, the detection is classified as *True Positive (TP)*, indicating that the predicted bounding box accurately identifies the object. On the other hand, if the IoU value is lower than the threshold, the detection is classified as *False Positive (FP)*, indicating that the predicted bounding box does not accurately identify the object. Additionally, if there is no TP detection for the corresponding ground-truth area, it is labeled as a *False Negative (FN)*.

Upon obtaining the TP, FP, and FN detections, the evaluation metrics such as *Precision*, *Recall*, and *F1-Score* can be computed using standard formulas:

$$Precision = \frac{Correct\ Predictions}{Total\ Predictions} = \frac{TP}{TP + FP} \tag{6.16}$$

$$Recall = \frac{Correct\ Predictions}{Total\ GroundTruth} = \frac{TP}{TP + FN} \tag{6.17}$$

$$F1\text{-}Score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{6.18}$$

6.4.2 TotalText Evaluation Protocol

This evaluation protocol differs from the approach presented in Section 6.4.1 by addressing a specific scenario. Consider two detections overlapping the same ground truth, where one detection covers the left half of the ground truth and the other covers the right half, both with an IoU of 0.5. Typically, when using a common IoU threshold of 0.7, this situation would result in two False Positives and one False Negative. However, in the context of the TotalText dataset evaluation protocol, this is considered as one True Positive. This scenario is common in Text Detection tasks, where the presence of large spaces between characters can cause the ground truth area to be split into multiple detections.

The primary idea is to identify all the overlapping detections for a particular ground truth and merge them to determine the IoU. This represents a one-to-many scenario. When combining these detections, the goal is to minimize the number of merged detections while gradually increasing them, if required. This strategy guarantees that the minimum number of detections are used to achieve the True Positive case and any leftover detections become False Positives.

The IoU calculation is divided into two cases, namely the Intersection over Detection and the Intersection over Ground Truth. The Intersection over Detection represents the intersection between the ground truth and the union of detections divided by the union of detections. On the other hand, the Intersection over Ground Truth calculation is the same intersection area but divided by the area of the ground truth. The threshold for the Intersection over Detection is set to 0.7, and for the Intersection over Ground Truth, it is set to 0.6, in order to provide some tolerance when detections cover additional area. To be considered a True Positive, both of these requirements must be met simultaneously.

The general workflow of this protocol involves initially attempting to match ground truths and detections using one-to-one scenarios. If any ground truths remain unmatched, then the one-to-many case is applied to satisfy the 0.7 and 0.6 thresholds. Finally, if any candidates remain, the many-to-many scenario is employed with the same principle.

Chapter 7

Experiments

This chapter provides a description and evaluation of the experiments carried out with both methods on the Nomenclature dataset. The loss functions are occasionally smoothed using *Exponential Moving Average (EMA)* for better visualization. The horizontal axis labeled as *step* in the graphs represents every tenth forward pass using the entire batch. Recording the results for every batch would be redundant as recording every tenth batch is sufficient. The Weights & Biases [33] tool was utilized for logging values during the training process.

7.1 Experiments with PSENet

This section presents the experiments performed using the Progressive Scale Expansion Network.

7.1.1 Initial Experiment

The Initial Experiment denotes that no augmentation or other modifications were made to the dataset. Only resizing the images to 640 × 640 was performed according to 6.3. Figure 7.1 displays the training loss during this experiment.

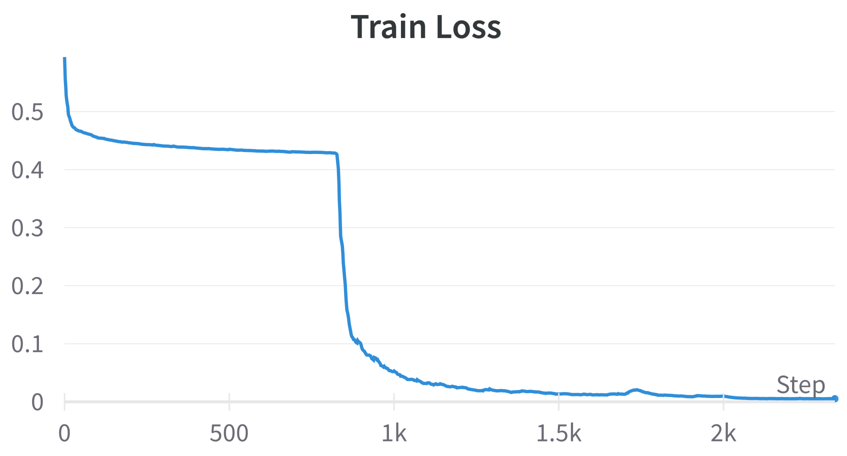


Figure 7.1 The training loss during initial experiment with EMA = 0.7 smoothing.

Due to the training loss reaching values near zero, the corresponding gradients became significantly small, and as a result, the model parameters stopped updating. Consequently, the run was terminated at the 233rd epoch out of a total of 500. The resulting validation loss is displayed in Figure 7.2.

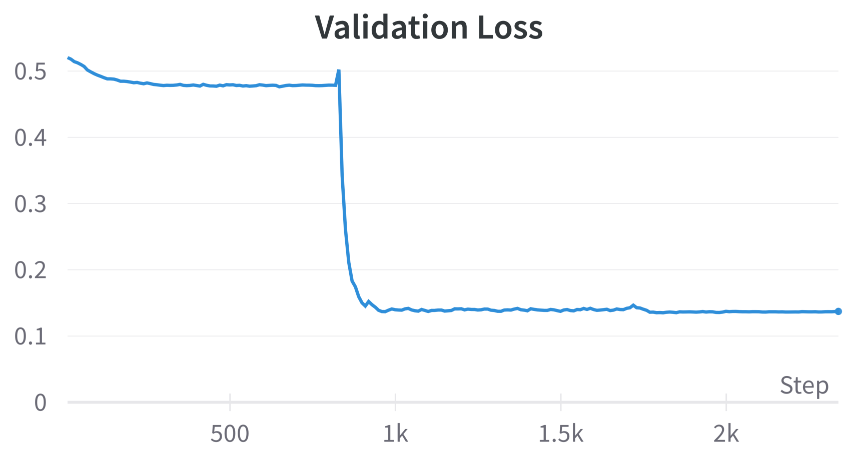


Figure 7.2 The validation loss during initial experiment with EMA = 0.2 smoothing.

At the 176th epoch, the validation loss reached its minimum, with a Precision of 70.0%, Recall of 83.3%, and F-Score of 76.1%. The lower Precision compared to Recall suggests a higher rate of false positives. The model struggled with recognizing toponyms on the maps and tended to detect some additional text, as depicted in Figure 7.3.

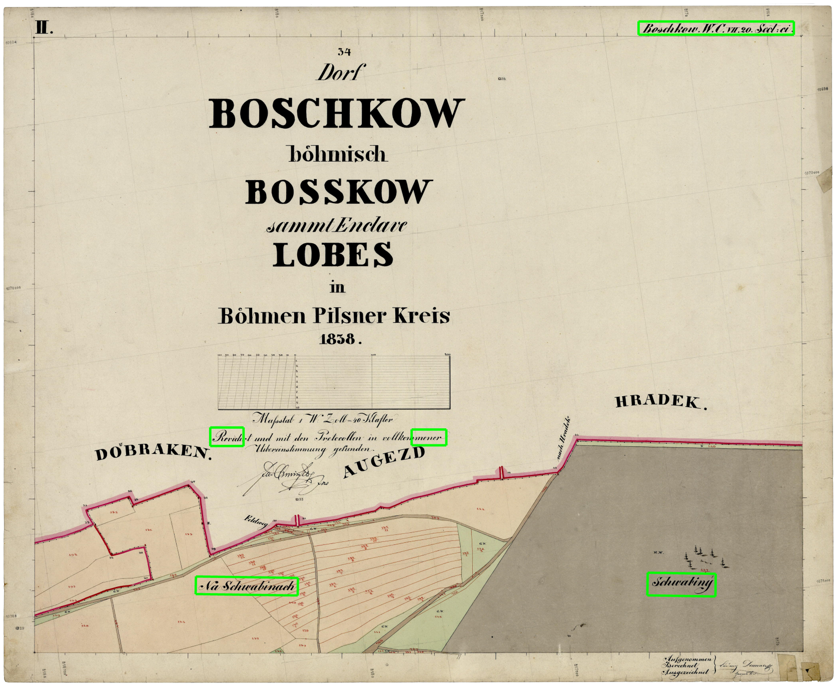


Figure 7.3 The output of the initial experiment is an illustrative example that showcases the typical false positives in the outputs generated by the model.

