INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Processing of Historical Phonetic Maps using Computer Vision and Deep Learning

Manh Duy Nguyen

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Processing of Historical Phonetic Maps using Computer Vision and Deep Learning

Manh Duy Nguyen

Supervision:	Prof. Dr. Dieter Kranzlmüller
Advisors:	Sophia Grundner-Culemann
	Peter Zinterhof
Date:	11. Oktober 2019

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 11. Oktober 2019

. . . .

(Unterschrift des Kandidaten)

Abstract

In the course of digitization, automated systems are increasingly being developed that simplify and accelerate processes in industry. The area of humanities would also benefit from automated data processing as it works with a large number of data collections. This work, which is developed in collaboration with the Institute of Romance Languages and Literatures and the IT Group Humanities of the Ludwig Maximilian University Munich (LMU), examines methods to automatically process historical phonetic maps from the VerbaAplina project. Using Computer Vision and Deep Learning methods, algorithms are developed to recognize the correct mapping of phonetic transcriptions to corresponding position numbers on a map. The quality of the mappings is evaluated by statistical measurements. The algorithms are then compared and evaluated.

Abstrakt

Im Zuge der Digitalisierung werden vermehrt automatisierte Systeme entwickelt, die Prozesse in der Industrie vereinfachen und beschleunigen. Auch der Bereich der Geisteswissenschaften würde von einer automatisierten Datenverarbeitung profitieren, da er mit einer hohen Anzahl von Datensammlungen arbeitet. Die vorliegende Arbeit, welche in Zusammenarbeit mit dem Institut für Romanistik und der IT-Gruppe Geisteswissenschaften der Ludwig-Maximilians-Universität München (LMU) entstand, untersucht Methoden zur automatischen Verarbeitung von historischen phonetischen Karten aus dem VerbaAplina-Projekt. Mittels Computer Vision- und Deep Learning-Methoden werden Algorithmen entwickelt, die korrekte Zuordnungen von Lautschriften zu korrespondierenden Positionsnummern auf einer Karte erkennen. Die Qualität der Zuordnungen wird mittels statistischer Messwerte evaluiert. Anschließend werden die Algorithmen miteinander verglichen und bewertet.

Contents

List of Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
FCN	Fully Convolutional Network
ML	Machine Learning
NN	Neural Network
ReLU	Rectified Linear Unit
ROI	Region Of Interest
RPN	Region Proposal Network
SVM	Support Vector Machine

1. Introduction

1.1. Motivation

In recent years, deep learning has developed enormously and achieved many successes [MN18]. Multiple deep learning methods have proven to be a state-of-the-art approach to many complex tasks [KSH12, ODZ+16, MKS+13], such as image classification, natural language processing or object detection. Most of them replaced the classical symbolic Artificial Intelligence (AI) approaches, where problems are solved by providing a large set of specific rules to cover the whole problem space. Especially for well-defined and logical problems, this is an appropriate application. Older chess programs, for example, are based on predefined rules created by humans. With increasing complexity, it is not possible to find enough rules to meet the complexity, so deep learning is becoming more popular [Cho18]. With Neural Networks (NNs) complex games such as Go [SHM+16] can be mastered to even beat humans.
The development of Convolutional Neural Networks (CNNs) led to a breakthrough in image processing and thus to new possibilities to process problems in Computer Vision (CV). Traditional CV techniques have been surpassed in terms of accuracy and occasionally efficiency [GGOEO+17]. With the advancement of CNNs, not only objects in an image can be classified, but also localized [GDDM14, Gir15, RHGS15, RDGF16]. Furthermore, specific CNNs can even segment an image into entities [ZMCL16], where for each pixel a label is identified [GGOEO+17]. This allows for example to recognize every pixel of a specific animal in an image (Figure 1.1).

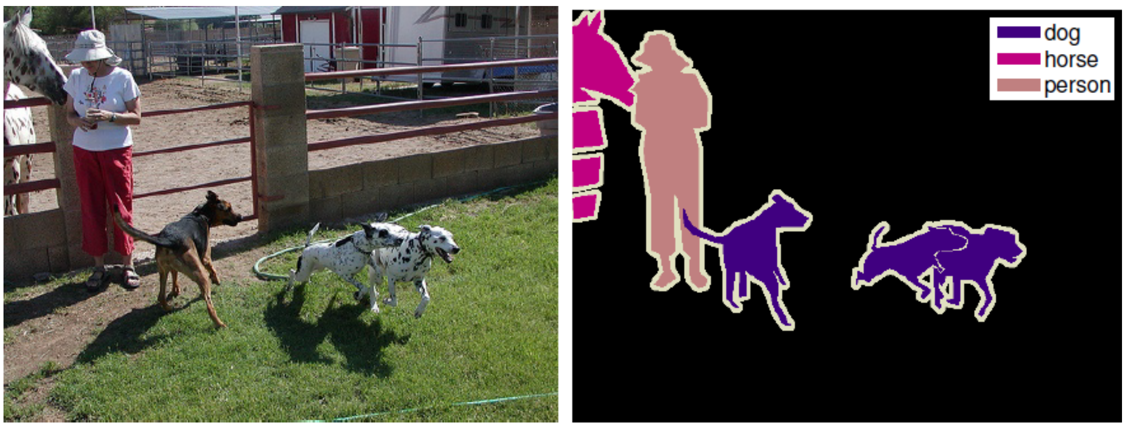


Figure 1.1.: Example of Image Segmentation [BKC17]

In the course of digitization, these developments form the basis for the automation of many data-based systems, such as in the area of humanities. This work is originally developed in collaboration with the Institute of Romance Languages and Literatures and the IT Group Humanities of the Ludwig Maximilian University (LMU) in Munich.

1.2. VerbaAlpina

The current research project of the Romance Institute and the IT Group Humanities: VerbaAlpina researches the "documentation and analysis of languages and dialects spoken in the Alpine region with a focus on lexis and with interdisciplinary and diachronic scope" [KL18].
VerbaAlpina is a research project of the LMU that explores the cultural- and linguistic area of the alpine region. The leading researchers Krefeld and Lücke [KL14] describe the project as the opening-up of the Alpine region, which is strongly fragmented in terms of individual language and dialect, selectively and analytically in its cultural and linguistic-historical togetherness. This overcomes the traditional limitation to essentially current political units.
Language atlases and dictionaries are used as the basis for the project database. To build this database, they are digitized in several steps. The starting point is the paper format, which is scanned to display the content as a text file. From this text file, the data is structured and stored in a database. One of the main atlas sources is the *AIS*, Language and Subject Atlas of Italy and Southern Switzerland [JJS28]. On these maps, atlas informants are depicted as position numbers and expressions are depicted as phonetic transcriptions. The goal is to automatically acquire structured data [Lü19].

1.3. Objective

NavigAIS, a web page provided by Tisato et al. [Tis17], represents the current state of research for historical phonetic maps. It allow users to navigate through the different historical phonetic maps and maneuver from one position number to another.
At the moment there is no suitable automated approach for the mapping of phonetic transcriptions to their corresponding position number on a historical phonetic map since it is a domain-specific task. The mapping is currently accomplished manually, which is a time-consuming and laborious process. This thesis develops and evaluates an automated method for the mapping of phonetic transcriptions to position numbers. We propose and compare two different approaches. Since this is a closed problem, the first approach is based on the concept of symbolic AI and classical CV. The second approach uses modern deep learning methods.

1.4. Outline

This thesis is structured as follows: Chapter 2 gives an outline of existing deep learning methods for the problem of image segmentation. Chapter 3 provides the theoretical background of a few computer vision methods and the basics of deep learning. In Chapter 4, we present the main contribution of this work: a methodology for the mapping of phonetic transcriptions to position numbers. This includes the corresponding preprocessing steps and a comparison of two different approaches. Chapter 5 presents the available data and analyzes the training of the neural networks. Furthermore, the results of the mapping of the two different approaches are evaluated and compared with each other. Finally, Chapter 6 concludes this work and discusses future work.

2. Related Work

This chapter provides an overview of some related work on deep learning for image segmentation. Since we can't cover all of the work, we present a few of the papers which are considered important for us. We group them into object localization, semantic segmentation and instance segmentation. The goal of object localization is to classify and localize each object in an image by creating exact bounding boxes around them. While semantic segmentation aims to locate every pixel corresponding to an object, instance segmentation even tries to apply different labels for different objects of the same class [GGOEO+17].

2.1. Object Localization

R-CNN

One of the first approaches to solving the problem of image segmentation with CNNs is called *R-CNN* and was developed by Girshick et al. [GDDM14]. Their goal was to use CNN to localize and segment objects. Figure 2.1 shows their methodology for this purpose. In the first step, the input image is preprocessed by the *Selective Search* [UVDSGS13]. By inspecting the image with windows of different sizes and grouping adjacent pixels according to certain properties such as textures or colors, so-called *region proposals* are generated for an input image. In the next step, these *region proposals* are transformed to the size of 227 x 227 pixels, regardless of size or aspect ratio, to be forwarded to a pre-trained CNN. A 4096-dimensional feature vector is extracted for every region proposal. Each feature vector is classified and scored by a Support Vector Machine (SVM) that is trained for every class. Bounding boxes are created by applying linear regression to each region proposal [GDDM14].

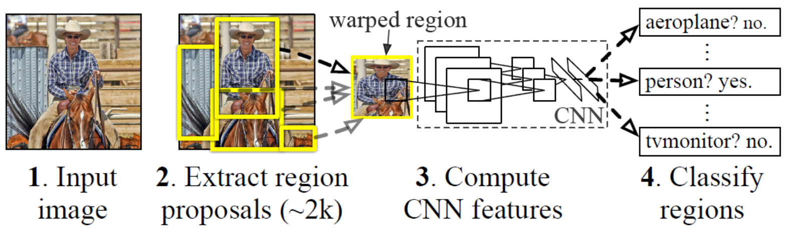


Figure 2.1.: R-CNN Methodology by Girshick et al. [GDDM14]

Fast R-CNN

Girshick et al. discovered some drawbacks for their proposed R-CNN [GDDM14], for each image every detected region proposal has to be forwarded to the CNN, which leads to long computation times. Furthermore, the localization process consists of several components, including a CNN, SVMs and bounding-box regressors. Hence R-CNN is not trained end-to-end. With *Fast R-CNN* Girshick et al. [Gir15] resolved these drawbacks and improved the speed and accuracy. Instead of processing all region proposals individually and using different components for the localization process, the whole image and all region proposals are used at once. A single network is employed to classify and build bounding boxes. Figure 2.2 illustrates the architecture of this method. The CNN takes the whole image as input and computes a *conv feature map*. A feature vector is extracted from the feature map for every class that uses a region of interest (ROI) pooling layer. By forwarding all feature vectors into a sequence of fully connected layers, two output layers are created. One for the classification and one for the prediction of the bounding boxes [Gir15].

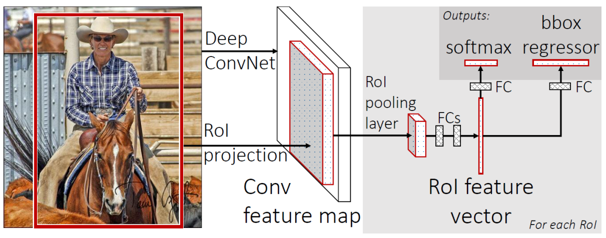


Figure 2.2.: Fast R-CNN Methodology by Girshick et. al. [Gir15]

Faster R-CNN

R-CNN and *Fast R-CNN* uses region proposals generated by the Selective Search [UVDSGS13]. Ren et al. [RHGS15] identified this method as a bottleneck of these approaches. The computation time for the region proposals consumes as much time as localization time. Therefore Ren et al. introduced *Faster R-CNN* [RHGS15], a Region Proposal Network (RPN) based on the *Fully-Convolutional Network*[LSD15] (Section 2.2) is used to create almost cost-free region proposals. As input, the RPN takes an image of arbitrary size and outputs object proposals along with the corresponding bounding box and objectness score. The objectness score represents the likelihood that an object is in the box. These outputs are used by the *Fast R-CNN* for detection. *Faster R-CNN* is a unified network, consisting of the RPN and the *Fast R-CNN*, both sharing the same convolutional layers, which results in an almost cost-free generation of region proposals. Figure 2.3 depicts the RPN. A sliding window is used on the convolutional feature map generated by the last shared convolutional layer. Each window returns **k** bounding boxes with their objectness score. Only those that are considered objects and have appropriate aspect ratios and sizes are passed to the *Fast R-CNN* as a lower-dimensional vector. As a result, the *Fast R-CNN* predicts classification scores and bounding boxes of an image. Overall, the quality of the region proposal and the detection accuracy are improved [RHGS15].

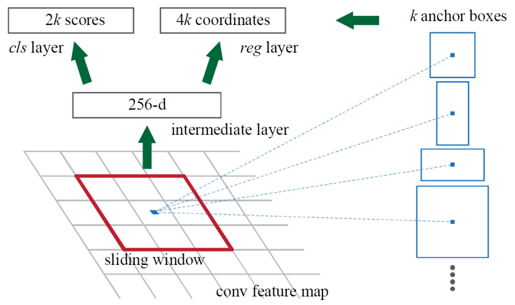


Figure 2.3.: Region Proposal Network (RPN) by Ren et al. [RHGS15]

YOLO - You Only Look Once

Redmon et al. [RDGF16] introduced another approach to object detection with *YOLO*, instead of performing the detection on region proposals, object detection is considered as a single regression problem. A CNN is trained to predict bounding boxes and the corresponding class probabilities. Figure 2.4 illustrates the methodology. The input image is divided into an S x S grid. For every grid cell, an arbitrary amount of bounding boxes are predicted, including their confidence score and class probabilities. Only bounding boxes with a confidence score above a specific threshold value are further processed.

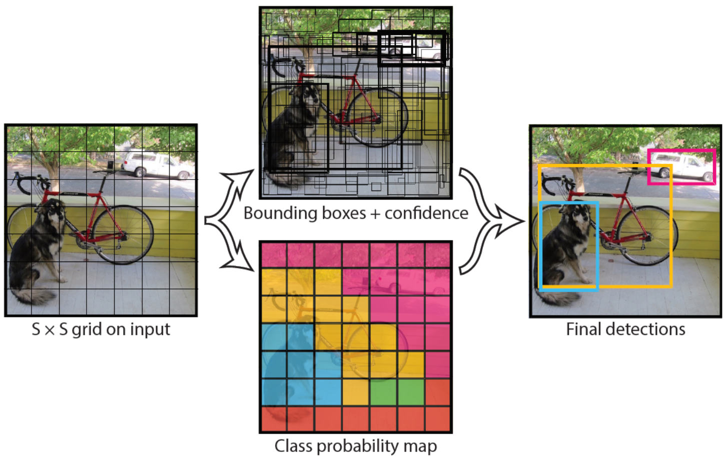


Figure 2.4.: YOLO Model by Redmon et al. [RDGF16]

The whole image is only processed once by the CNN, so YOLO sees the entire image and can encode class information during training and testing. Furthermore it is generalizable as it learns general representations of objects. In terms of processing time, *YOLO* is extremely fast because a new image is simply processed by the trained CNN. Compared to *Fast R-CNN* [Gir15] (Figure 2.5), the localization errors are increased, but the number of false positives in the background is decreased.