7.1.2 Augmentation

The experiment utilizes the augmentation strategy outlined in Section 6.3. This involves exposing the model to inputs during each epoch of training that include specific variations and randomness, which the model has not previously encountered. As a result, the model can encounter a subset of new inputs during each epoch, which contrasts with the Initial Experiment 7.1.1 that presented the model with the exact same data every epoch. Figure 7.4 displays the training loss.

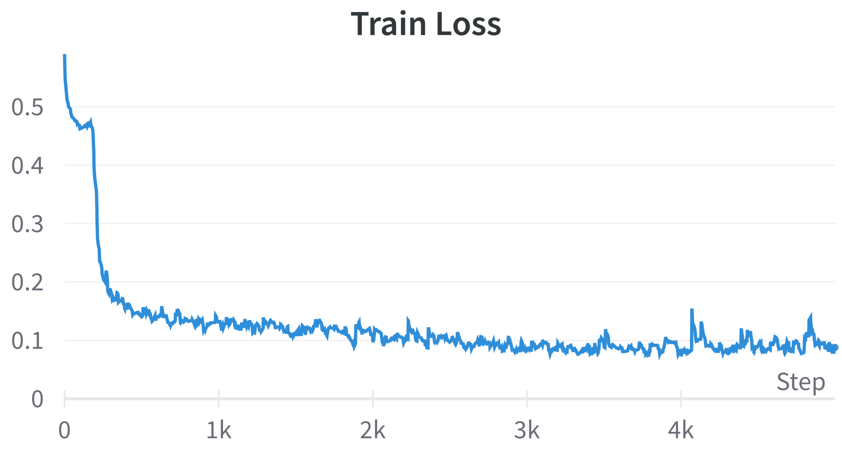


Figure 7.4 The training loss during augmentation experiment with EMA = 0.7 smoothing.

The training loss shows a pattern of fluctuation between high and low values. However, still presents a decreasing trend in the larger context. The run completed all 500 epochs, with Validation Loss shown in Figure 7.5.

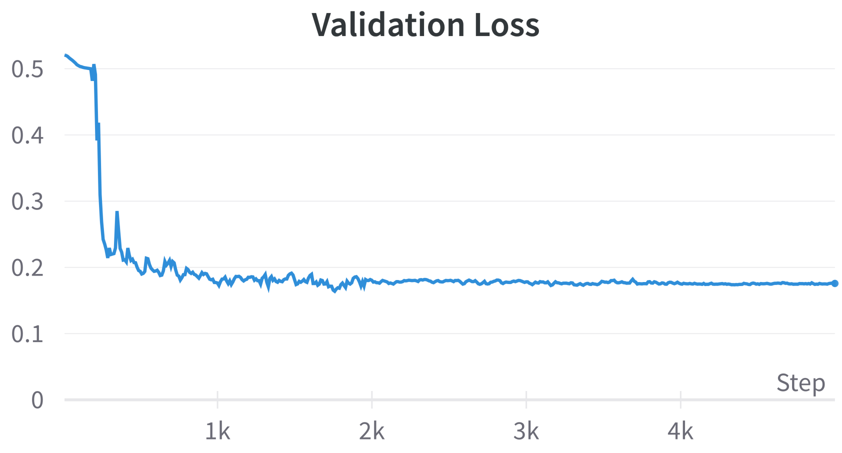


Figure 7.5 The validation loss during augmentation experiment with EMA = 0.2 smoothing.

At the 395th epoch, a small peak is observed in the Validation Loss, with the lowest value achieved at this point. This peak corresponds to a Precision of 91.6%, Recall of 86.2%, and F-Score of 88.8%, which are promising results. However, a significant portion of False Detections can be attributed to inaccurate ground truth annotations. The model performs well in differentiating between map toponyms and other instances of text. The main error lies in detecting the smallest instances in the maps, as illustrated in Figures 7.6 and 7.7.

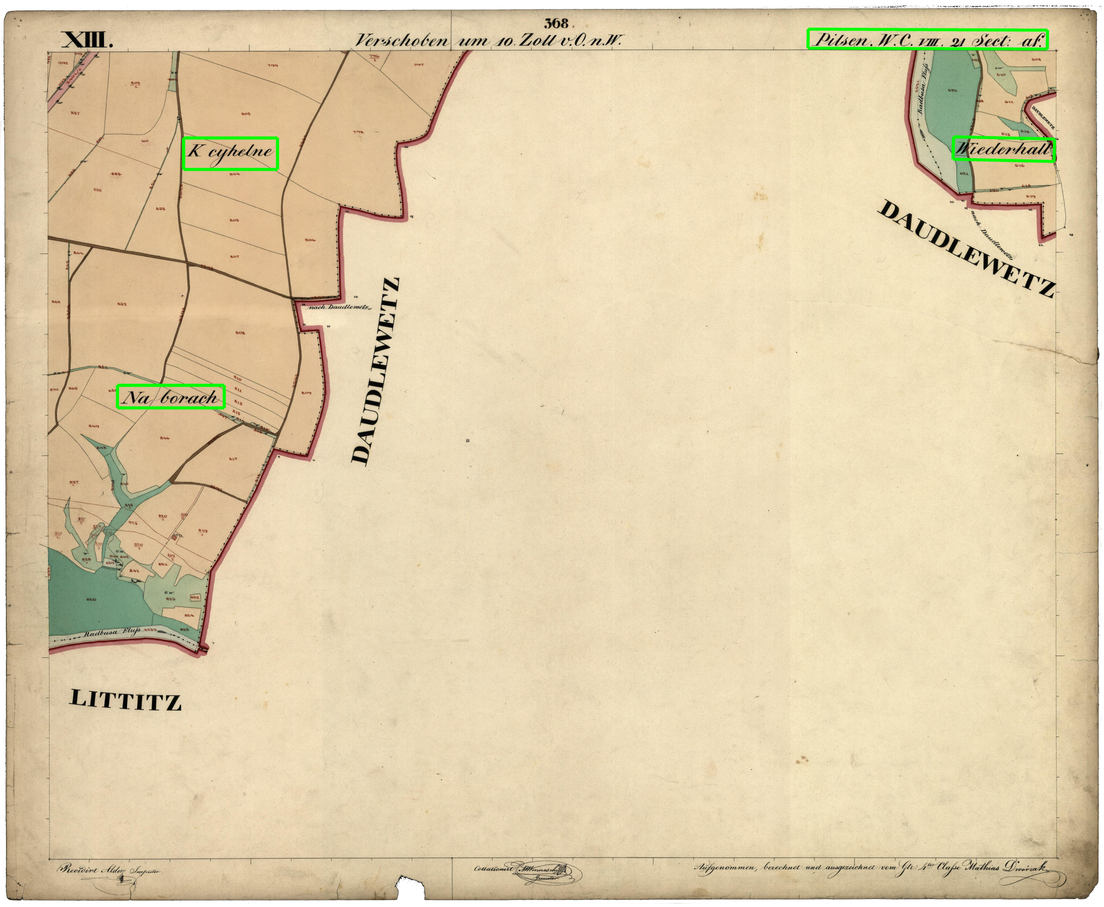


Figure 7.6 The outputs of the augmentation experiment indicate that the model is unable to detect the smallest maps toponyms. Figure 7.7 illustrates the corresponding ground truth area to this False Negative, located in the bottom left part of this Figure.

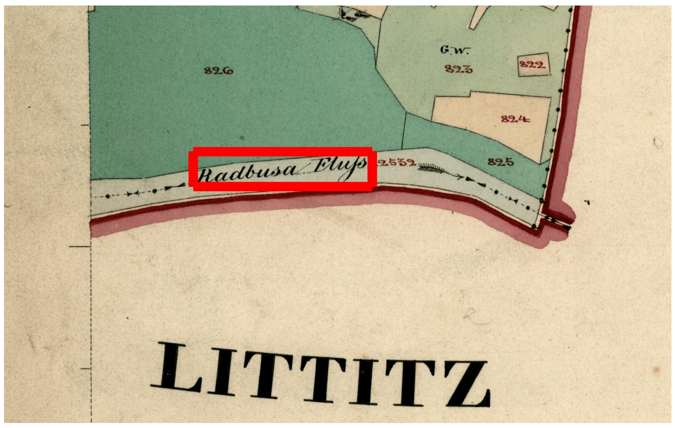


Figure 7.7 The ground truth area of tiny map toponym that was not detected in Figure 7.6.

7.1.3 Pretrained Model

The aim of this experiment is to evaluate the performance of a pre-trained model on the TotalText dataset, which is a more generalized text detection task. The experiment utilized the same weights[1] as described in the official PSENet [25] paper, and was trained using the process outlined in Section 6.3, except for the use of a batch size of 16. The resulting weights achieved a Precision of 84.0%, Recall of 78.0%, and F-Score of 80.9% on the TotalText dataset.

Applying the weights to the Nomenclature dataset yields a Precision of 21.9%, Recall of 32.4%, and F-Score of 26.1%. The model can detect printed instances, but it fails completely to detect handwritten toponyms, which are common in most scenarios. Instead of detecting map toponyms, the model typically detects other printed texts on the maps, as illustrated in Figure 7.8. Occasionally, the model detects text-like areas such as trees, paths, or numbers. The evaluation parameters and output visualizations demonstrate that the pre-trained model is not suitable for toponym detection in the Nomenclature dataset.

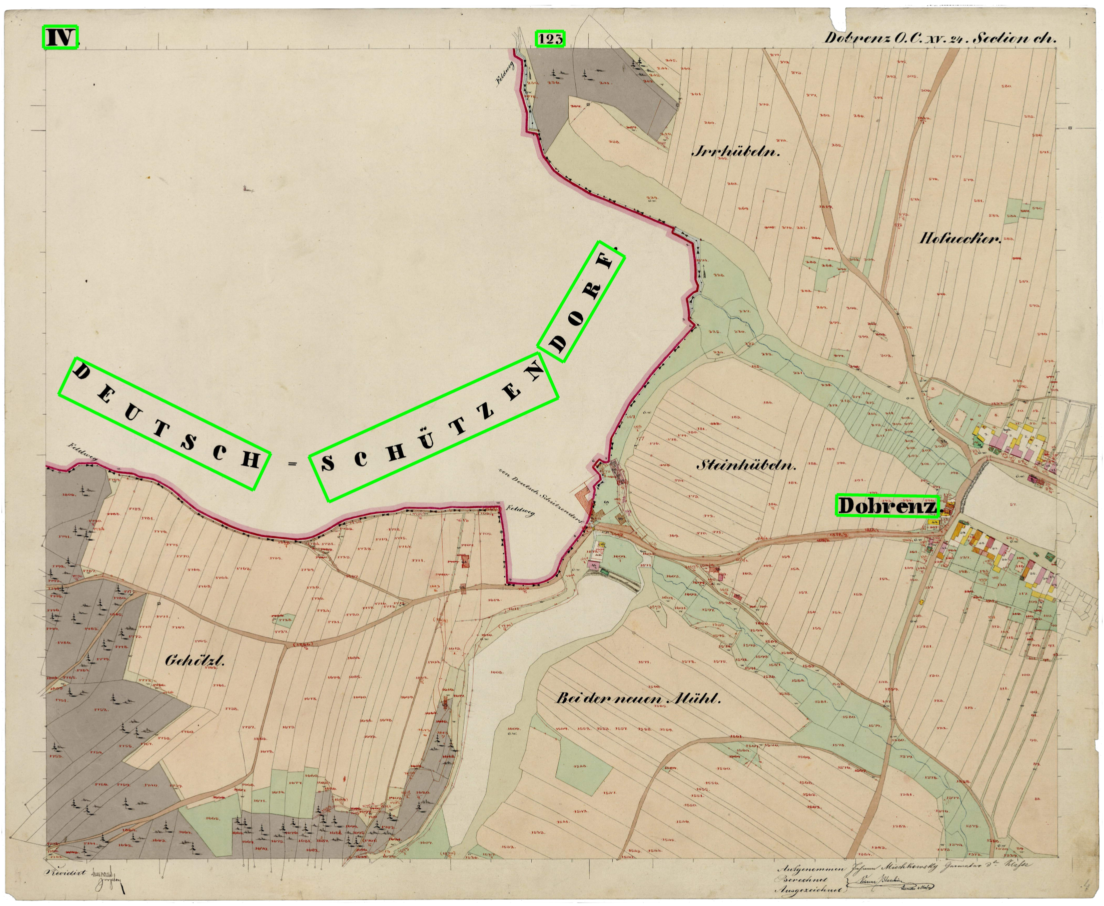


Figure 7.8 The output of the pretrained model is typically unable to detect handwritten toponyms, and only detects simple printed text instances, regardless of whether they are map toponyms or not.

[1] https://github.com/whai362/PSENet

7.1.4 Finetuning

This experiment aims to finetune the pretrained model used in the previous experiment 7.1.3 to enhance its performance. The experiment employs slightly different hyperparameters from the previous experiments, which are described in Section 6.3. The learning rate is lowered to 10^{-4}, and the number of epochs is reduced to 400. Otherwise, the process is the same as described in Section 6.3, with the utilization of an augmentation strategy. Figure 7.9 displays the training loss.

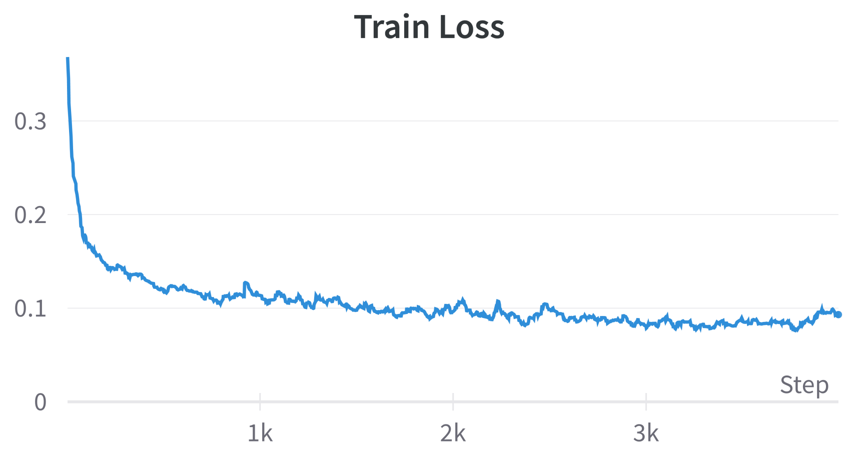


Figure 7.9 The training loss during finetuning experiment with EMA = 0.9 smoothing.

All 400 epochs were completed, and Figure 7.10 displays the Validation Loss.

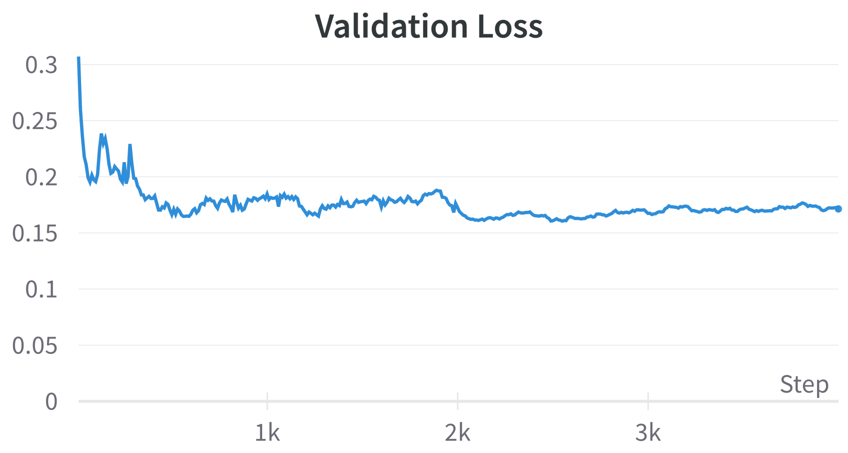
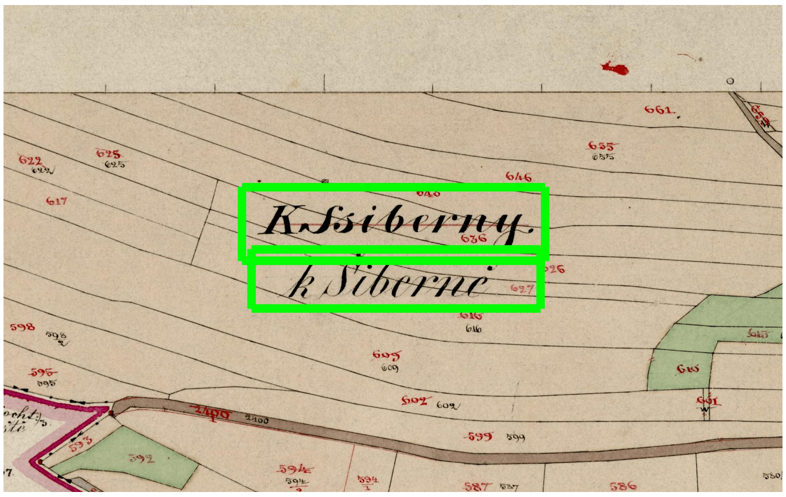


Figure 7.10 The validation loss during finetuning experiment with EMA = 0.4 smoothing.