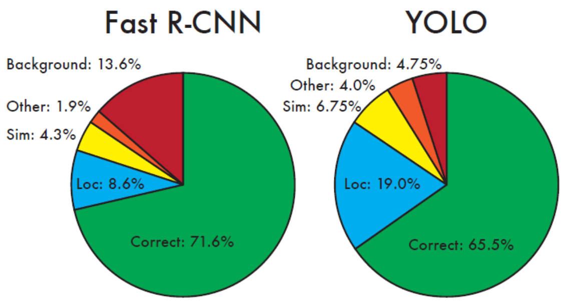


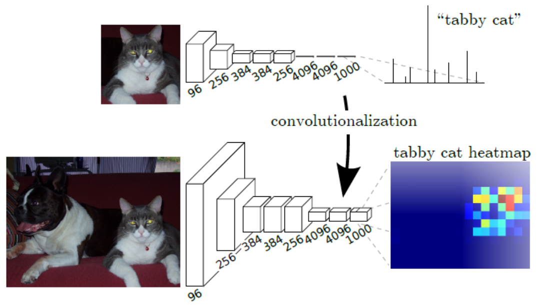
Figure 2.5.: Error Comparison of YOLO [RDGF16] with Fast R-CNN [Gir15] by Redmon et al. [RDGF16]

YOLO also has limitations, since each grid is only capable of predicting two boxes and assigned to one class, the amount of predicted nearby objects is limited. It also has problems with the detection of small objects appearing in groups [RDGF16].

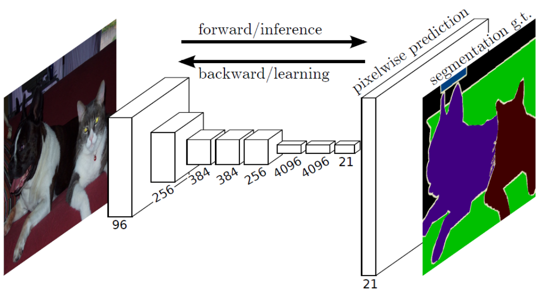
2.2. Semantic Segmentation

Fully Convolutional Network

Garcia et al. [LSD15] defined the *Fully Convolutional Network (FCN)* as the foundation for the latest most successful deep learning methodologies for semantic segmentation [GGOEO$^+$17]. Long et al. [LSD15] trained an end-to-end network for pixelwise prediction and named it *Fully Convolutional Network*. The *FCN* takes an input of arbitrary size and generates a fully segmented image of a corresponding size. In this network, all fully connected layers are replaced by convolutional layers. Hence, instead of a flat non-spatial output, a heatmap is outputted that retains the spatial coordinates (Figure 2.6a). The heatmap represents many small feature maps and requires upsampling to restore the original size. Upsampling, also called deconvolution, consists of a convolutional layer with a fractional stride. In every upsampling step, an output with a larger size than the input is created. This enables an end-to-end learning using the pixelwise loss (Figure 2.6b) [LSD15]. Since the multiple upsampling steps create a coarse output, Long et al. introduced a skip architecture to improve semantic context and localization accuracy. By combining fine layers with coarse layers, the location information of the fine layers is not lost when going deeper, while at the same time deep features are obtained. Figure 2.7 illustrates the segmentation result of different FCN models. The *FCN-32s* does not use the skip architecture at all and performs the upsampling 32 times, while the *FCN-16s* and *FCN-8s* use the skip architecture in different depth. The *FCN-8s* achieved the best result, while the result of *FCN-32s* is coarse due to the loss of location information [LSD15].



(a) Transforming Fully Connected Layers Into Convolution Layers



(b) FCN Architecture Without Skip

Figure 2.6.: Fully Convolutional Network by Long et al. [LSD15]

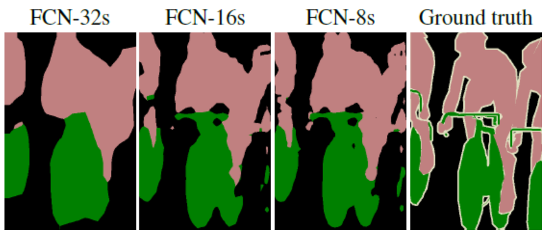
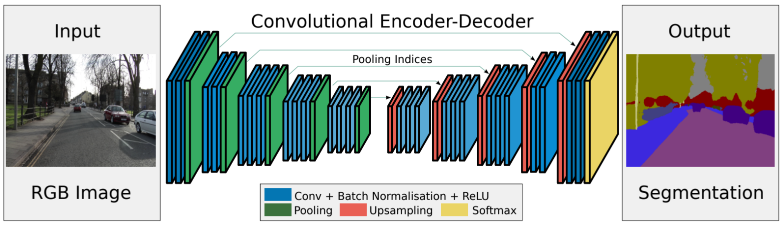


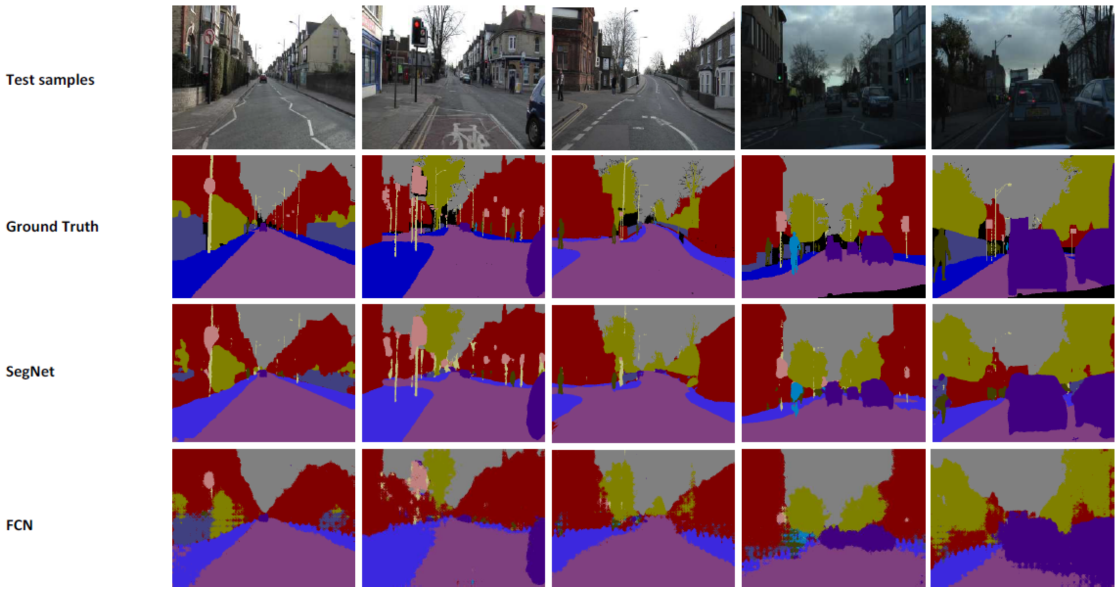
Figure 2.7.: Tests of different FCN Models [LSD15]

SegNet

Based on *FCN* [LSD15] Badrinarayanan et al. [BKC17] developed *SegNet*, a convolutional encoder-decoder architecture for segmentation. Figure 2.8a shows the architecture of the model, the last layer is a softmax layer, which enables pixel-wise classification. The difference to *FCN* is in the decoder part. Every upsampling layer corresponds to one pooling layer in the encoder. Only the max-pooling indices are used for upsampling the feature maps, reducing the number of trainable parameters while maintaining high-level details. To construct a segmented input image, the resulting feature maps are convolved with a trainable decoder filter bank. Due to the decoder size, the accuracy is higher compared to *FCN*. Also, less memory is used during the prediction because only the max-pooling indices are stored, in contrast to *FCN*, which stores the full encoder feature maps. However, the forward time is slower due to more convolution layers in the decoder. Figure 2.8b shows a comparison of the two methods. Overall, the segmentation results of *SegNet* are more superior to *FCN*, especially with regard to the definition of object boundaries [BKC17].



(a) Network Architecture

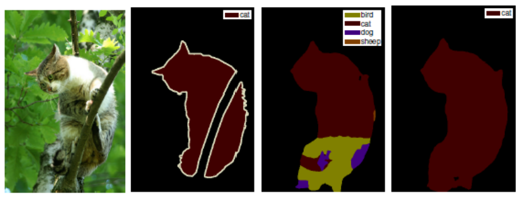


(b) SegNet Results Compared To *FCN* [LSD15] Results

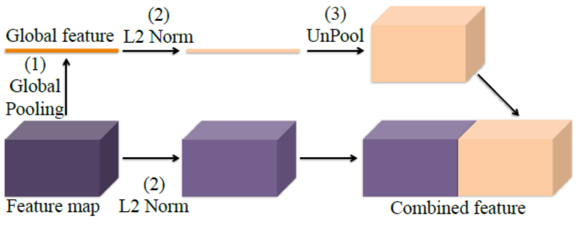
Figure 2.8.: SegNet by Badrinarayanan et al. [BKC17]

ParseNet

According to Liu et al. [LRB15] the *FCN* disregards the global context of the input image since it only considers the largest receptive field. A more global context can improve the results of segmentation tasks. Therefore *ParseNet* was introduced, an end-to-end FCN for semantic segmentation. It takes the whole image as input in order not to lose global information and simultaneously predict all pixel values. By adding a contexture module (Figure 2.9e) to a convolutional layer, context features can be used to improve the segmentation results. The module takes feature maps as input. In the first step, a context vector is generated with global pooling. Then the original feature map and the pooled context vector are normalized using *L2 normalization*. The normalization scales different sizes of feature maps that are combined for the prediction. In the next step, the scaled vector is unpooled to generate a new feature map of the same size as the original input feature map, which is concatenated to the original input in the last step. Figure 2.9 shows the result of a segmentation example with *FCN* and *ParseNet*.



(a) Image (b) Truth (c) FCN (d) ParseNet



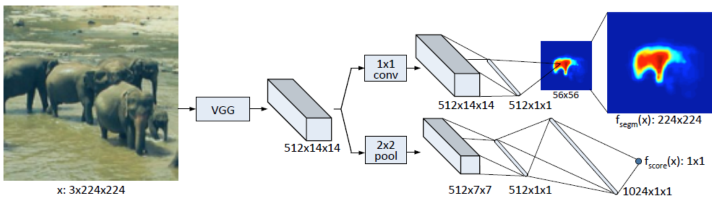
(e) ParseNet contexture module overview.

Figure 2.9.: Segmentation Comparison between FCN & ParseNet and Overview of the ParseNet Module by Liu et. al. [LRB15]

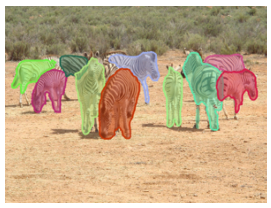
2.3. Instance Segmentation

Deep Mask

Instance Segmentation is viewed as the next level of semantic segmentation. Objects of the same class are not only segmented but also differentiated into individual instances [GGOEO$^+$17]. Pinheiro et al. [PCD15] proposed a single based CNN for this problem with *Deep Mask*. Figure 2.10a illustrates the network architecture. For an input patch, the model predicts a segmentation mask and the corresponding likelihood of containing an instance. The prediction tasks share most layers of the network, except the last, and are therefore trained together. At the last shared feature layer, the network is divided into two parts. The first predicts the segmentation mask of the input patch, while the second predicts the likelihood of containing an object [PCD15]. Figure 2.10b shows an example of an instance segmentation.



(a) Network Architecture



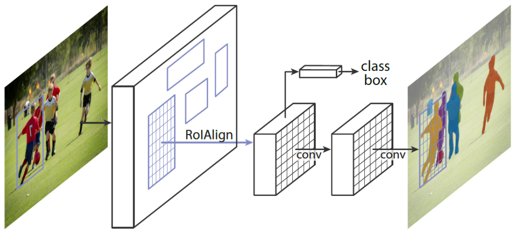
(b) Instance Segmentation Results

Figure 2.10.: Deep Mask by Pinheiro et al. [PCD15]

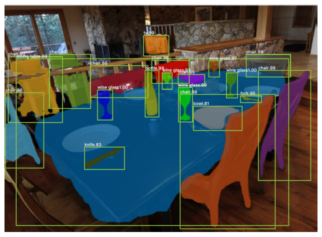
Mask R-CNN

Based on *Faster R-CNN* [RHGS15] He et al. [HGDG17] developed a method called Mask R-CNN that adds an additional output branch to the initial method to predict an object mask for a *Region of Interest (ROI)* generated by the Region Proposal Network (RPN). The other two branches predict in parallel the corresponding bounding box and classification score (Figure 2.11a). The segmented mask is generated by a *FCN*. Since

Faster R-CNN was not designed to segment pixel by pixel, the RoI Pooling Layer is replaced by the *RoIAlign Layer* to correct misalignment of pixels by maintaining the precise spatial locations. The decoupling of mask and class predictions results in better performance when segmenting instances. A binary mask is predicted independently for each class. This is achieved by a multi-task loss that combines the prediction loss of the bounding box, class and segmentation mask. During training, the model tries to optimize each loss separately, resulting in better overall performance. Figure 2.11b shows an example of the instance segmentation results, for each instance, a bounding box is created with the corresponding likelihood and segmentation.



(a) Network Architecture



(b) Instance Segmentation Results

Figure 2.11.: Mask R-CNN by He et al. [HGDG17]

2.4. Classification of this Work

This chapter introduced several methods for the different areas in image segmentation. In our work, we focus on the area of object localization and semantic segmentation. For this purpose classical Computer Vision methods as well as the *Fully Convolutional Network* are used. The next chapter covers the basics of all methods employed in this work.

3. Fundamentals

In this chapter the historical phonetic maps are explained. Furthermore, we cover several Computer Vision methods used in this thesis and the basic concepts of Deep Learning.

3.1. Historical Phonetic Maps

The foundation of this work consists of images of historical phonetic maps. The maps originate from the *VerbaAlpina-Project* [KL14] and display the geographical map of southern Switzerland and Italy. Each map represents the phonetic transcriptions of different localities for a specific term. The phonetic transcriptions are displayed as black text, localities as red numbers. At the top left of the map, the corresponding term is displayed on the map. Below the term there is a legend that provides additional information. Figure 3.1 shows a historical phonetic map on which the term *father* is explored.

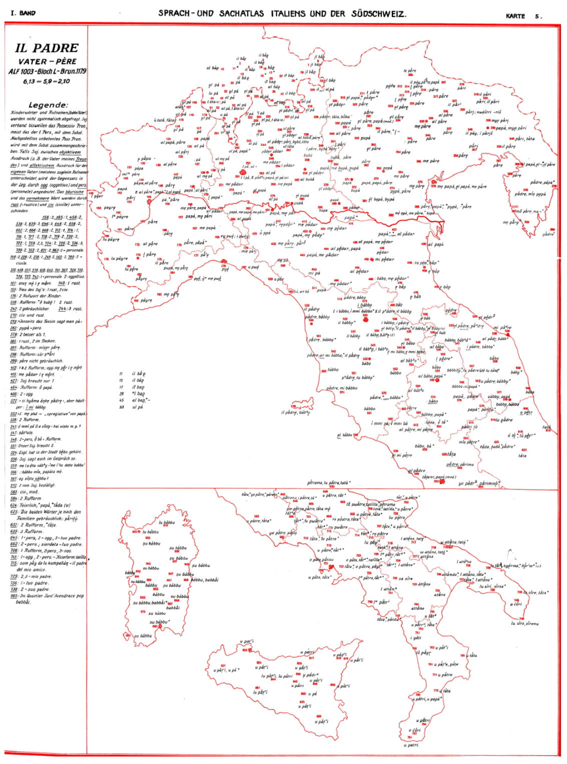


Figure 3.1.: Example of a historical phonetically Map displaying the term *father*

3.2. Computer Vision

This section provides information about the image processing framework used and an overview of the various image processing methods.

3.2.1. OpenCV

OpenCV is an open-source computer vision library. It provides an infrastructure for computer vision applications by implementing algorithms than can solve computer-vision problems in a computationally efficient and quick way. The library contains low-level image processing algorithms as well as high-level algorithms for a wide range of topics. Starting with basic image processing methods up to object detection and extraction and many other functions [Tea19b] [BK08][Lag14].

3.2.2. Image Morphology

The processing of digital images can be approached with mathematical morphology. Mathematical morphological operators can transform image data into a simpler representation by preserving shape characteristics and removing noise [HSZ87]. With the help of a structuring element/kernel defined as a set of pixels or a shape and a defined anchor point, morphological operations can be performed. When a given pixel is aligned with the anchor point, the pixels of the intersection with the image are used to apply a morphological operation. The shape of the structuring element is not predetermined but can be any shape, mostly is consists of elementary shapes such as square, circle or diamond [Lag14]. In the following we introduce with *dilatation* and *erosion* two morphological operators. They are used for noise removing, isolating of particular elements or for connecting inconsistent elements in an image [BK08].

Dilation

The dilation operator is also called the local maximum operator. It expands the region on which it is applied and tends to smooth concavities. For this purpose, the maximum pixel value, computed from the intersection of the structuring element with the region, is used to replace the pixel value in the anchor point [BK08].

$$dilate(x, y) = \max_{(x',y') \in kernel} src(x + x', y + y')$$

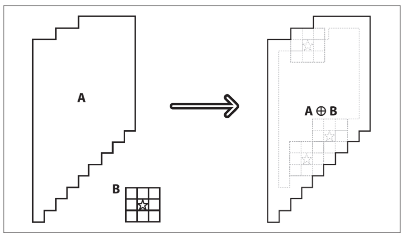


Figure 3.2.: Example of the dilation operator [BK08]

Erosion

Erosion is the complementary operator to dilation. It is also referred to as the local minimum operator and reduces the region on which it is applied and tends to smooth protrusions away. The minimum pixel value is computed from the overlap of the structuring element with the region. The image pixel in the anchor point is replaced by the minimum value [BK08].

$$erode(x, y) = \min_{(x',y') \in kernel} src(x + x', y + y')$$

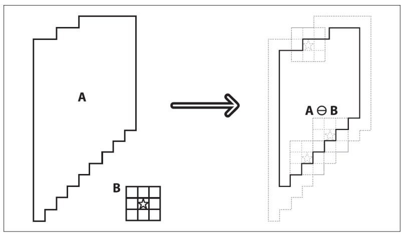


Figure 3.3.: Example of the erosion operator [BK08]

Opening

The dilation of the erosion of an image is called opening [Lag14]. It is mainly used to count areas in a binary image and also to remove individual outliers that have a higher pixel value than their neighborhood [BK08] (Figure 3.4) .

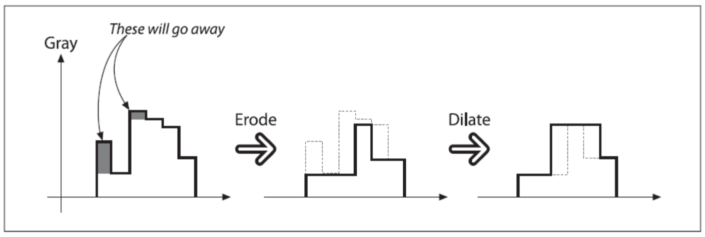


Figure 3.4.: Opening Operation: Elimination of the upward Outliers [BK08]

Closing

While the dilation of the erosion of an image is called opening, the erosion of the dilation of an image is called closing [Lag14]. It is mainly used to remove noise-driven or undesired sections. Lone outliers that have a lower pixel value than their neighborhood are eliminated [BK08] (Figure 3.5).

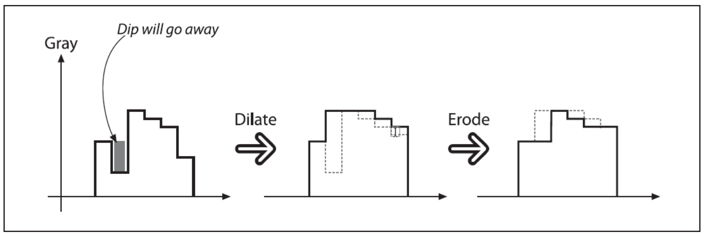


Figure 3.5.: Closing Operation: Elimination of the downward Outliers [BK08]

3.2.3. Image Thresholding

Another image processing method is called thresholding. It is used to categorize pixels into two classes that are either below or above a certain threshold value [BK08]. We only consider the binary variants of thresholding since the output is a binary image and can, therefore, separate foreground objects from the background [SS04]. Figure 3.6 depicts an example of the binary and inverse binary thresholding.

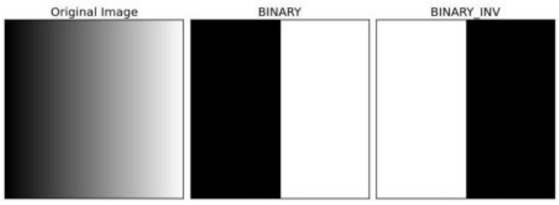


Figure 3.6.: Example of the Binary and Inverse Binary Thresholding [Tea19a]

In the case of binary thresholding, pixels below a certain threshold get a black color value of 0, while the others get a white color value of 255. As seen in equation 3.1 for binary thresholding, the threshold value can be freely chosen as well as the color value M for a pixel that is above the threshold. The color value for a pixel that is below the threshold is always black.

$$dst(x,y) = \begin{cases} M, & \text{if src(x,y)} > threshold \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

3.2.4. Connected Components

Areas of adjacent pixels are denoted as *connected components*. A pixel a is adjacent to pixel b only if it is an immediate neighbor of b and both pixels have the same color value. Connected components can be used to find letters in a document or objects in an image. The computation of these components is divided into two phases. In the first phase, adjacent pixels are labeled with unique labels, the pixels are accessed horizontally first. Whenever it is possible labels of vertically adjacent pixels are reused. In the second phase, pixels with different labels but the same color value are merged [Sze10]. Figure 3.7 shows an example of the entire process with a grayscale input image containing four connected components (Figure 3.7a). First all horizontal, equal pixels are labeled (Figure 3.7b) and then connected (Figure 3.7c).

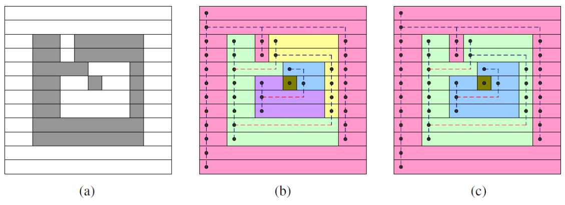


Figure 3.7.: Example of the Connected Component Computation [Sze10]

3.2.5. Image Transformation

The technique for changing an image into an alternate representation of data is called image transformation. As a result of this transformation, new unseen features of the original data may arise [BK08]. There exists a large number of different transformations in computer vision. In the following, the edge and line detection methods are further explained.

Edge Detection

At borders between areas of different colors, intensity and shape, edges can occur. With the computation of derivatives, it is possible to detect the edges. The gradient can be used to determine the positions of these edges [Sze10]. In 1986 Canny [Can87] developed the *Canny algorithm* for edge detection. The first derivatives are calculated in horizontal and vertical direction and then combined into vertical, horizontal and two diagonal directions. The points of the local maxima of these derivatives are considered as possible edge candidates. By applying a hysteresis threshold to these points, where there is an upper and lower threshold, the Canny algorithm attempts to group the candidates. If the gradient of a point is higher than the upper threshold, it is seen as an edge point, if it is below the lower threshold, it is declined. If the gradient is in between the two thresholds, the point is accepted as an edge when it is connected to an edge point [BK08].

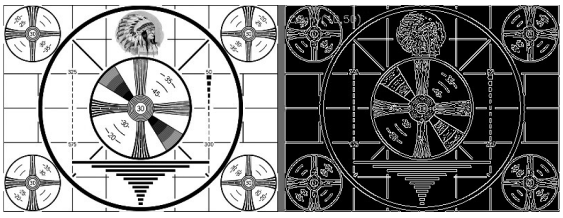


Figure 3.8.: Application Example of the Canny Edge Detection [BK08]

Line Detection

A method for detecting lines in an image was developed by Hough et al. [Hou62] in 1962, called *Hough transform*. It groups edges into lines but can also find circles and other simple forms. In this work we use the *Hough line transform* method [BK08]. For every edge point, all possible lines that can pass through it are considered [Sze10]. If each line is parameterized by a slope a a and an intercept b then for every point a, locus of points can be built, which is localized in the (a,b) plane. Every locus corresponds to all lines passing through the corresponding point. With the conversion of all nonzero pixels into a locus of points and the sum over all, local maxima appear in the (a,b) plane that corresponds to lines in the image. Due to the summing of all loci, the (a,b) plane is called the *accumulator plane*.

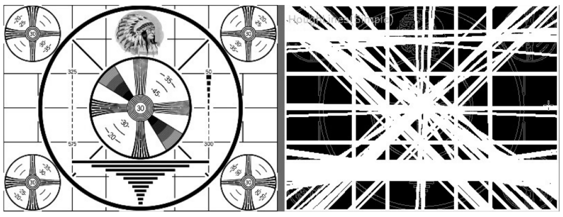


Figure 3.9.: Application Example of the Hough Line Transformation [BK08]

Because of the various densities of the lines depending on the slope and the interval of possible values for a slope ($-\infty - +\infty$), the slope-intercept form is not the most suitable. Therefore the representation of each line as point in polar coordinates(ρ, Θ) is more appropriate, where each line goes through the given point and is orthogonal to the radial from the origin to the point [BK08]. Θ represents the angle and ρ the perpendicular distance of the line from the origin. The following equation denotes a line:

$$\rho = xcos(\Theta) + ysin(\Theta)$$

3.2.6. Contour

As seen in Chapter 3.2.4 a method to find related pixels is called *connected components*. However, a closed structure with multiple connected components cannot be recognized as a whole component. Hence contours are introduced. A contour presents a curve in an image as a list of points, whereby the curve can be of any shape. Each point in the list has knowledge about the location of the next point on the curve. The concept of viewing a contour as a tree is important to find contours. The *contour tree* contains the relationship and the structure of the individual connected components. Figure 3.10 represents the methodology for finding contours. The input image contains 5 white regions and is shown in the upper part of the figure. The lower figure depicts the different levels of contours, where the contours are marked with "c", the holes with "h" and the level with the length of numbers. Here the contour "c0" is the root node and has the holes "h00' and "h01' as his children. These holes in return have the contours "c000" and "c010" as its children. The contour tree is complete when the leaf is reached [BK08]. A leaf can also be seen as a *connected component*.

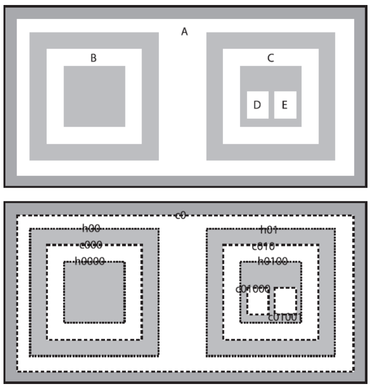


Figure 3.10.: Example of the finding contours method [BK08]

3.3. Deep Learning

In this section first, the relation of the terms artificial intelligence, machine learning and deep learning are shown and then defined. Afterward, we briefly cover the basics of deep learning and convolutional neural networks. In the end, the concept of image segmentation is explained.

3.3.1. Definition

Nowadays the terms *Artificial Intelligence (AI)*, *Machine Learning (ML)* and *Deep Learning (DL)* are often used in the same context when talking about latest technological developments such as autonomous driving, face recognition or recommendation systems, without even knowing their true meaning. Figure 3.11 shows the relationship between these terms.

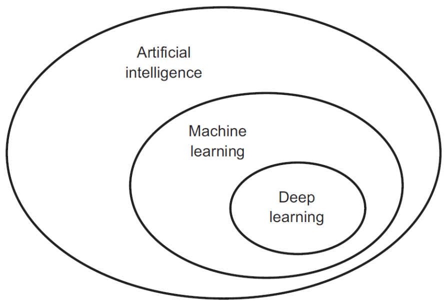


Figure 3.11.: Relationship between the terms Artificial Intelligence, Machine Learning and Deep Learning [Cho18]

Artificial Intelligence

Chollet [Cho18] defined AI as "the effort to automate intellectual tasks normally performed by humans". Artificial intelligence is a general area that includes machine learning and deep learning as well as tasks not involved with learning, such as rule-based approaches. These approaches are called *symbolic AI*. By providing a very large set of explicit rules, a specific domain should be manipulated or controlled. Early chess programs, for example, contain a large number of hard-coded rules to assess the game situation and making the best possible move. Well-defined and logical problems are the best application for symbolic AI. For more complex and fuzzy problems, like image classification or natural language processing, *symbolic AI* reaches its limit since the amount of needed rules are countless and in most cases indeterminable [Cho18].

Machine Learning

To overcome the hurdle of indeterminable and countless rules Machine Learning comes to use. The *Symbolic AI* approach requires predefined rules in order to process the incoming data. In ML the task performance is seen with a different perspective, the computer should learn the corresponding rules to a task by himself (Figure 3.12). By providing input data, the expected outcome and a measurement a machine-learning system can learn useful representations of the input data to produce the correct output. To determine the quality of the machine-learning algorithm the measurement is needed as the produced output is compared with the expected output. The result is used for adjustment and improvement to the algorithm, this process is called learning [Cho18].

Deep Learning

While other machine learning algorithms mostly focusing on learning one or two layers of representations of the data, deep learning is trying to learn multiple representations by stacking up layers of representations. Hence the word *deep* in the name. Typically these layered representations are learned with *neural networks* (NNs). A deep neural network

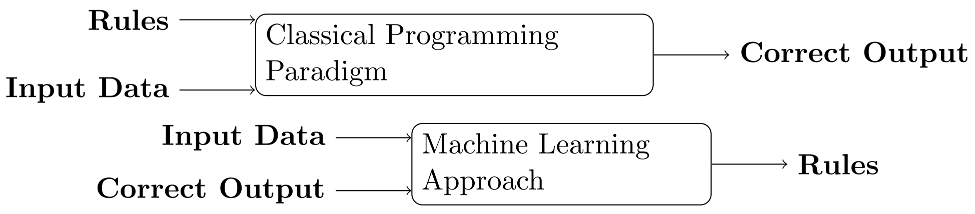


Figure 3.12.: Rule-Based Approach vs. Machine Learning Approach [Cho18]

consists of multiple layers (Figure 3.13) in which each layer tries to learn a different, useful representation of the input data [Cho18]. The deeper the layer, the more finely granular is the representation of the original input.

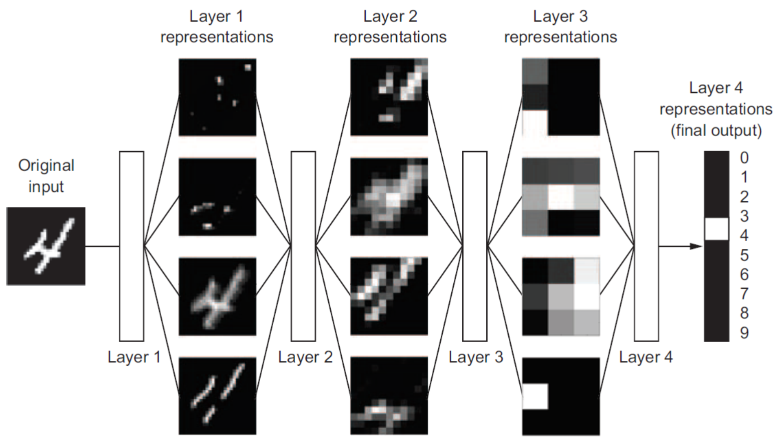


Figure 3.13.: Learning of Deep Representations with the help of a number classification model [Cho18]

3.3.2. Training

Figure 3.14 visualizes the training process of a neural network (NN) with input $\mathbf{X}$, output $\hat{\mathbf{Y}}$ and ground truth $\mathbf{Y}$. The layers between the input and output are called fully connected hidden layers. Every layer consists of $\mathbf{n} \in \mathbb{N}$ units and is connected to the previous layer via a weight matrix $\mathbf{W} \in \mathbb{R}^{\mathrm{m\ x\ n}}$, where $\mathbf{m}$ is the number of units and $\mathbf{n}$ the number of units of the previous layer. With the help of the weight matrices, the state of each hidden layer can be calculated by computing the weighted sum of all its input. As for the first layer, the state is calculated by multiplying its weight matrix $\mathbf{W_1}$ with the input layer $\mathbf{X}$. All states for every layer will be computed consecutively so that the result $\mathbf{Z}$ can be used to calculate output $\hat{\mathbf{Y}}$. Hence the input $\mathbf{X}$ needs to be passed through all the layers. This chain of computation is called forward-propagation.

$$Z = \sum W_i X_i$$

For the sake of simplicity the bias term **b** is omitted. A non-linear function (e.g. ReLU) also called activation function σ is applied to the output of the last layer **Z** to compute the output $\hat{\mathbf{Y}}$ [LBH15]. Activation functions are needed to decide whether a unit should be activated or not, in other words, whether the information the unit contains is important for the training process. The output $\hat{\mathbf{Y}}$ of a neural network is also called prediction.

$$\hat{Y} = \sigma(Z)$$

With a loss function **L** a score can be calculated which represents the distance between the prediction $\hat{\mathbf{Y}}$ and the ground truth **Y**. Based on this score the performance of the network can be optimized. The score serves as feedback for adjustments of the weights to minimize the loss score so that the difference between prediction and ground truth is as small as possible. A method for loss-minimization is gradient descent in particular back-propagation. In an iterative way the weights in the network can be updated. With the loss score, the gradients of all weights can be calculated and used to update the weights according to a learning rate. The repetition of forward- and back-propagation represents the training of a neural network. This process is repeated several times until the lowest possible loss score is reached.