At the 248th epoch, the validation loss achieved its lowest value, resulting in a Precision of 88.8%, Recall of 84.9%, and F-Score of 86.8%. The generated outputs were very similar to the ones produced by the augmentation experiment mentioned in section 7.1.2. The only difference was that this finetuned version attempted to produce smaller detections, which led to a lower F-Score due to imprecise ground truth annotations. Because the detections aim to be as precise as possible, they sometimes cause the area to be divided into several detections, as depicted in Figure 7.11 but are still classified as True Positive according to 6.4.2.



■ **Figure 7.11** The output of the finetuned model shows an example where a single ground truth area is detected with multiple smaller detections instead of a single large one.

7.1.5 Summary

Table 7.1 summarizes the evaluation results obtained from the experiments.

PSENet	Precision	Recall	F-Score
Initial	70.0	83.3	76.1
Augmentation	**91.6**	**86.2**	**88.8**
Pretrained	21.9	32.4	26.1
Finetuning	88.8	84.9	86.8

■ **Table 7.1** Overview of experiments results.

The Augmentation experiment achieved the best results with an F-Score of 88.8%. The most challenging aspect appears to be the detection of small text instances, as illustrated in Figure 7.7.

Although the Finetuning experiment did not perform better than the Augmentation experiment in terms of evaluation scores, primarily due to inaccurate ground truth annotations, its outputs cover a smaller area, which may be advantageous in certain applications.

The pretrained model struggles significantly with the complex background nature of cadastral maps and often fails to detect the handwritten toponyms, which comprise the majority of the dataset. As a result, the pretrained model is not appropriate for toponym detection in the Nomenclature dataset.

On average, the experiments took approximately 19 hours to complete the training, with the memory usage being displayed in Figure 7.12.

7.2 Experiments with TextPMs

This section presents the experiments performed using the Text Detection with Probability Maps approach.

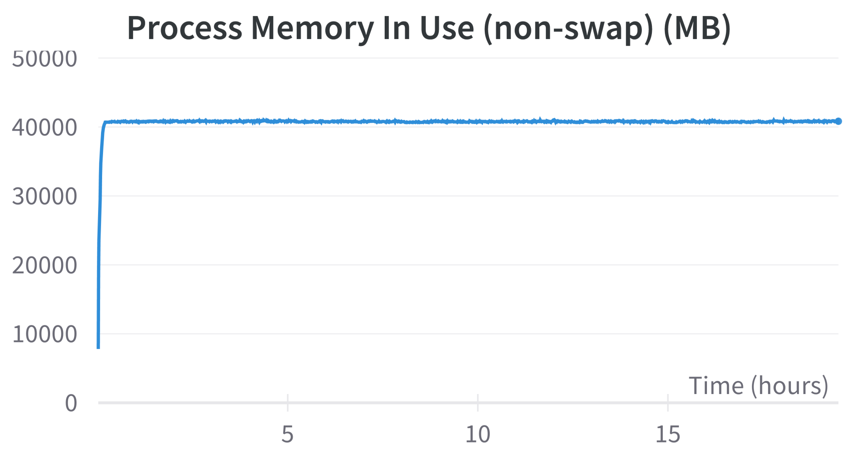


Figure 7.12 Memory usage during the training process of the experiments.

7.2.1 Initial Experiment

This experiment follows the same approach as presented in Section 7.1.1, where no augmentation was applied. The training process is described in section 6.3. The training and validation losses for the first stage are presented in Figures 7.13 and 7.14, respectively.

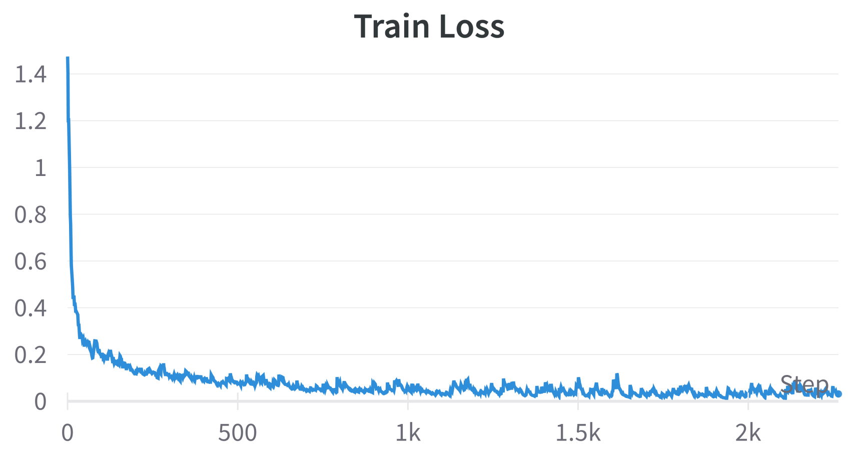
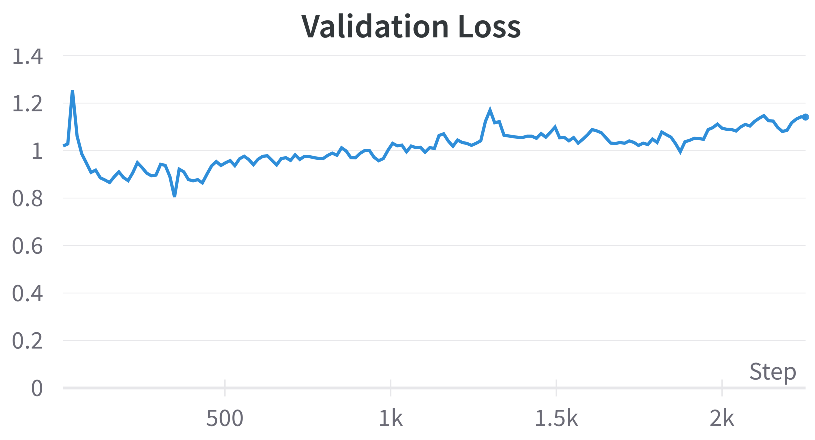


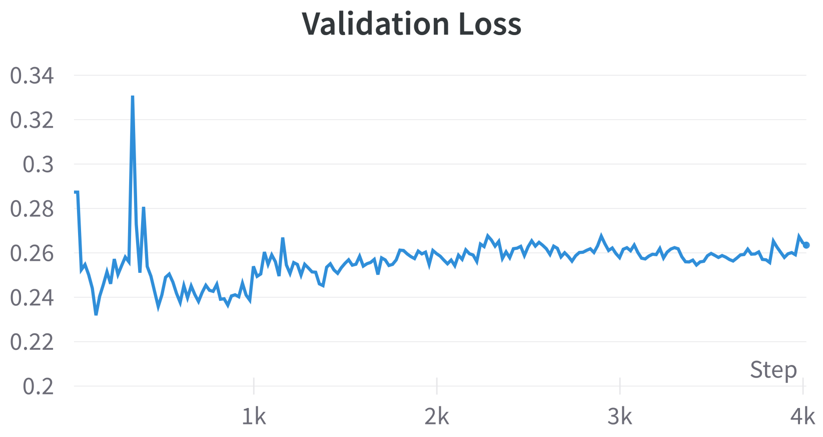
Figure 7.13 The training loss during the first stage with EMA = 0.7 smoothing.

Due to the increasing trend of the validation loss (Figure 7.14) function, it suggested that there was overfitting, leading to the termination of the run at the 161st epoch out of the 300. The validation loss reached its minimum at the 30th epoch, so the weights from this epoch were used for the second stage. Since the minimum was achieved at the 30th epoch out of 300, the number of epochs for the second stage was reduced to 200. The validation loss for the second stage is shown in Figure 7.15.

At the 9th epoch, the validation loss (Figure 7.15) reached its minimum, resulting in a Precision of 88.7%, Recall of 85.9%, and F-Score of 87.3%. This represents a significant improvement when compared to the F-Score of 76.1% obtained in PSENet experiment 7.1.1. While the model occasionally predicts additional text instances, as shown in Figure 7.16, it does so much less

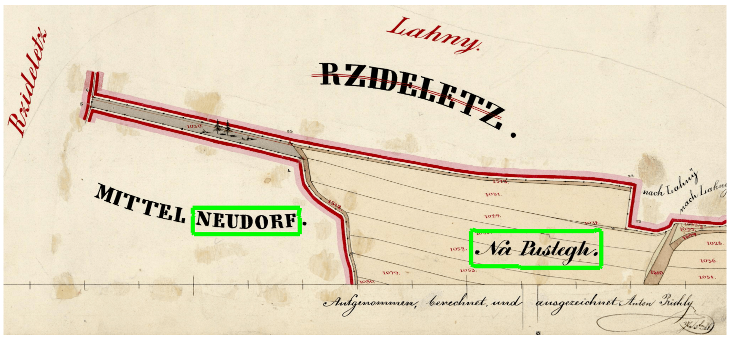


■ **Figure 7.14** The validation loss during the first stage with EMA = 0.2 smoothing.



■ **Figure 7.15** The validation loss during the second stage with EMA = 0.2 smoothing.

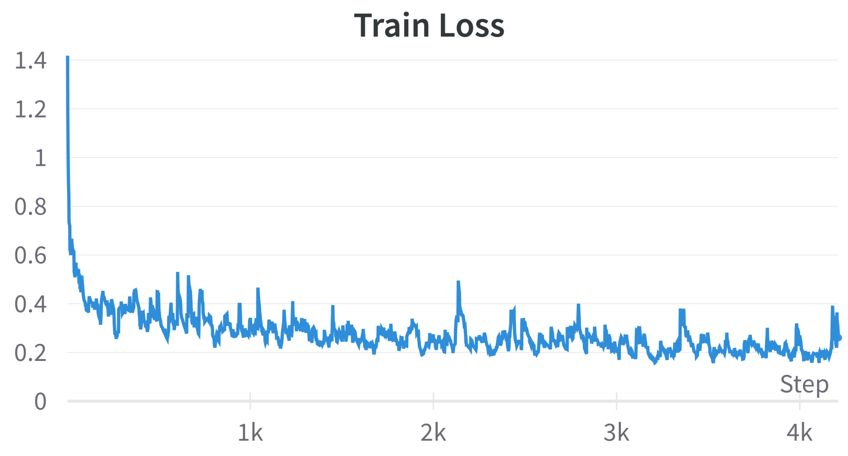
frequently than PSENet. Both models are still unable to detect the tiny text toponyms as shown in Figure 7.7. In very rare cases, the model may detect only part of a text instance or split it into multiple parts due to the larger space between characters.



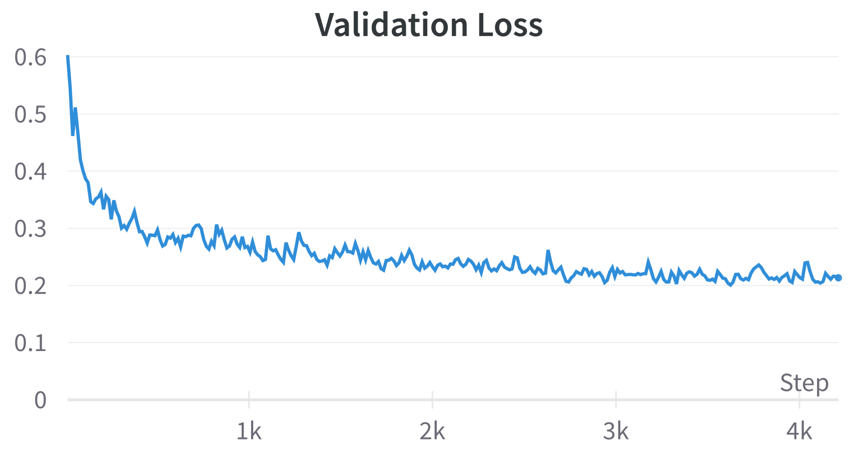
■ **Figure 7.16** The output of the initial experiment where a model detects non-toponym text instances (the left one).

7.2.2 Augmentation

This experiment involves the same augmentation concept as discussed in Section 7.1.2. The training process is detailed in 6.3. Figures 7.17 and 7.18 display the training and validation losses, respectively.



■ **Figure 7.17** The training loss during the first stage with EMA = 0.7 smoothing.

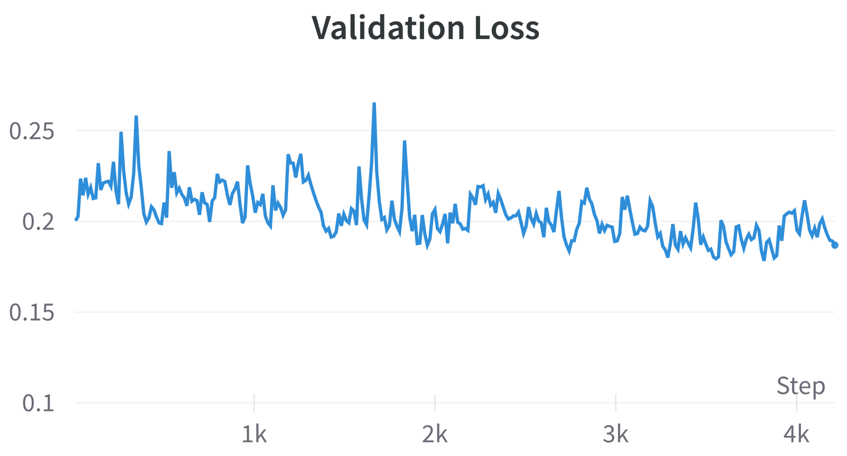


■ **Figure 7.18** The validation loss during the first stage with EMA = 0.2 smoothing.

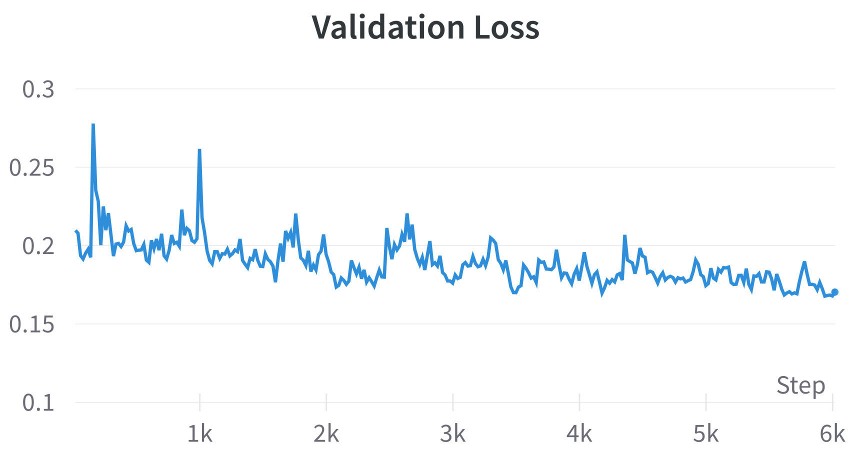
As the validation loss 7.18 was still decreasing, it was deemed appropriate to continue the training process for another 300 epochs with the same configuration. This resulted in a validation loss shown in Figure 7.19.

The validation loss 7.19 continued to decrease, but at such a slow rate that continuing for additional epochs would likely not make a significant difference. The minimum was reached at the 273rd epoch, which is the 573rd epoch in total with the first part. Weights from this epoch were selected for the second stage.

The validation loss 7.20 reached its minimum at the 296th epoch, which is the 869th epoch in total including the first stage. This resulted in a Precision of 93.5%, Recall of 88.8%, and F-Score of 91.1%, improving upon the best F-Score of 88.8% achieved by PSENet. Most errors still stem



■ **Figure 7.19** The validation loss during the first stage continuation with EMA = 0.2 smoothing.