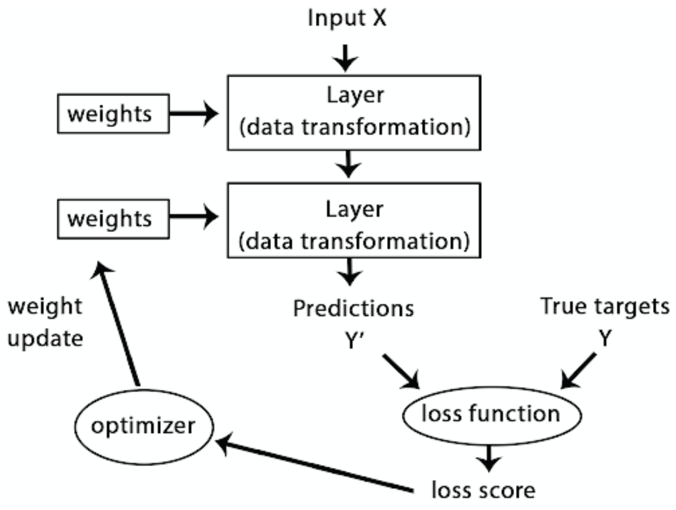


Figure 3.14.: Training's process of a Neural Network [Cho18]

There are two main types of learning. We distinguish between supervised and unsupervised learning.

Supervised Learning

Supervised learning aims at learning a function $\mathbf{f}$ to map input data $\mathbf{X}$ to output variables $\mathbf{Y}$. The name supervised derives from the knowledge of output variables beforehand. Typically it is used in the area of classification or regression [Cho18].

$$Y = f(X)$$

Unsupervised Learning

With unsupervised learning, however, only the input data X is available and no knowledge about the corresponding output variables Y is given. The aim is to discover the data structure and extract the features of the input data independently of targets. Clustering is a typical use case [Cho18].

3.3.3. Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a specialized type of neural network for processing N-dimensional data. The name for the network originates from the mathematical operation named convolution, a specialized type of linear operation [GBC16]. Successful applications of CNNs are in the field of image classification and object detection, as the strength of CNN is to extract features with complex shapes or color patterns. A typical CNN consists of two parts (see figure 3.15). The first one is composed of two specific layers: convolutional and pooling layers (sometimes called subsampling). The convolutional layer is structured in units that are organized in feature maps. Via a set of weights each unit is connected to sets in the feature maps of the previous layer. All units in a feature map share the same weights since they try to detect the same features at different locations in the array data. These features are mostly distinct because many parts of the input data are often correlated. In sum, the task of the convolutional layer is to detect connections of features from the previous layer. Whereas the pooling layer attempts to merge semantically related features into a single one to reduce the dimensionality of feature maps and still keeping the essential features [LBH15]. This makes the representation more invariant to small changes in the input [GBC16]. The invariance to the location of these patterns also supports the principle of weight sharing of the units despite their location. A frequently used pooling method is the *max-pooling*. It applies the *max* function on a patch of units in one feature map and only stores the maximum. By having multiple consecutive stages of convolutional and pooling layers, the network is able to learn more complex patterns. These consecutive stages are also called *hidden layers*. The second part represents the *fully-connected layers* that require one-dimensional data as input. Hence a flatten layer is applied before getting to the fully connected layers. The last fully connected layers produce the output of the network. Figure 3.15 shows LeNet-5, a CNN for digit classification developed by LeCun et al. [LBB+98], the output of this network contains the confidence levels for the different digits.

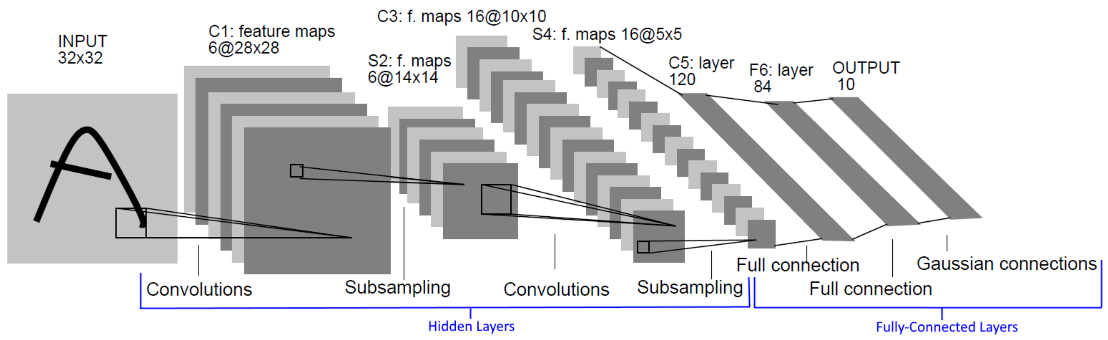


Figure 3.15.: LeNet-5, a Convolutional Neural Network for digits recognition by Yann LeCun [LBB+98]

3.3.4. Data Augmentation

When training a neural network, typical problems such as overfitting can arise. Data augmentation is a technique to counter this problem. It can be used as a regularizer to prevent overfitting and at the same time to improve the generalization capabilities [GGOEO+17, WGSM16]. It can also improve the class imbalance problems by generating more training samples from the imbalance class. The generation of additional training samples is typically performed by an arbitrarily number of transformations to the data. Rotation, shifting and translation are only a few possible transformations [WGSM16].

3.4. Image Segmentation

Image segmentation is another key problem in the area of computer vision. It represents the division of an image into meaningful entities. For a human, it is easy to detect patterns or group related objects in an image without having explicit knowledge of the content [ZMCL16]. The visual system of a human automatically processes the perception of the image based on the Gestalt laws [Tod08]. As far as the computer is concerned, it is a difficult task, as the Gestalt laws do not apply, but for many applications, it is important to understand the whole image. Autonomous driving or human-machine interaction are only a few examples. With the advancement in DL and the development of CNNs not only the image classification task can be tackled, but also the image segmentation problem.
Figure 3.16 represents the different possibilities for object recognition, from a coarse-grained method like classification to fine-grained method like instance segmentation. The classification makes a prediction about the whole image without having information about localization or detection. Semantic segmentation, on the other hand, predicts the corresponding class for each pixel so that objects or regions can be recognized. The

instance segmentation even detects the instance of each known object for every pixel [GGOEO+17]. In this work, we focus on semantic segmentation.

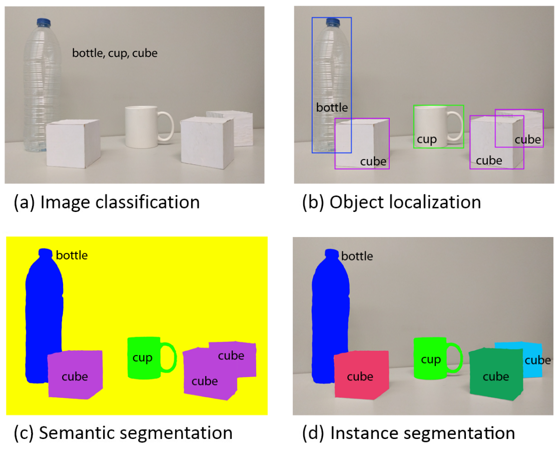


Figure 3.16.: Development of Object Recognition: from Classification to Segmentation [GGOEO+17]

3.5. Dice Loss

In order to achieve good segmentation prediction results, not only the network architecture plays an important role, but also the choice of loss function. A neural network can get stuck in local minima during the learning process due to the apparent importance of large background areas while small foreground areas are more relevant. Sometimes these foreground areas are completely missing or only partially detected. The predictions of a network would be strongly biased towards the background area. Some previous work approaches this problem with re-weighting the importance towards the foreground areas [MNA16]. Milletari et al. [MNA16] proposed the Dice Loss (DL) function, which does not require any reassignment of weights to achieve a correct balance between foreground and background areas.

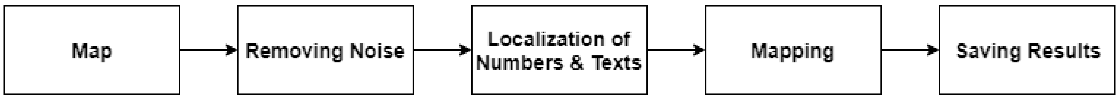
$$DL = 1 - D$$

The loss function is based on the dice coefficient D with the value range from 0 to 1. The dice coefficient between two binary areas consists of the sum over N pixels, the predicted binary segmentation area $p_i \in P$ and the ground truth binary area $g_i \in G$. It is a good metric for the segmentation performance, but can only be applied if the ground truth is available [SLV+17].

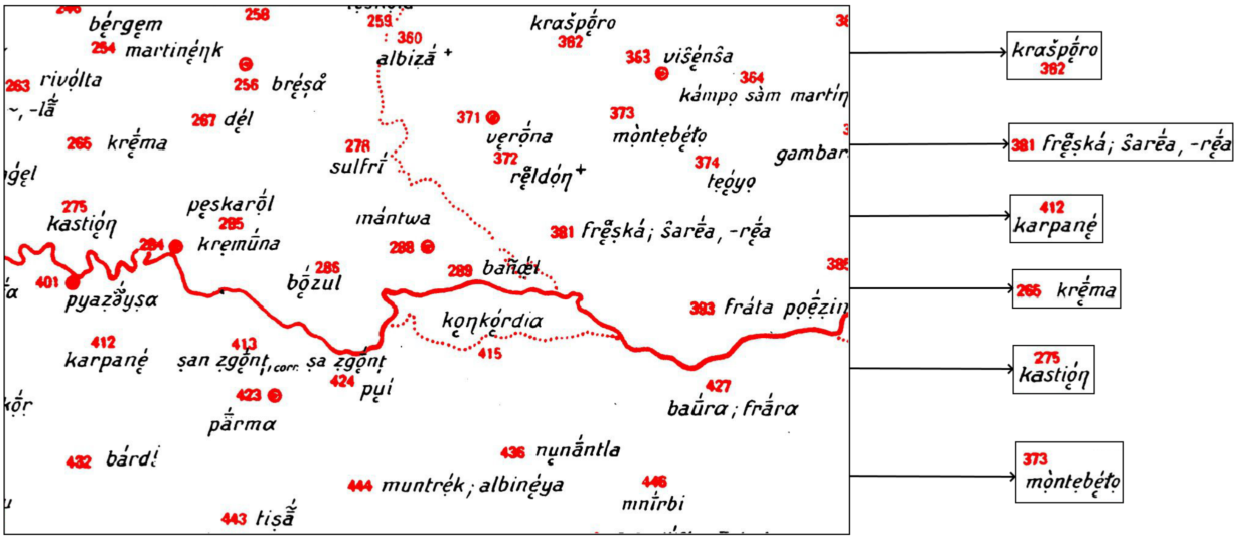
$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

4. Methodology

In this chapter, we present a methodology for mapping phonetic transcriptions to corresponding position numbers on a historical phonetic map. Figure 4.1a illustrates the methodology, it consists of 4 steps. In the first step, we try to remove any noise that might interfere with the mapping process. The left side of Figure 4.1b shows a section of a map, in which noise is visible in the form of boundaries or geometric figures such as circles. Based on a cleaned map, the localization of position numbers and phonetic transcriptions is easier in the next step. This localization information is used in the final step to map numbers and phonetic transcriptions. We save each mapped pair as a separate image file. Figure 4.1b shows the objective output of the methodology, multiple images of number and text pairs to be stored.



(a) Methodology



(b) Goal of the Method

Figure 4.1.: Methodology for the mapping of position numbers and phonetic transcriptions

We propose two different approaches for the first two processing steps (Figure 4.2). The first approach is based on the expert system concept of symbolic AI, also known as a knowledge-based system. Expert systems aim to incorporate the knowledge of a human expert to solve a specific domain problem [Tri11]. Since we explore a closed domain-specific problem with the removing of noise and the localization of numbers and texts on a historical phonetic map, the development of a rule-based expert system is useful. The second approach focuses on the successor of symbolic AI, Deep Learning. We try to approach the same problem with neural networks.

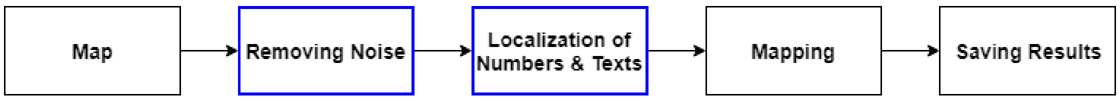


Figure 4.2.: Map Processing: Noise Removing and Number/Text Localization

Before we can process the map as an image, we need to convert the maps to PNG format, as they are available in PDF format. For this purpose we use *ImageMagick*[1], an open-source tool for converting images. The image processing is done with OpenCV.
At first, the map is read as a color image, since there are two colors on the map, red and black. All texts, including phonetic transcriptions, are in black. In contrast, the position numbers and any noise are displayed in red. Therefore removing of this noise is important because it could interfere with the localization of numbers.
In the following, we refer to the OpenCV color format, which uses the BGR format instead of the RGB format. Hence all color values in the next sections are in the BGR format. The red color space ranges over several values, as not every red pixel in the image have the value [0,0,255]. The same applies to the black color space, thus we define a threshold range for red and black. For the red color, the lower threshold starts at [0,0,60] and ends at the upper threshold [55,55,255], each pixel value within this threshold is considered red. Analog for black the lower threshold starts at [0,0,0] and the upper threshold ends at [55,55,55]. We extract red and black separately, wherefore the original map image is divided into two new images, which serve as input for the two proposed approaches. Henceforth the image containing only black pixel is referred to as **B** and the image containing only red pixel as **R**. For a faster computation time we work with binary images, instead of three color channels only one is calculated. Therefore the binary thresholding method is performed for **B** and **R**. The new resulting images $\mathbf{B}_{\mathbf{threshold}}$ and $\mathbf{R}_{\mathbf{threshold}}$ only contain white and black pixels, white for text respectively red pixels, black for everything else. Figure 4.3 depicts an example of the color extracted, thresholded images.

4.1. Rule-Based Approach

Figure 4.4 presents the methodology for the rule-based approach of removing noise and localization of numbers and texts. **B** and **R** are processed differently.

Since **B** contains no noise, only the noise in **R** has to be cleaned. First, we search for all connected components in $\mathbf{R}_{\mathbf{threshold}}$. The method implemented in OpenCV returns

[1]`https://imagemagick.org/`

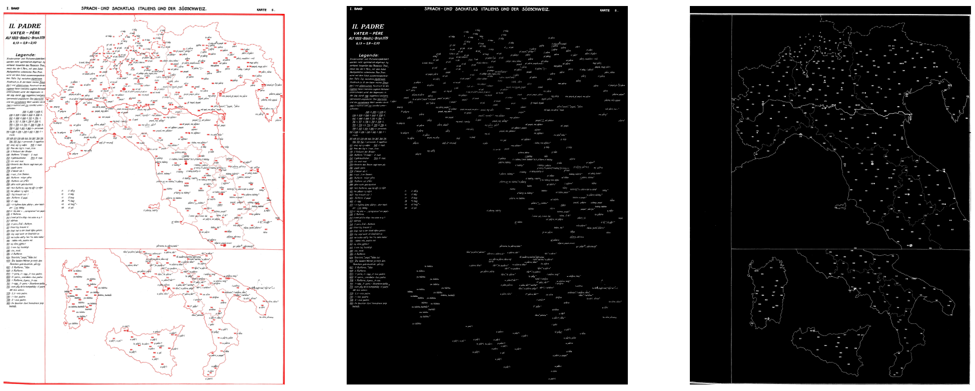

(a) Original Image (b) Text Image **B** (c) Number Image **R**

Figure 4.3.: Example of the Color Extraction

the bounding box for each found connected component, where for each box we have the position, the width, the height and the total area in pixels. Based on these properties, an expert system for noise classification is developed. We compare the width, height, ratio and area of pixels of a digit against that of noise, and then define decision rules. In an iterative process, these rules are improved and extended so that as much noise as possible is detected. Since the properties of bounding boxes are not sufficient to detect every form of noise, an additional method is used. The *line detection* algorithm is applied to detect all lines for each component that is not recognized as noise by its bounding box properties. Using the angle Θ and the line distance ρ of each detected line, we define another set of decision rules for the classification. These rules are created by analyzing Θ and ρ values of digits and noise and improved in an iterative process. The same procedure is repeated once to define another set of rules that removes the remaining noise. All pixels, classified as noise, become black in **R**.
In the next step, we try to detect all numbers. Therefore we again use the thresholding technique for **R** to get an updated version of $\mathbf{R_{threshold}}$. Since a number can consist of multiple digits, $\mathbf{R_{threshold}}$ is dilated with a kernel multiple times to find related digits. Dilation ensures that close pixels overlap to allow digits that are close together to be recognized as a number (Figure 4.5).

We find these new overlapping by finding the contours of them, as each overlapping represents a number. For every found contour we have the position, the height and the width. Contours not corresponding to the length or the height of a number are discarded, i.e. their pixels in **R** become black. All other contours/numbers are saved for the mapping process.
In the last step, we try to detect every phonetic transcription in **B**. Just as numbers consist of multiple digits, a phonetic transcription consists of multiple words and characters. Therefore **B** is also dilated to find related words and characters that form a phonetic transcription. For retrieving the contours the finding contours method is used again. Based on the width and height of the contour, a last filtering mechanism is per-

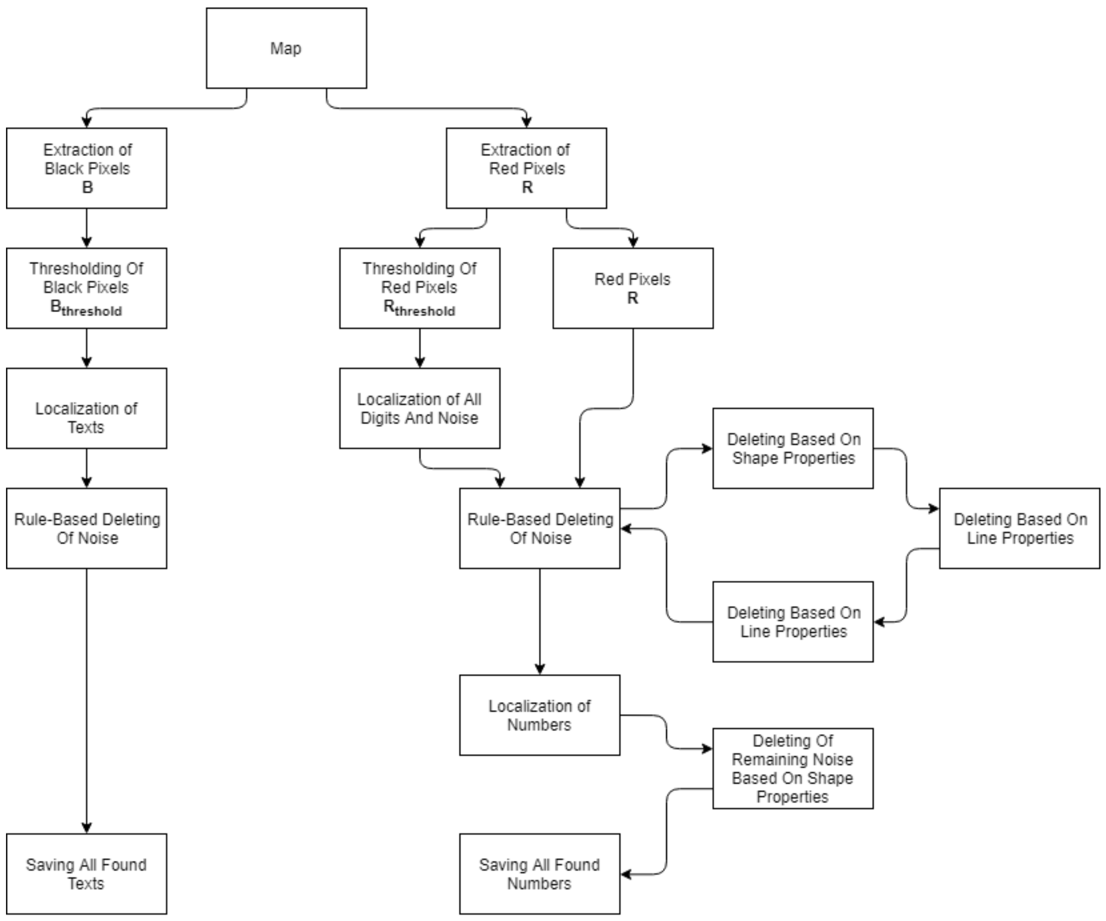

Map
Extraction of Black Pixels B
Extraction of Red Pixels R
Thresholding Of Black Pixels $B_{threshold}$
Thresholding Of Red Pixels $R_{threshold}$
Red Pixels R
Localization of Texts
Localization of All Digits And Noise
Rule-Based Deleting Of Noise
Rule-Based Deleting Of Noise
Deleting Based On Shape Properties
Deleting Based On Line Properties
Deleting Based On Line Properties
Localization of Numbers
Deleting Of Remaining Noise Based On Shape Properties
Saving All Found Texts
Saving All Found Numbers

Figure 4.4.: Methodology for Removing Noise and Localizing Numbers and Phonetic Transcriptions according to Rules

formed. Every position and bounding box property is stored for the mapping process. The next chapter presents the deep learning approach for the removing of noise and the localization of numbers and texts.

4.2. Deep Learning Approach

The recognition of texts and numbers can be considered as a binary segmentation problem. We try to differentiate texts and numbers from the background. In the case of numbers, noise shall be classified as background and thus cleaned up when just focusing on the foreground. Therefore the recognition of numbers and texts are two different problems. For each problem, we train a different neural network. Because we work with images we specifically use CNNs.
Our approach is grouped into three phases. First, we define the architecture of the neural network. The next step is to preprocess the images for the training. In the end, we

(a) Original Number (b) After 1 Dilation (c) After 2 Dilation (d) After 3 Dilation

Figure 4.5.: Example for the Dilation of the Number 118

describe how the trained model is used to remove noise and localize numbers and texts on a map (Figure 4.4).

4.2.1. Infrastructure

The Leibniz Supercomputing Centre (LRZ) provides us with two virtual servers for training and using our neural networks. The first virtual server contains 16 cores, 240 GB of memory, a NVIDIA Tesla P100 16 GB PCIe GPGPU and a 800 GB PCIe SSD. The second virtual server runs on the *LRZ Compute Cloud*, which enables us to use our own virtual machine image. The server is assigned to 20 virtual cores, 368 GB of memory, a NVIDIA Tesla V100 16 GB PCI GPU and a volume storage of 200 GB.

4.2.2. U-Net Architecture

In this work we use the *U-Net* developed by Ronneberger et al. [RFB15] for segmentation of biomedical images. It is based on the concept of a Fully Convolutional Network (FCN)[LSD15]. In contrast to normal CNNs, used for classification, the prediction is pixelwise and does not apply to the entire image, also each pixel is localized [LSD15]. Adding successive layers to a contracting network and replacing pooling operators with upsampling operators in these layers, is the main idea behind the FCN. Hence a FCN consists of a downsampling and an upsampling path, sometimes referred to as a contracting and expansive path. The resolution of the output is increased in the upsampling path. By combining high-resolution features from the contracting path with the upsampled output, the localization is performed. Based on the localization information a successive convolution layer can improve its output. Ronneberger et al. [RFB15] improved this approach by increasing the number of feature channels in the upsampling part, in order to forward context information to higher resolution layers. The expansive path almost contains the same amount of layers as the contracting path, hence the U-shape architecture. There are no fully-connected layers, which reduces the number of parameters, making it possible to train with a few samples. For their training, the authors used a dataset with 30 images and additional data augmentation. In addition, only the valid part of each convolution is considered, as the full context of the input image is used. This allows the segmentation of images of any size. The extrapolation of the missing context of the border areas during a prediction is done by mirroring the input image. As a result, the output image is cropped.
Figure 4.6 illustrates the network architecture. The left side represents the contracting path and the right side the expansive path. A downsampling step consists of two

convolutional layers, including a rectified linear unit (ReLU), and a directly appended max-pooling operation. In every step, the number of features is doubled. The number of downsampling steps is arbitrary but needs to match the number of upsampling steps. An upsampling step consists of an upsampling layer followed by an up-convolution, that concatenates the previous upsampling layer with the matching cropped feature map from the left side. As a result, the number of features is halved. Two convolutional layers including ReLU follow. The last layer is a 1 x 1 convolution, which maps the resulting feature vector to the number of classes [RFB15].

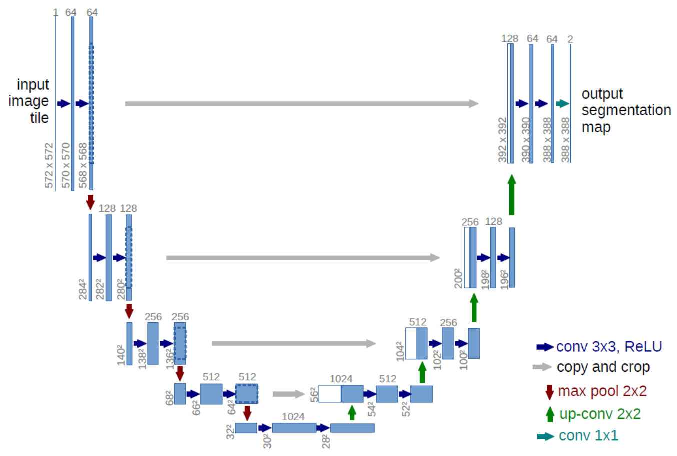
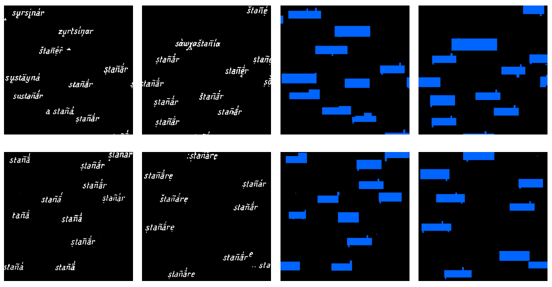


Figure 4.6.: U-Net Architecture by Ronneberger et al. [RFB15]

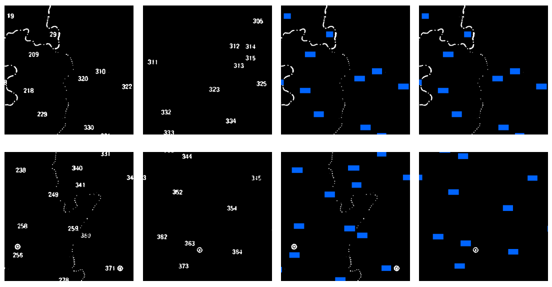
To use *U-Net* to segment texts and numbers, we adjust the number of down and upsampling steps. The original *U-Net* has 4 down and 4 upsampling steps, while in our case we reduce them to 3, and we also use the ’same’ padding for the convolutional layers, because we do not want the output to be cropped. Since the contours of the different characters and digits are too thin to be reproduced legibly so that they can be distinguished, the extrapolation of the lost information does not work. In addition, the localization information of the results found in the output needs to match the localization information in the input image in order to be stored and used. The last adaption affects the loss function, Ronneberger et al. [RFB15] used the cross-entropy function, while we use the Dice Loss, as we have a sparse area with content.

4.2.3. Data Preprocessing

Since the training of the U-Net is a supervised learning problem, we need to provide training samples, which consists of original images and labeled images. The localization of texts and numbers is independent of each other, hence two NNs are trained, one for numbers and one for texts. The labeling of positions numbers and phonetic transcriptions on each training map is done manually by hand with a colored marking. The maps are available in a size of 4821 x 6494 pixels and thus too large to be processed at once, so we decompose the images into patches of 512 x 512 pixels. Since the image size is not a multiple of 512, padding is added before the decomposing. The width is extended to the right by 199 black pixels and the height is extended to the bottom by 162 black pixels so that the new image size is 5120 x 6656 pixels and thus divisible by 512. One map is decomposed into 130 patches, hence 130 training samples for the number model and 130 training samples for the text model. Figure 4.7 shows some training samples including the ground truth and the labeled images. In order to improve the generalization ability and prevent overfitting, we use data augmentation. By applying different transformations to the training samples, the number of training samples increases while making the network more robust to invariance.



(a) Text Samples

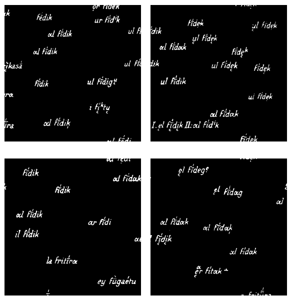


(b) Number Samples

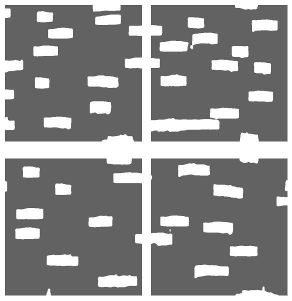
Figure 4.7.: Training Samples Of Texts and Numbers

4.2.4. Methodology

Figure 4.9 presents the methodology of the deep learning approach for the removing of noise and localization of numbers and texts. Just like the rule-based approach in Section 4.1, we also work with the binary images $\mathbf{B_{threshold}}$ and $\mathbf{R_{threshold}}$. Since our NNs are trained with input sizes of 130 x 130 pixels, we decompose the input images $\mathbf{B_{threshold}}$ and $\mathbf{R_{threshold}}$ into patches of 130 x 130 pixels so that they can be processed by the NNs. For each patch all texts respectively all numbers on it are labeled by the trained network.



(a) Input



(b) Output

Figure 4.8.: Example of Neural Network Segmentation Results for a few Inputs

Figure 4.8 shows the output of the NNs for some text input samples. After each patch is processed, all patches are merged to restore the original image size for further processing. Contrary to the rule-based approach, we do not need to consider corresponding digits or words, as numbers or phonetic transcriptions are recognized automatically by the NN. In the following we refer to the predicted image of texts as $\mathbf{B_{pred}}$ and the predicted image of numbers as $\mathbf{R_{pred}}$.
First we try to detect every labeled text in $\mathbf{B_{pred}}$ by finding their contours on the map. Based on the height and width of a contour, we differentiate between phonetic transcription and normal text like headers or descriptions on the map. Only the contour properties of phonetic transcription are saved for further processing. Regarding the noise cleaning in $\mathbf{R_{threshold}}$ no large set of rules is needed like in the rule-based approach. The noise detection is automatic proceed by the NN, every labeled contour in $\mathbf{R_{pred}}$ represents a number everything else is noise. However, based on the height and the width of a number contour, falsely labeled numbers are filtered out. Apart from that, every number contour is stored for the mapping process. The next section describes the mapping process of a phonetic transcription to a position number.