■ **Figure 7.20** The validation loss during the second stage with EMA = 0.2 smoothing.

from inaccurate ground truth annotations. While the model is able to detect some tiny text toponyms, as illustrated in Figure 7.21, it is not capable of detecting all of them. Additionally, as in the initial experiment 7.2.1, there are still rare cases where the issue with bigger spaces between characters occurs.

7.2.3 Pretrained Model

The model used in this experiment is similar to the one used in the PSENet experiment 7.1.3, as it is trained on the TotalText dataset. The weights[2] for this experiment are described in the official TextPMs [26] paper, and the training process is explained in Section 6.3. These weights achieved a Precision of 89.9%, Recall of 87.6%, and F-Score of 88.7% on the TotalText dataset.

Applying these weights to the Nomenclature dataset resulted in a Precision of 27.0%, Recall of 43.8%, and F-Score of 33.4%. As in the PSENet experiment 7.1.3, the precision is low because the models detect non-toponym text instances frequently. However, the recall is slightly better than

[2]`https://github.com/GXYM/TextPMs`

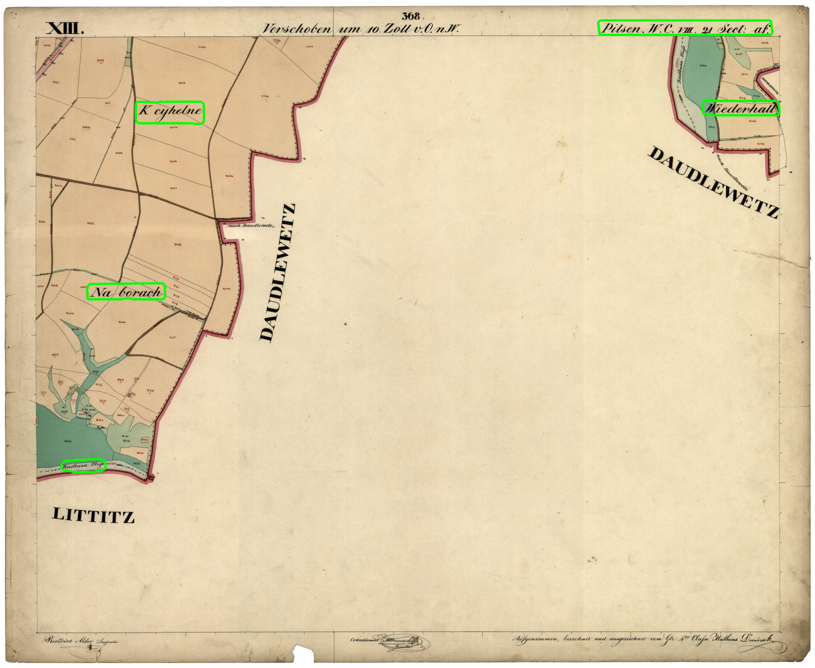

XIII.
368
Verschoben um 10 Zoll v.O. n.W.
Pilsen. W.C. VIII. 21 Sect. af.
Kopelne
Wiederhall
DAUDLEWETZ
DAUDLEWETZ
Na borach
LITTITZ

Figure 7.21 This output demonstrates that the model can detect some of the tiny map toponyms, as compared to Figure 7.6.

in PSENet 7.1.3. The model can sometimes detect map toponyms, as illustrated in Figure 7.22, but not all of them, especially those "hidden" in trees, paths, buildings, or other map elements.

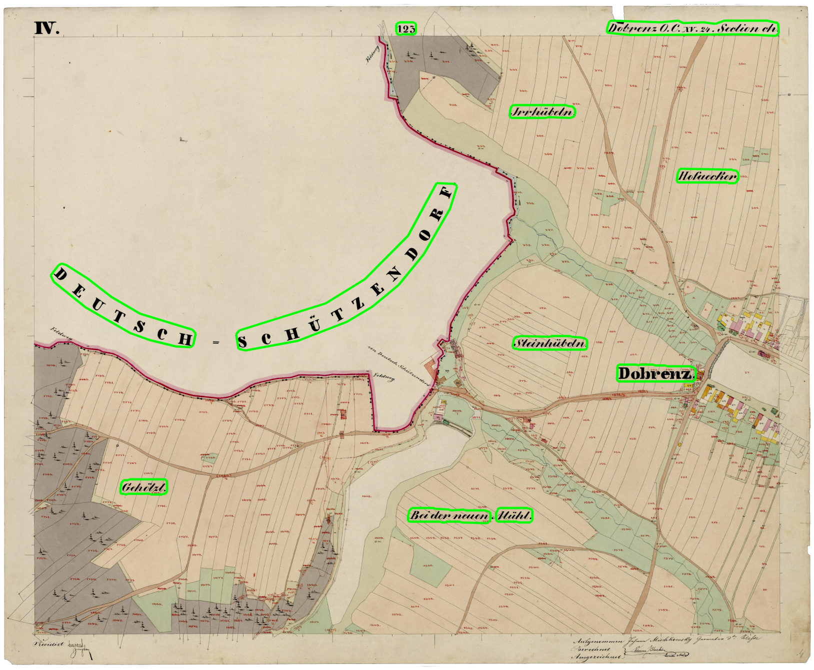


Figure 7.22 The example output of the TextPMs pretrained model demonstrates its improved ability to detect handwritten toponyms when compared to the output of the PSENet pretrained model, as illustrated in Figure 7.8.

7.2.4 Finetuning

This experiment aims to fine-tune the pretrained model used in the previous experiment 7.2.3 to improve its performance. The training process differs from the original one described in Section 6.3. Only one stage with 600 epochs is used, with images resized to 800 × 800 and a batch size of 4. The augmentation strategy is also employed, and the learning rate is lowered to 0.001. The training and validation losses are displayed in Figures 7.23 and 7.24, respectively.

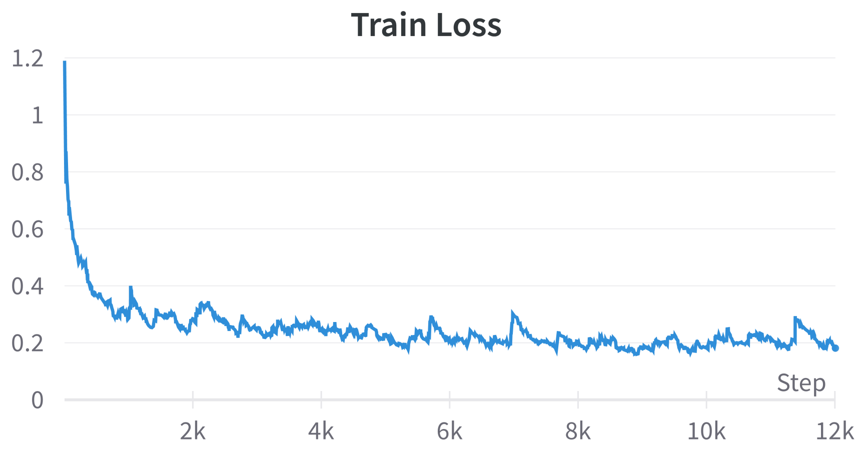


Figure 7.23 The training loss during finetuning experiment with EMA = 0.9 smoothing.

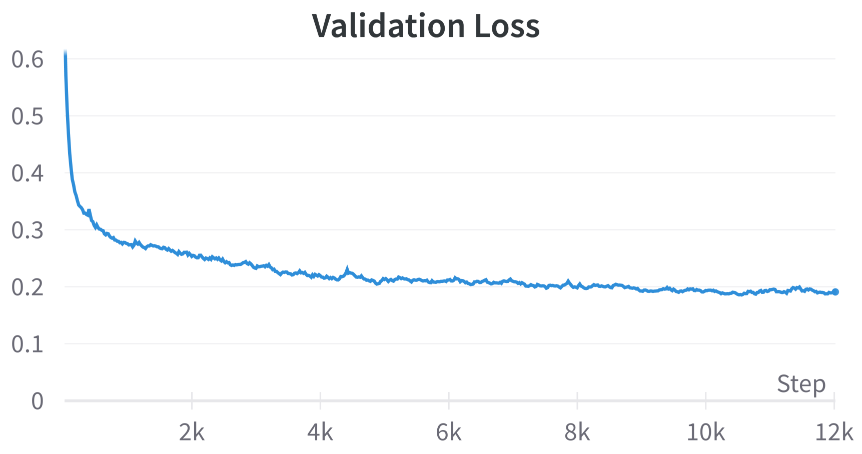


Figure 7.24 The validation loss during finetuning experiment with EMA = 0.4 smoothing.