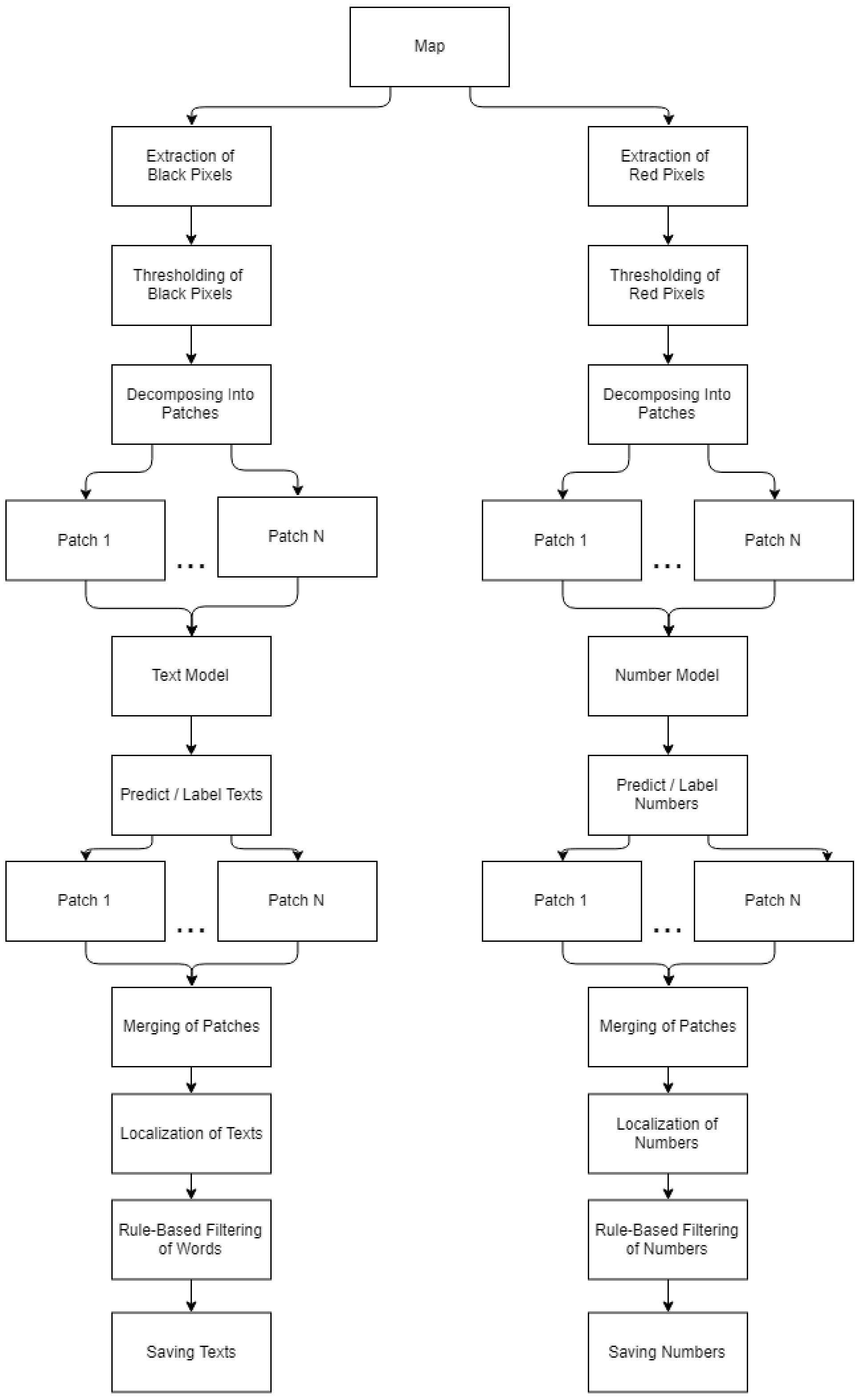


Figure 4.9.: Deep Learning Methodology for Noise Removal and Localization of Numbers and Phonetic Transcriptions

4.3. Mapping of Position Numbers and Phonetic Transcriptions

In the last step, the mapping of phonetic transcriptions to position numbers is performed and the results are saved (Figure 4.10). Since the phonetic transcription was handwritten by a human on the map, the training of a NN for mapping is not feasible because the Gestalt laws [Tod08] apply. Therefore we use the symbolic AI approach and define mapping rules, based on our visual perception.

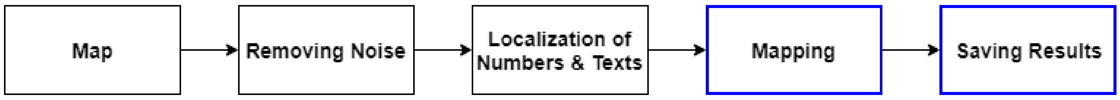


Figure 4.10.: Mapping Of Numbers And Texts And Saving Of Results

In the previous steps, we saved the positions and contours of every number and phonetic transcription. These serve as the basis for rule creation. We differentiate between successfully mapped pairs and unsuccessfully mapped pairs. The map is processed starting from the top left. For every phonetic transcription, we try to find the corresponding position number. Moreover, each number can only be mapped to one phonetic transcription. We split the mapping of phonetic transcriptions into three cases.
At first, we start with a special case, where phonetic transcriptions are depicted as crosses **X** on the map (Figure 4.11). The corresponding phonetic transcription can be found in the legend on the map.

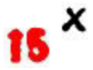


Figure 4.11.: Special Case: Phonetic Transcription is denoted as a Cross **X**

Using the height and width of a text contour, we determine whether it corresponds to a cross **X** case. Next, we look for the nearest position number with the centroid (x,y) of the contour as the origin. Thereafter the distance to the centroids (x,y) of all numbers is calculated. In addition, only the numbers to the left of the cross are considered, since in this special case only these can be the corresponding position number. The smallest distance represents the corresponding position number. As distance function, the euclidean distance is used.

$$d_{eucl.}(cent_{text}, cent_{number}) = \sqrt{(x_{text} - x_{number})^2 + (y_{text} - y_{number})^2}$$

We need to store an image of the mapped pair. Hence a bounding box is calculated, containing the cross **X** and the number. Figure 4.12 shows an example, where we can see the bounding box of a cross, the bounding box of the corresponding number and the new calculated bounding box. The thick purple line indicates the mapping of the two bounding boxes. For the top and the bottom of the new bounding box, padding is added. The new bounding box is saved as an image.

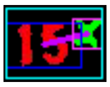


Figure 4.12.: Bounding Box containing a Cross and a Position Number

Next, we process another special case. Based on our mapping knowledge and the Gestalt laws, we know that position numbers very close to the left edge of the bounding box of a text contour correspond to it. Therefore, we calculate the distance to the next left number for each text contour. The distance calculation is the same as in the previous step. We define a threshold value for the maximum possible distance that a number can have from the left edge of the contour. If the threshold value is not exceeded, a connected bounding box containing number and text, with a threshold at top and bottom, is built (Figure 4.13) and saved as an image. Those whose nearest left number exceeds the threshold value are considered in the next case.

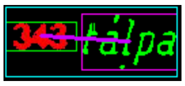


Figure 4.13.: Bounding Box containing Phonetic Transcription And Position Number

At last, we try to map all remaining text contours to their numbers. For this, we consider every character in a phonetic transcription. By finding all connected components in a text contour, we can determine each character and their corresponding bounding boxes. Based on the centroid of each bounding box, the nearest number is calculated with the euclidean distance and marked for each character. The most frequently marked number is used as the corresponding position number. However, if the distance of the selected number exceeds a certain threshold, the phonetic transcription is marked as unsuccessfully mapped and saved as an image containing only the transcription.
Figure 4.14 represents this mapping process, thin turquoise lines show the nearest number from each character, the thick purple line represents the final mapped position number. A bounding box including the phonetic transcription and the final position number is created and saved as an image.

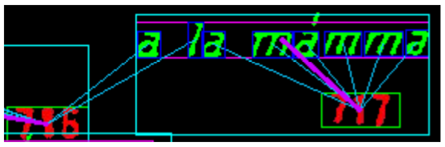


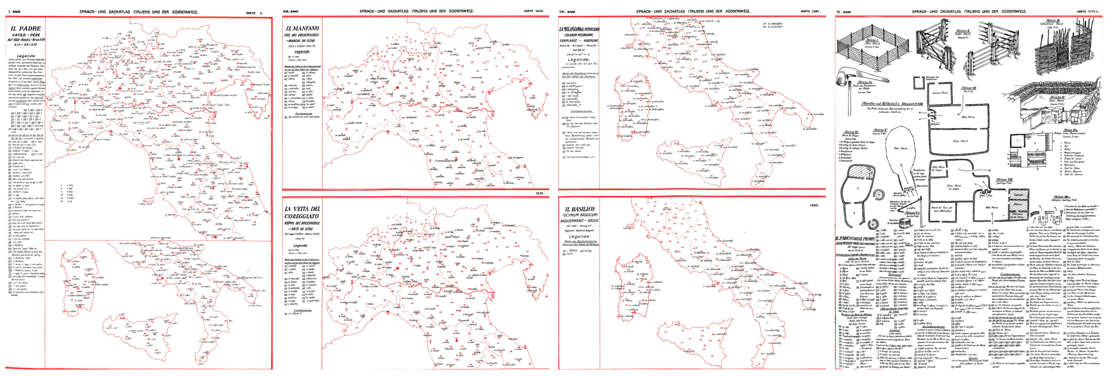
Figure 4.14.: Mapping of each Character to the nearest Position Number

All other text contours filtered in the previous processing steps are marked as noise and saved as images in a separate folder. At the end, there are two image folders, one with the successfully mapped pairs and one with all unknown contours and unsuccessfully mapped phonetic transcriptions.

5. Evaluation

5.1. Data

The IT-Group Humanities of the LMU provides us with historical phonetic maps extracted from the *AIS* [JJS28]. We classify them into 4 categories and differentiate between *complete*, *upper*, *lower* and *other* maps (Figure 5.1). A map is considered as *complete* if only one term is evaluated and the entire geographical map of southern Switzerland and Italy is visible. As for *upper* maps, the map of southern Switzerland and northern Italy is visible twice. Each map represents a different term. Similarly, for *lower* maps there a two maps of southern Italy, each representing a different term. Maps assigned to *other* show no historical phonetic map at all, but additional information such as conjugation tables or illustrations with names.



(a) Complete (b) Upper (c) Lower (d) Other

Figure 5.1.: Types of historical phonetic maps

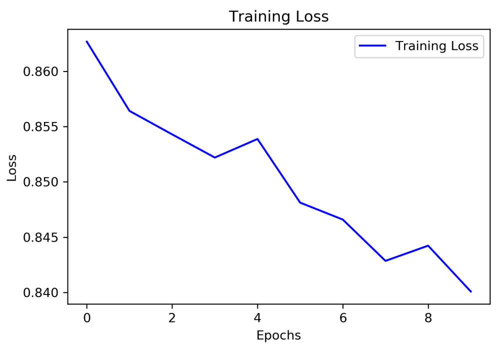
There are a total of 1721 maps, from which 1601 and thus most are classified as *complete*. Only about 7% are assigned to other categories. There are 36 *upper* maps, 39 *lower* maps and 45 *other* maps (Table 5.1). Therefore the focus of this work and the evaluation is on the *complete* maps.

Total	Complete	Upper	Lower	Other
1721	1601	36	39	45

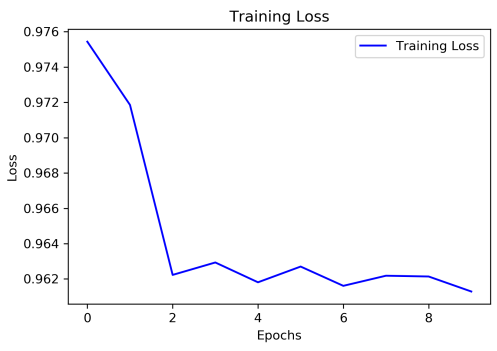
Table 5.1.: Number of all provided historical phonetic maps

5.2. Model Training

In this section, we discuss the training of our employed neural networks. As basis for our network we adapted an implementation of *U-Net* with *Keras* provided by `https://github.com/zhixuhao/unet`. Figure 5.3 illustrates the adopted model with an input of (512,512,3).
We trained two different models with the same network architecture (Figure 5.3), one for text segmentation and one for number segmentation. The segmentation is a binary segmentation, all found texts and numbers should be assigned to the foreground, everything else to the background. Foreground objects are displayed in white whereas background objects are displayed in black. To train the text model, we labeled the phonetic transcriptions in three historical phonetic maps that are used as training samples, as described in Chapter 4.2.3. Each map is decomposed into 130 patches of size 512 x 512. In total, there are 390 training samples for the text model. For a more robust segmentation, we use data augmentation to generate more variety of training samples. The training samples are randomly rotated, shifted, flipped, zoomed and sheared.



(a) Text Model's Training Loss



(b) Number Model's Training Loss

Figure 5.2.: Overview of the Training Loss for Text and Number Model

We trained the model for 10 epochs with the *Adam* Optimizer [KB14] and used a learning rate of 1e-4, the *dice* loss and a batch size of 1.
Figure 5.2a shows the progress of the training loss for the text model. As we can see, the loss per epoch decreases continuously. The final loss is 0.841, which is apparently high, but many further training attempts showed that the smaller the loss value, the more pixels are assigned to the background. With a higher number of epochs, the training loss converges towards 0. The final result is a black image, where all pixels are segmented towards the background. Because most pixels in a map correspond to the background and the map is therefore sparse in content, especially when texts or numbers are filtered. This complicates model optimization since the loss does not cover the true segmentation performance. To see how well a trained model performs, we need to segment a test image and visually look at how good the segmentation is. Hence the optimization of our trained model is also based on our subjective opinion and not only on the loss value.

Regarding the training samples of the number model, we only need to label one historical phonetic map, since the position of the numbers on each map is the same. The map is again decomposed into patches of size 512 x 512, representing the training samples. We simply increased the number of training samples by reusing the already labeled map. In total, we trained our number model with 1040 training samples. The training and augmenting parameters remained the same as in the text model training. Figure 5.2b illustrates the progress of the training loss for the number model. After 2 epochs the loss stagnates and decreases minimally. This is due to the sparsity in the training samples, since the amount of pixels corresponding to a number is much smaller than the number of background pixels. It is also reflected in the final training loss value, which is very high at 0.961. Compared to the training loss of the text model, the text loss is 0.1 lower, because the number of text pixels is much higher and therefore the training samples are denser. We have the same optimization problem as before, however in this case the training samples are sparser, resulting in a faster convergence towards 0 and thus to a more difficult optimization. In terms of training time, each model is trained in less than one hour, as we only trained 10 epochs with a batch size of 1, which makes the model training very fast.

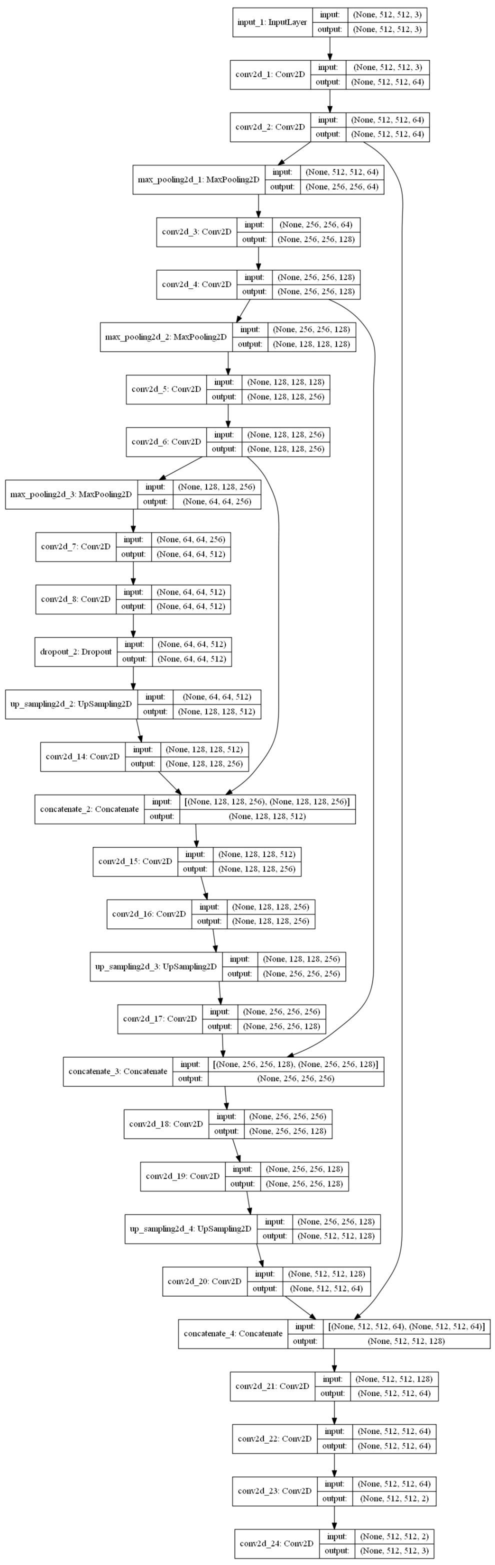

input_1: InputLayer
input: (None, 512, 512, 3)
output: (None, 512, 512, 3)
conv2d_1: Conv2D
input: (None, 512, 512, 3)
output: (None, 512, 512, 64)
conv2d_2: Conv2D
input: (None, 512, 512, 64)
output: (None, 512, 512, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 512, 512, 64)
output: (None, 256, 256, 64)
conv2d_3: Conv2D
input: (None, 256, 256, 64)
output: (None, 256, 256, 128)
conv2d_4: Conv2D
input: (None, 256, 256, 128)
output: (None, 256, 256, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 256, 256, 128)
output: (None, 128, 128, 128)
conv2d_5: Conv2D
input: (None, 128, 128, 128)
output: (None, 128, 128, 256)
conv2d_6: Conv2D
input: (None, 128, 128, 256)
output: (None, 128, 128, 256)
max_pooling2d_3: MaxPooling2D
input: (None, 128, 128, 256)
output: (None, 64, 64, 256)
conv2d_7: Conv2D
input: (None, 64, 64, 256)
output: (None, 64, 64, 512)
conv2d_8: Conv2D
input: (None, 64, 64, 512)
output: (None, 64, 64, 512)
dropout_2: Dropout
input: (None, 64, 64, 512)
output: (None, 64, 64, 512)
up_sampling2d_2: UpSampling2D
input: (None, 64, 64, 512)
output: (None, 128, 128, 512)
conv2d_14: Conv2D
input: (None, 128, 128, 512)
output: (None, 128, 128, 256)
concatenate_2: Concatenate
input: [(None, 128, 128, 256), (None, 128, 128, 256)]
output: (None, 128, 128, 512)
conv2d_15: Conv2D
input: (None, 128, 128, 512)
output: (None, 128, 128, 256)
conv2d_16: Conv2D
input: (None, 128, 128, 256)
output: (None, 128, 128, 256)
up_sampling2d_3: UpSampling2D
input: (None, 128, 128, 256)
output: (None, 256, 256, 256)
conv2d_17: Conv2D
input: (None, 256, 256, 256)
output: (None, 256, 256, 128)
concatenate_3: Concatenate
input: [(None, 256, 256, 128), (None, 256, 256, 128)]
output: (None, 256, 256, 256)
conv2d_18: Conv2D
input: (None, 256, 256, 256)
output: (None, 256, 256, 128)
conv2d_19: Conv2D
input: (None, 256, 256, 128)
output: (None, 256, 256, 128)
up_sampling2d_4: UpSampling2D
input: (None, 256, 256, 128)
output: (None, 512, 512, 128)
conv2d_20: Conv2D
input: (None, 512, 512, 128)
output: (None, 512, 512, 64)
concatenate_4: Concatenate
input: [(None, 512, 512, 64), (None, 512, 512, 64)]
output: (None, 512, 512, 128)
conv2d_21: Conv2D
input: (None, 512, 512, 128)
output: (None, 512, 512, 64)
conv2d_22: Conv2D
input: (None, 512, 512, 64)
output: (None, 512, 512, 64)
conv2d_23: Conv2D
input: (None, 512, 512, 64)
output: (None, 512, 512, 2)
conv2d_24: Conv2D
input: (None, 512, 512, 2)
output: (None, 512, 512, 3)

Figure 5.3.: Architecture of our employed Model

5.3. Mapping Results

Next, we evaluate the mapping results of our proposed approaches. An automated evaluation system is not feasible since the Gestalt laws [Tod08] apply, which is why we cannot use an appropriate automatic evaluation measure besides looking at each mapping one by one. For our tests, we randomly select 10 *complete* historical phonetic maps from 1601. We consider the evaluation of our results as a binary classification problem, a mapping can be either correct or incorrect. For the evaluation we use three statistical measures[Pow11]: *precision*, *recall* and *F1-measure*.

Precision

The precision indicates how many of the found mappings actually are correctly mapped.

$$\text{Precision} = \frac{\#\ \text{Correctly Found Mappings}}{\#\ \text{Found Mappings}}$$

Recall

The recall, also called sensitivity, specifies the proportion of the correct found mappings.

$$\text{Recall} = \frac{\#\ \text{Correctly Found Mappings}}{\#\ \text{Total Mappings}}$$

F1-Measure

The F1-measure is the harmonic mean of precision and recall and therefore summarizes both values.

$$\text{F1-Measure} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

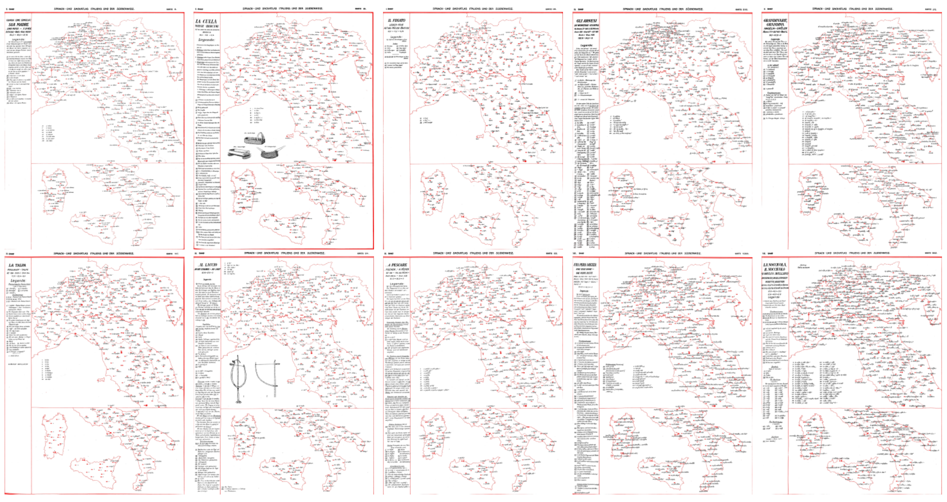


Figure 5.4.: Overview of all tested *complete* Maps

Figure 5.4 shows all *complete* maps we used for our test. There are a total of 407 position numbers on each map, each number can correspond to a phonetic transcription but does not have to, because there are numbers without phonetic transcription. We consider a mapping correct if the correct number is assigned to the correct phonetic transcription. Every map differs in the number of mappings. Figure 5.5 illustrates the mapping results of our approaches as a bar graph, with two bars for each map representing the results of our two different approaches. The rule-based approach is represented by bars with an orange border, while the deep learning bars have a blue border. A bar is divided into three sections. The first section shows the number of correctly found mappings and is displayed in green, the second section represents the number of incorrectly found mappings and is displayed in red. The last section in gray shows the number of not found mappings. The height of a bar represents the total number of mappings on a map. Overall the DL approach finds a higher total number of correctly found mappings. Furthermore, the number of incorrectly found mappings is significantly lower. To analyze the test results in detail, we look at Table 5.2.

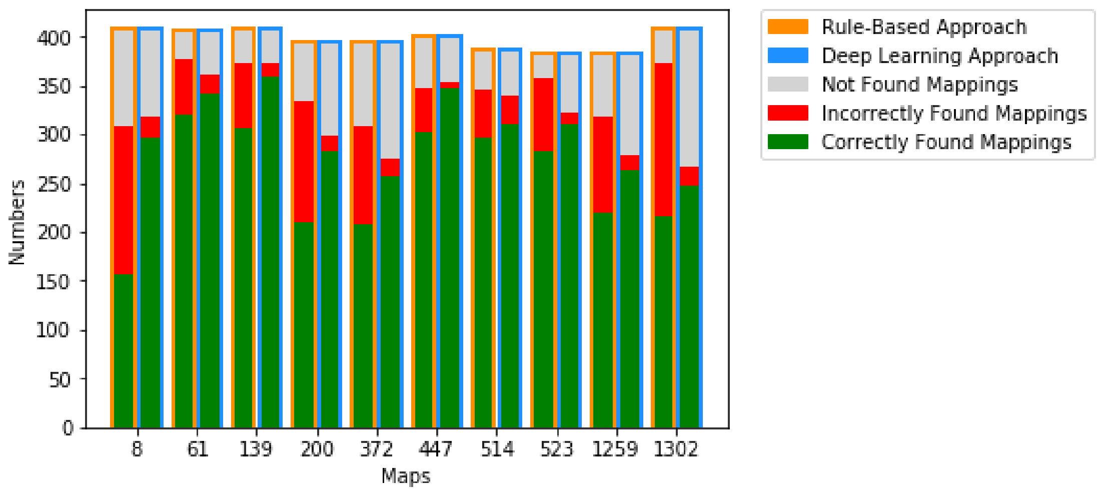


Figure 5.5.: Visualization of mapping results for each test Map

For each map, the map type, the approach type, the number of correctly and incorrectly found mappings and the total number of mappings are displayed. We distinguish between two map types: sparse and dense. A map is considered as sparse when most phonetic transcriptions are short in terms of text length and thus the distance between each phonetic transcriptions is greater. Otherwise, the map is considered dense because of the long phonetic transcriptions that make the map denser in the number of pixels. Moreover, we also can see the values for precision, recall and F1 for better comparability.

In 14 of 20 tests a precision value of over 0.9 can be achieved, of which 10 tests belong to the DL approach. Hence, the DL approach achieves very high precision in all tests, which means that most of the predicted mappings are correct. Considering the recall of the DL results, 7 out of 10 tests have a value above 0.7 compared to the RB approach

Map	Type	Method	# Found Mappings	# Incorrect	# Correct	Precision	# Total Mappings	Recall	F1
8	Dense	RB	309	152	157	0.508	406	0.387	0.439
8	Dense	DL	318	22	296	0.931	406	0.729	0.818
61	Sparse	RB	377	57	320	0.849	405	0.790	0.818
61	Sparse	DL	361	20	341	0.945	405	0.842	0.890
139	Sparse	RB	373	67	306	0.820	407	0.752	0.785
139	Sparse	DL	374	15	359	0.960	407	0.882	0.919
200	Dense	RB	333	123	210	0.631	393	0.534	0.579
200	Dense	DL	299	16	283	0.946	393	0.720	0.818
372	Dense	RB	308	101	207	0.672	393	0.527	0.591
372	Dense	DL	274	17	257	0.938	393	0.654	0.771
447	Sparse	RB	347	45	302	0.870	398	0.759	0.811
447	Sparse	DL	353	5	348	0.986	398	0.874	0.927
514	Sparse	RB	346	50	296	0.855	385	0.769	0.810
514	Sparse	DL	339	28	311	0.917	385	0.808	0.859
523	Sparse	RB	357	75	282	0.790	381	0.740	0.764
523	Sparse	DL	321	10	311	0.969	381	0.816	0.886
1259	Dense	RB	318	98	220	0.692	381	0.577	0.629
1259	Dense	DL	278	15	263	0.946	381	0.690	0.798
1302	Dense	RB	373	158	215	0.576	406	0.530	0.552
1302	Dense	DL	266	19	247	0.929	406	0.608	0.735

Table 5.2.: Test Results: Map Name, Type of Map, Approach Type (Rule-Based or Depp Learning), Number of Found Mappings, Number of Incorrectly Found Mappings, Number of Correctly Found Mapping, Precision, Number of All Mappings, Recall, F1-Measure

where only 5 tests have a value above 0.7. This is also reflected in the F1 values, the DL approach can reach an average of 0.84, while the RB approach only has an average of 0.68. Overall, the DL approach achieves a better result for each map, both in terms of precision and recall. In Table 5.3 we can see that the average precision of the DL approach is nearly 0.95 and the average recall is 0.76, whereas the RB approach has an average precision of 0.73 and a recall of 0.64. Thus, the DL approach can find an average of 76% of all mappings on a map and therefore 12% more with an average 22% higher precision than the RB approach.

Approach Type	Avg. Precision	Avg. Recall	Avg. F1	Avg. Running Time (CPU)	Avg. Running Time (GPU)
RB	0.726	0.636	0.678	120 s	-
DL	0.947	0.762	0.842	-	68 s

Table 5.3.: Test Summary: Average Mapping Results and Inference Time for the Rule-Based (RB) and the Deep Learning Approach (DL)

We executed the RB approach on a CPU with 2 cores, while the DL approach is executed on GPU. Due to the increased computing capacity of the GPU, the inference time is reduced. The inference time is the time required to forward an input through a neural network and generate the corresponding output. In our case to segment the numbers and phonetic transcriptions. The RB approach can't be run on GPU since it is only implemented to run on CPU. The average running time for the RB approach is

120s. For a direct comparison, the DL approach was also executed with CPU, however, the run time for the first map was 2664s, therefore we did not run the remaining maps with CPU, because the running time is many times longer than on GPU. The much longer execution time results from the increased inference time, which results from the low computing capacity. We employed two neural networks, one for the segmentation of numbers and one for phonetic transcriptions. The input image is divided into a number and a text image. Each is decomposed into 130 patches so that a total of 260 inputs have to be processed by our models sequentially, which causes the high computational effort. On GPU the average running time for the DL approach is only 68s and is thus, as expected, faster than the average running time of the RB approach on CPU.
To see the advantages of the DL approach over the RB, we consider the results of the different map types. For both sparse and dense maps the average precision is above 0.93, while the RB approach can only achieve an average precision of 0.84 for sparse and 0.62 for dense maps. Considering the average recall, the RB approach has a value of 0.76 for sparse and only value of 0.51 for dense maps. In contrast, the DL approach has an average recall of 0.84 for sparse maps, which is 0.08 higher, and 0.68 recall for dense maps, which is even 0.17 higher. The F1-values summarizes precision and recall, the DL approach has an average F1-value of 0.90 for sparse maps, the RB approach has only 0.80. For dense maps, the DL approach can show a value of 0.71, while the RB only has 0.56. Therefore the DL approach achieves better results for both maps, especially for dense maps.