The validation loss 7.24 reached its minimum at the 592nd epoch, achieving a Precision of 91.5%, Recall of 87.4%, and F-Score of 89.4%. The outputs are similar to the augmentation experiment in Section 7.2.2. In general, the detected areas are slightly smaller, but this means that the model is more sensitive to larger character spacing, as shown in Figure 7.25.

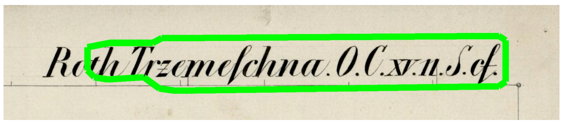


Figure 7.25 The output of the finetuning experiment demonstrates that the model is not always capable of detecting the whole nomenclature area due to larger space between characters.

7.2.5 Summary

The evaluation results obtained from the experiments are summarized in Table 7.2.

TextPMs	Precision	Recall	F-Score
Initial	88.7	85.9	87.3
Augmentation	**93.5**	**88.8**	**91.1**
Pretrained	27.0	43.8	33.4
Finetuning	91.5	87.4	89.4

Table 7.2 Overview of experiments results.

The augmentation experiment produced the best results, achieving an F-Score of 91.1%. However, the model still struggles with detecting all of the smallest map toponyms, as shown in Figure 7.7. In rare instances, the model also experiences difficulty with larger character spacing, but this is a necessary trade-off for generating the smallest detections possible. The authors of the Nomenclature dataset [22] obtained a best F-Score of 87.1% using the EAST [17] detector, which demonstrates the potential of the TextPMs method.

The finetuning experiment generally produces smaller detections, but it is more susceptible to larger character spacing than the augmentation experiment, resulting in slightly lower evaluation scores.

The performance of the pretrained model in detecting map toponyms is unsatisfactory, as it often detects non-toponymic text instances and struggles to identify most of the handwritten toponyms.

The training process took about 10 hours on average with the memory usage depicted in Figure 7.26, resulting in roughly 5 times better memory usage than the PSENet experiments shown in Figure 7.12.

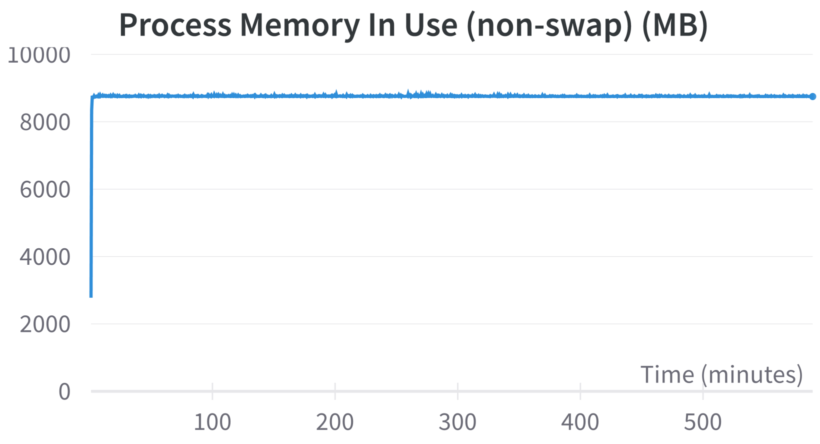


Figure 7.26 Memory usage during the training process of the experiments.

The evaluation results obtained from the experiments are not particularly optimistic, mainly because of inaccurate ground truth annotations. An example output presented in Figure 7.27 shows much more precise detections compared to the ground truth in Figure 5.4, which leads to the two middle detections not being classified as True Positives. Consequently, the F-Score for this output is only 50%, even though the detections are correct.

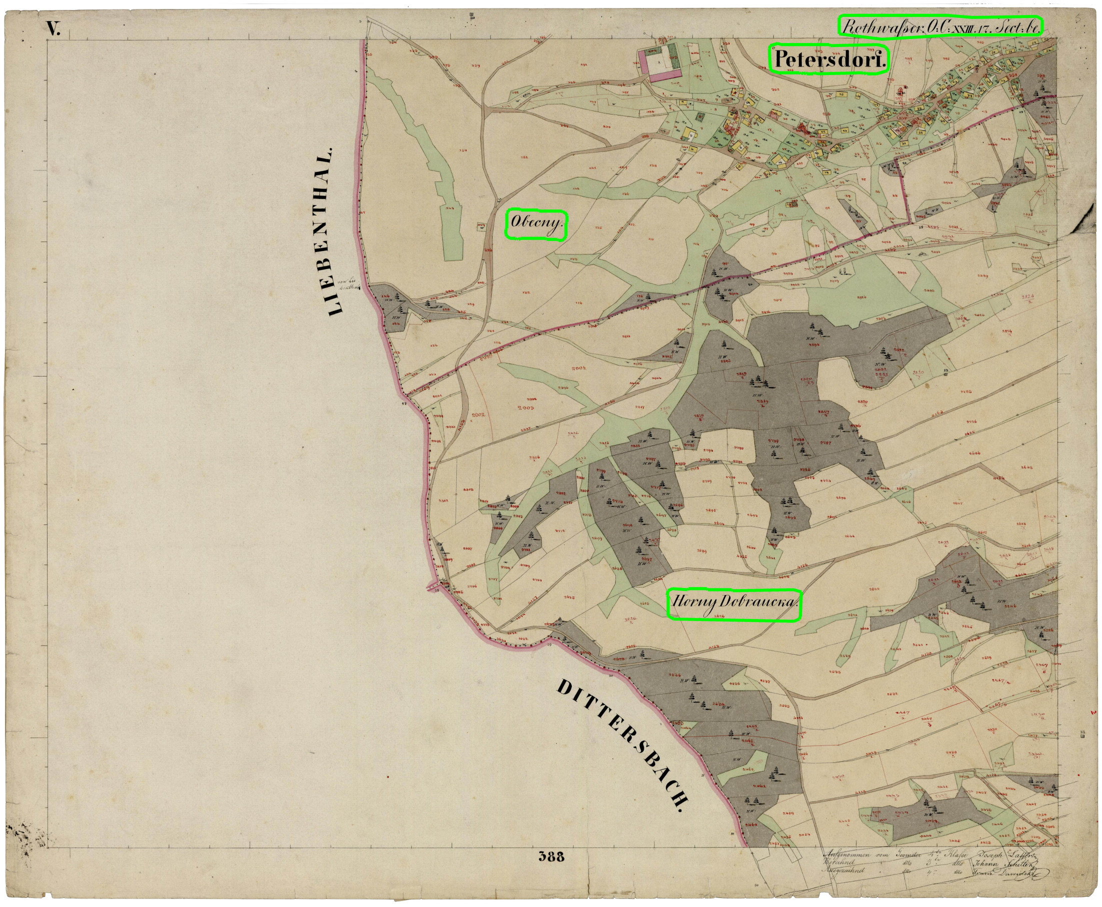


Figure 7.27 This output contains more precise detections in comparison to the ground truth annotations in Figure 5.4, which resulted in lower evaluation scores.

Chapter 8
Conclusion

The main objective was to implement and apply two text detection methods on a dataset of historical maps. Both methods were successfully implemented using PyTorch and achieved F-scores of 88.8% and 91.1%, demonstrating their effectiveness for text detection on historical maps. However, the methods' performance was negatively affected by the inaccurate annotations they were trained on. This resulted in the addition of extraneous margins around text instances.

The unresolved weakness lies in the large spaces between letters, which may result in partial text detection or the division of the text into multiple instances.

The capability of the pretrained models to detect map toponyms is unsatisfactory as they frequently detect non-toponymic text instances and struggle with the complex background nature of cadastral maps, leading to F-scores of 26.1% and 33.4%.

For future work, an OCR module could be added to extract and compare the resulting text. One possible approach could be to use the Editing Distance metric to compare the detected text with the ground truth annotations.

Overall, the implemented methods demonstrate their suitability for text detection on historical maps and have the potential for various applications.

Bibliography

1. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. 2015. Available from arXiv: `1512.03385 [cs.CV]`.
2. LIN, Tsung-Yi; DOLLÁR, Piotr; GIRSHICK, Ross; HE, Kaiming; HARIHARAN, Bharath; BELONGIE, Serge. Feature Pyramid Networks for Object Detection. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944. Available from DOI: `10.1109/CVPR.2017.106`.
3. KARPATHY, Andrej. *CS231n: Convolutional Neural Networks for Visual Recognition* [online]. 2020. [visited on 2023-01-12]. Available from: `https://cs231n.github.io/`.
4. MISHRA, Mayank. *Convolutional Neural Networks, Explained* [online]. Towards Data Science, 2020-09 [visited on 2023-01-14]. Available from: `https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939`.
5. ALBELWI, Saleh; MAHMOOD, Ausif. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*. 2017, vol. 19, no. 6. ISSN 1099-4300. Available from DOI: `10.3390/e19060242`.
6. REYNOLDS, Anh. *Convolutional Neural Networks (CNNs)* [online]. 2017. [visited on 2023-01-20]. Available from: `https://anhreynolds.com/blogs/cnn.html`.
7. NANDEPU, Raghuram. Understanding and implementation of Residual networks(ResNets). *Medium* [online]. 2019 [visited on 2023-02-01]. Available from: `https://medium.com/analytics-vidhya/understanding-and-implementation-of-residual-networks-resnets-b80f9a507b9c`.
8. SHINDE, Yashowardhan. How to code your ResNet from scratch in Tensorflow?. *Analytics Vidhya* [online]. 2021 [visited on 2023-02-01]. Available from: `https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/`.
9. HUI, Jonathan. *Understanding Feature Pyramid Networks for Object Detection (FPN)* [online]. 2018-03. [visited on 2023-02-02]. Available from: `https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c`.
10. SHIN, Sangho; HAN, Hyoju; LEE, Sung Ho. Improved YOLOv3 with duplex FPN for object detection based on deep learning. *International Journal of Electrical Engineering & Education*. 2021. Available from DOI: `10.1177/0020720920983524`.
11. DENG, Dan; LIU, Haifeng; LI, Xuelong; CAI, Deng. PixelLink: Detecting Scene Text via Instance Segmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018, vol. 32, no. 1. Available from DOI: `10.1609/aaai.v32i1.12269`.

12. LONG, Shangbang; RUAN, Jiaqiang; ZHANG, Wenjie; HE, Xin; WU, Wenhao; YAO, Cong. TextSnake: A Flexible Representation for Detecting Text of Arbitrary Shapes. In: *Computer Vision – ECCV 2018*. Springer International Publishing, 2018, pp. 19–35. Available from DOI: `10.1007/978-3-030-01216-8_2`.

13. YE, Jian; CHEN, Zhe; LIU, Juhua; DU, Bo. TextFuseNet: Scene Text Detection with Richer Fused Features. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2020. Available from DOI: `10.24963/ijcai.2020/72`.

14. XU, Yongchao; WANG, Yukang; ZHOU, Wei; WANG, Yongpan; YANG, Zhibo; BAI, Xiang. TextField: Learning a Deep Direction Field for Irregular Scene Text Detection. *IEEE Transactions on Image Processing*. 2019, vol. 28, no. 11, pp. 5566–5579. Available from DOI: `10.1109/tip.2019.2900589`.

15. LIAO, Minghui; ZOU, Zhisheng; WAN, Zhaoyi; YAO, Cong; BAI, Xiang. Real-Time Scene Text Detection With Differentiable Binarization and Adaptive Scale Fusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022, vol. 45, no. 1, pp. 919–931. Available from DOI: `10.1109/tpami.2022.3155612`.

16. LIAO, Minghui; SHI, Baoguang; BAI, Xiang. TextBoxes++: A Single-Shot Oriented Scene Text Detector. *IEEE Transactions on Image Processing*. 2018, vol. 27, no. 8, pp. 3676–3690. Available from DOI: `10.1109/tip.2018.2825107`.

17. ZHOU, Xinyu; YAO, Cong; WEN, He; WANG, Yuzhi; ZHOU, Shuchang; HE, Weiran; LIANG, Jiajun. EAST: An Efficient and Accurate Scene Text Detector. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. Available from DOI: `10.1109/cvpr.2017.283`.

18. ZHANG, Chengquan; LIANG, Borong; HUANG, Zuming; EN, Mengyi; HAN, Junyu; DING, Errui; DING, Xinghao. Look More Than Once: An Accurate Detector for Text of Arbitrary Shapes. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019. Available from DOI: `10.1109/cvpr.2019.01080`.

19. SHI, Baoguang; BAI, Xiang; BELONGIE, Serge. Detecting Oriented Text in Natural Images by Linking Segments. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. Available from DOI: `10.1109/cvpr.2017.371`.

20. TANG, Jun; YANG, Zhibo; WANG, Yongpan; ZHENG, Qi; XU, Yongchao; BAI, Xiang. SegLink++: Detecting Dense and Arbitrary-shaped Scene Text by Instance-aware Component Grouping. *Pattern Recognition*. 2019, vol. 96, p. 106954. Available from DOI: `10.1016/j.patcog.2019.06.020`.

21. BAEK, Youngmin; LEE, Bado; HAN, Dongyoon; YUN, Sangdoo; LEE, Hwalsuk. Character Region Awareness for Text Detection. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019. Available from DOI: `10.1109/cvpr.2019.00959`.

22. LENC, Ladislav; MARTÍNEK, Jiří; BALOUN, Josef; PRANTL, Martin; KRÁL, Pavel. Historical Map Toponym Extraction for Efficient Information Retrieval. In: *Document Analysis Systems*. Springer International Publishing, 2022, pp. 171–183. Available from DOI: `10.1007/978-3-031-06555-2_12`.

23. CH'NG, Chee Kheng; CHAN, Chee Seng; LIU, Chenglin. Total-Text: Towards Orientation Robustness in Scene Text Detection. *International Journal on Document Analysis and Recognition (IJDAR)*. 2020, vol. 23, pp. 31–52. Available from DOI: `10.1007/s10032-019-00334-z`.

24. LENC, Ladislav; KRÁL, Pavel. *Nomenclature Dataset v 1.0: Dataset for Detection and Recognition of Handwritten Nomenclatures and toponyms from Historical Cadastral Maps* [online]. [visited on 2023-02-15]. Available from: `https://corpora.kiv.zcu.cz/nomenclature/`.

25. WANG, Wenhai; XIE, Enze; LI, Xiang; HOU, Wenbo; LU, Tong; YU, Gang; SHAO, Shuai. Shape Robust Text Detection With Progressive Scale Expansion Network. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019. Available from DOI: `10.1109/cvpr.2019.00956`.

26. ZHANG, Shi-Xue; ZHU, Xiaobin; CHEN, Lei; HOU, Jie-Bo; YIN, Xu-Cheng. Arbitrary Shape Text Detection via Segmentation with Probability Maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022, pp. 1–1. Available from DOI: `10.1109/tpami.2022.3176122`.

27. VATTI, Bala R. A generic solution to polygon clipping. *Commun. ACM*. 1992, vol. 35, pp. 56–63.

28. SHRIVASTAVA, Abhinav; GUPTA, Abhinav; GIRSHICK, Ross. Training Region-Based Object Detectors with Online Hard Example Mining. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. Available from DOI: `10.1109/cvpr.2016.89`.

29. HAN, Jun; MORAGA, Claudio. The influence of the sigmoid function parameters on the speed of backpropagation learning. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1995, pp. 195–201. Available from DOI: `10.1007/3-540-59497-3_175`.

30. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

31. GUPTA, Ankush; VEDALDI, Andrea; ZISSERMAN, Andrew. Synthetic Data for Text Localisation in Natural Images. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. Available from DOI: `10.1109/cvpr.2016.254`.

32. ANWAR, Aqeel. *What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?* [online]. Towards Data Science, 2022-05 [visited on 2023-03-12]. Available from: `https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b`.

33. BIEWALD, Lukas. *Experiment Tracking with Weights and Biases*. 2020. Available also from: `https://www.wandb.com/`. Software available from wandb.com.

Contents of the attached media

```
 LICENCE.txt ............ licence for the attachment
 README.md .............. description of the media content
 src
   PSENet ............... the source code of the PSENet implementation in PyTorch
   TextPMs .............. the source code of the TextPMs implementation in PyTorch
   thesis ............... the source format of the thesis in LaTeX
 thesis.pdf ............. the thesis text in PDF format
```