Approach Type	Map Type	Avg. Precision	Avg. Recall	F1
RB	Sparse	0.837	0.762	0.798
DL	Sparse	0.955	0.844	0.896
RB	Dense	0.616	0.511	0.558
DL	Dense	0.938	0.680	0.713

Table 5.4.: Test Results for each Map Type: Dense and Sparse

The reasons for the better performance are the better localization of phonetic transcriptions and position numbers as well as the better recognition of related characters and numbers. Furthermore, noise cleaning is included in the number segmentation and not according to a large set of rules that can contain a human error. The RB approach causes the loss of digits in a number or the deletion of a number since it is mistakenly considered as noise. The DL approach removes almost all noises on the map and does not delete any number. Hence, there is a better basis for the mapping, since noise is not considered as a number and every number is localized. In dense maps, a good localization of any phonetic transcription is important, otherwise closely related texts are recognized as one related text. Since we use dilation in the RB approach, more phonetic transcriptions are combined into one, resulting in a higher number of incorrectly found mappings. In general, the RB approach finds more mappings than the DL approach, but due to the listed drawbacks, the average percentage of correctly found pairs is much lower. The DL approach has a very high precision of correct mappings, but still cannot find all mappings, due to the merging of multiple phonetic transcriptions.

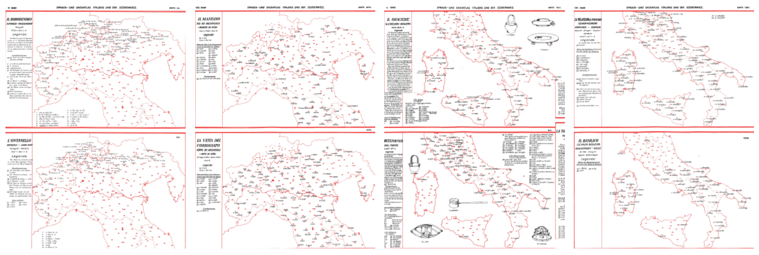


Figure 5.6.: Overview of all tested *upper* and *lower* Maps

Next, we look at *upper* and *lower* maps to evaluate the generalization of our methods for deviating maps, especially as our network is trained on training samples of *complete* maps. Figure 5.6 shows the maps we used for our tests. As we can see, many position numbers do not have a corresponding phonetic transcription, which reduces the number of mappings. Table 5.5 presents the test results for every map in detail. We see again that the DL approach finds nearly only correctly mappings, for every map, a precision value of over 0.9 can be achieved. The average precision value for the RB approach is only 0.74 (Table 5.6). Since the number of mappings is not as high as in a *complete* map, the DL approach has a high average recall of 0.86 compared to 0.71 for the RB approach. Overall, we can say that for both map types: *lower* and *upper*, the performance of the DL approach is better than the RB approach.

Map	Type	Method	# Found Mappings	# Incorrect	# Correct	Precision	# Total Mappings	Recall	F1
				Upper Maps					
582	Dense	RB	191	57	134	0.702	218	0.615	0.655
582	Dense	DL	182	16	166	0.912	218	0.761	0.830
1475	Sparse	RB	222	52	170	0.766	205	0.829	0.796
1475	Sparse	DL	201	4	197	0.980	205	0.961	0.970
				Lower Maps					
940	Sparse	RB	156	23	133	0.853	163	0.816	0.834
940	Sparse	DL	147	9	138	0.939	163	0.847	0.890
1382	Dense	RB	173	61	112	0.647	191	0.586	0.615
1382	Dense	DL	179	13	166	0.927	191	0.869	0.897

Table 5.5.: Lower and Upper Test Results: Map Name, Type of Map, Approach Type (Rule-Based or Depp Learning), Number of Found Mappings, Number of Incorrectly Found Mappings, Number of Correctly Found Mapping, Precision, Number of All Mappings, Recall, F1-Measure

Approach Type	Avg. Precision	Avg. Recall	Avg. F1
	Upper Maps		
RB	0.734	0.722	0.726
DL	0.946	0.861	0.900
	Lower Maps		
RB	0.750	0.701	0.725
DL	0.933	0.858	0.894

Table 5.6.: Lower and Upper Test Summary: Average Mapping and Runtime Results for the Rule-Based (RB) and the Deep Learning Approach (DL)

6. Conclusion

This work presented a methodology for the mapping of phonetic transcriptions to their corresponding position number on a historical phonetic map. The methodology is split into two phases. The first phase completes the preprocessing of a map, including the removing of noise and the localization of numbers and texts. For this, we proposed two different approaches.
The first approach is based on the concept of symbolic AI. We created a large set of explicit rules, using the analysis of number and text properties. With this set, we can differentiate between noise and valuable information and therefore remove it. Furthermore, numbers and phonetic transcriptions are marked and localized with classic CV methods.
The second approach is based on the concept of image segmentation with deep learning. Text and number localization is seen as a binary segmentation problem, if it is recognized as one of them it is assigned to the foreground and therefore marked white otherwise it corresponds to the background, which is marked black. We trained two models, one for segmenting phonetic transcriptions and one for segmenting numbers. The segmentation automatically removes the noise as it is neither recognized as a number nor as text and hence as background. As foundation for our neural networks, *U-Net* [RFB15] is used. We use classical CV methods to localize the marked numbers and texts on the segmented images.
Based on the localized objects the mapping is performed. Since the information on the map was recorded by a person, it does not follow any logic that can be captured with the help of automated algorithms, hence the Gestalt laws apply. Therefore, the training of a neural network to predict corresponding numbers and phonetic transcriptions is not feasible. We define three mapping rules derived from our subjective visual understanding. The first one considers abbreviations on a map, where a phonetic transcription is depicted as a cross. The second one takes the position of the phonetic transcription into account, most of them are written left to their corresponding numbers. The last rule finds the nearest position number for every character in a phonetic transcription and selects the number that occurs most frequently as the position number. Every mapped pair is stored as an image.
We evaluate both approaches by analyzing the predicted mappings in 10 randomly selected maps. To evaluate, we use precision, recall and F1-score. When considering the test results, we see that all maps differ in their density of information. There are dense maps as well as sparse maps. Depending on this, the performance varies. For sparse maps, the mapping results are better than for dense maps, since the phonetic transcriptions are more clearly separated from each other and thus the localization is better. While for dense maps phonetic transcriptions are often not recognized as individuals

but as one big phonetic transcriptions. This leads to worse recall values, as not every transcription is recognized and therefore can not be mapped to a number.
Overall the deep learning (DL) approach achieves better mapping results in every map than the rule-based (RB) approach. The average F1-score of the DL approach is 0.86 whereas the RB approach only has an average score of 0.69. Furthermore, the average precision is very high with 0.947, so almost all found pairs are mapped correctly. The RB approach achieves an average precision of 0.726. When looking at the recall values, we can see that not all mappings are found. The DL approach has an average recall score of 0.762, while the RB approach has 0.636.
With this thesis and the achieved scores, we show the successful application of deep learning techniques for a problem in the area of humanities. An automated process has been developed to support the digitization of language atlases in the *VerbaAlpina* project.

6.1. Outlook

Our proposed methodology can be used as a first approach for the mapping of phonetic transcriptions to their corresponding position number on a historical phonetic map. There are still many possible refinements to our approach and future work since the evaluation showed that not every mapping on a map is found correctly. In this section, we present some improvement strategies.

Recall Improvement On average 25% of all mappings are not found by our deep learning approach. To find all mappings, the localization of the phonetic transcriptions has to be enhanced. There are different approaches to this. Due to the time constraints of this work, we were unable to find the optimal hyperparameters for our employed models. The better the hyperparameters are selected, the better the model can segment the different phonetic transcriptions. There exists a wide variety of hyperparameter tuning methods, for example *Grid Search* or *Random Search* [BB12]. Both depend on a set of possible configurations for the hyperparameters to be optimized. This includes the learning rate, batch-size or the number of hidden units, to name just a few. *Grid Search* considers any possible combination in this set and is, therefore, an exhaustive search, whereas *Random Search* randomly selects and tests combinations.
Another approach is to increase the amount of training data. With more data, the model could be better trained for the segmentation task. Besides, the input size could be of importance since we decompose the original input into patches of a specific size, related texts can be separated and thus incorrectly localized.

Automated Detection of False Positive Mappings The mappings are currently still evaluated manually. Automated evaluation is challenging since the Gestalt laws apply. An introduction of a confidence score for the mapping could support the detection of false-positive mappings. The confidence score can depend on the selected mapping rule, the distance to the position number and also the neighborhood of already found mappings. Based on a threshold value, the mappings could be classified into "confidentially mapped" and "require checks". That would reduce the number of possible false positives that need to be checked manually.

Improvement and Extension of the Mapping Rules Since the *Gestalt laws* apply to the mapping of position numbers and phonetic transcriptions, we define rules based on our visual knowledge to find these mappings. The rules are not error-free and can be improved, extended or even replaced with further research to achieve a higher matching quality.

A. Appendix

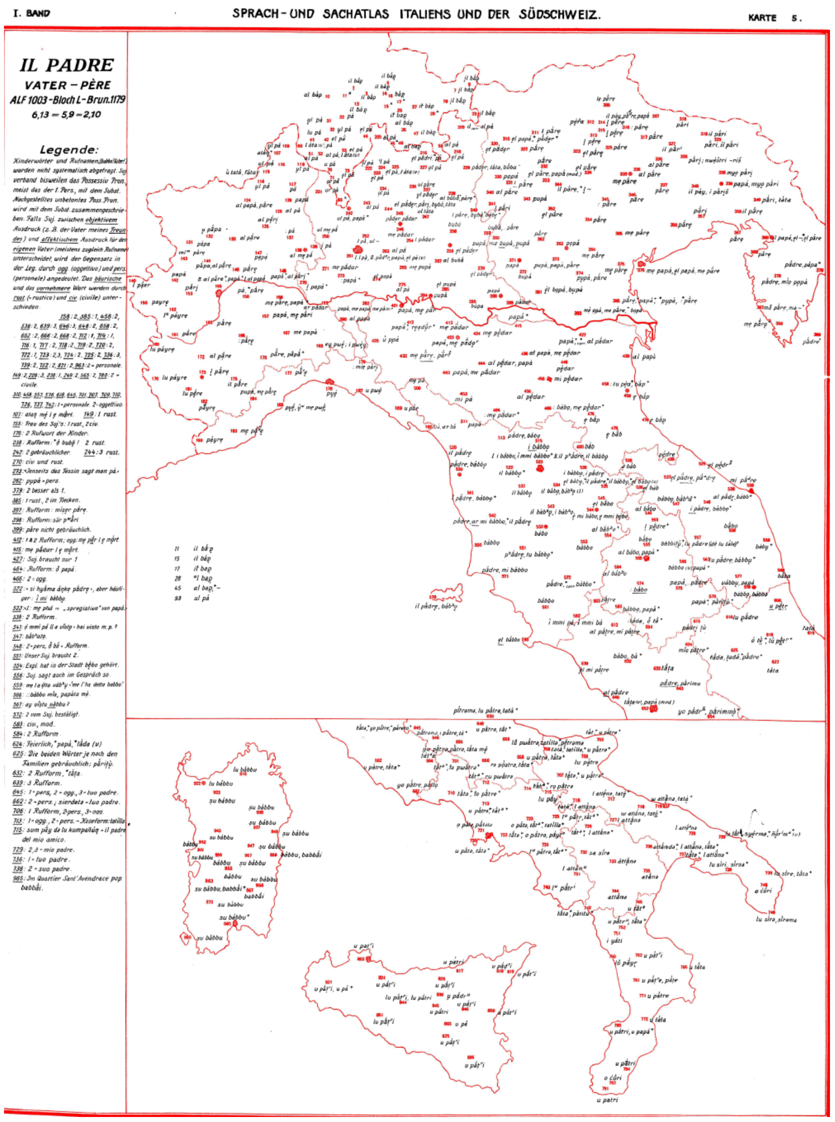

I. BAND
SPRACH- UND SACHATLAS ITALIENS UND DER SÜDSCHWEIZ.
KARTE 5.
IL PADRE
VATER – PÈRE
ALF 1003 - Bloch L - Brun. 1179
6,13 – 5,9 – 2,10
Legende:

Figure A.1.: Historical phonetic map

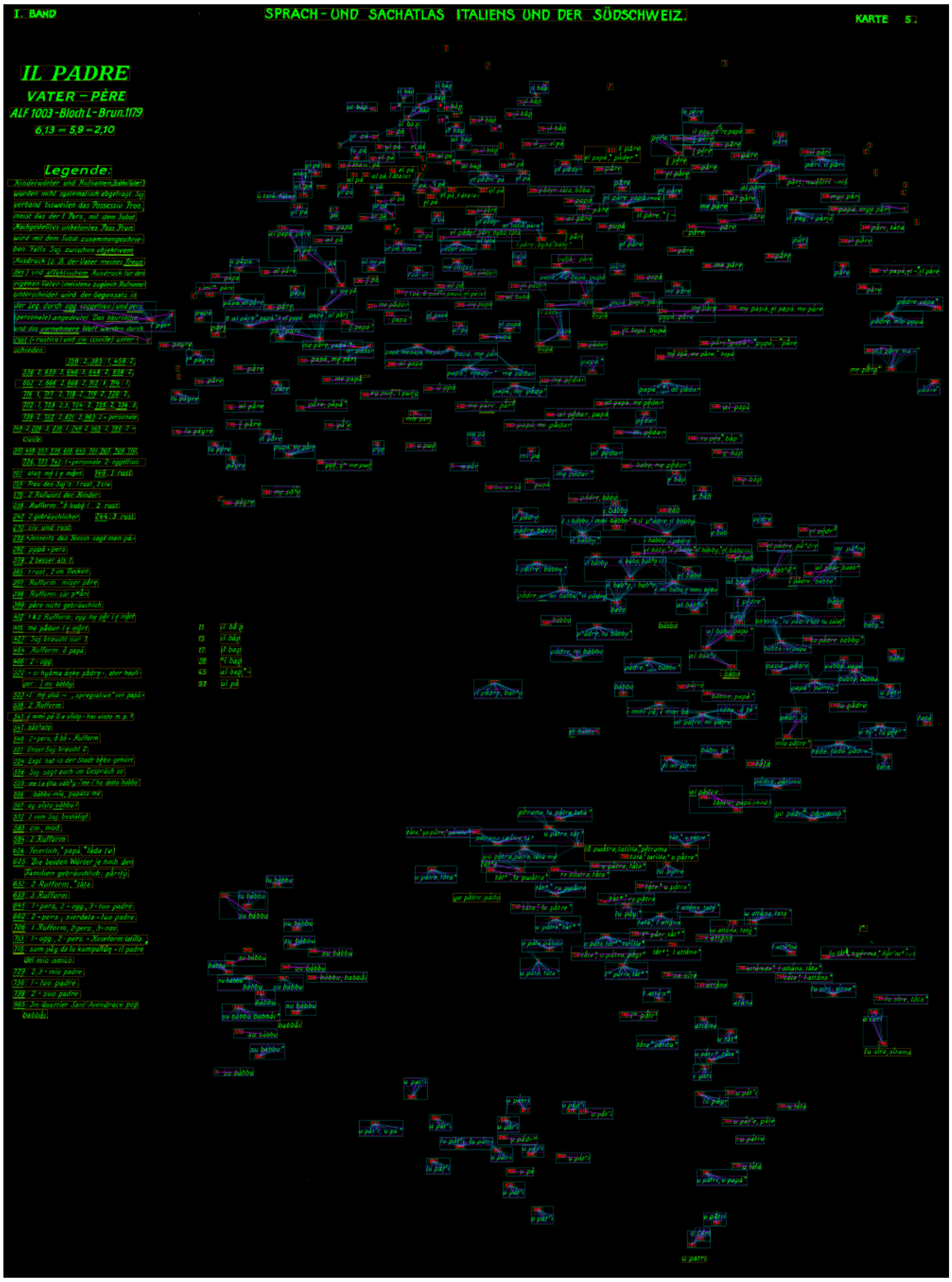

I. BAND
SPRACH- UND SACHATLAS ITALIENS UND DER SÜDSCHWEIZ.
KARTE 5.
IL PADRE
VATER – PÈRE
ALF 1003 - Bloch L - Brun.1179
6,13 = 5,9 – 2,10
Legende:

Figure A.2.: Example of the Mapping Process of the Rule-Based Approach

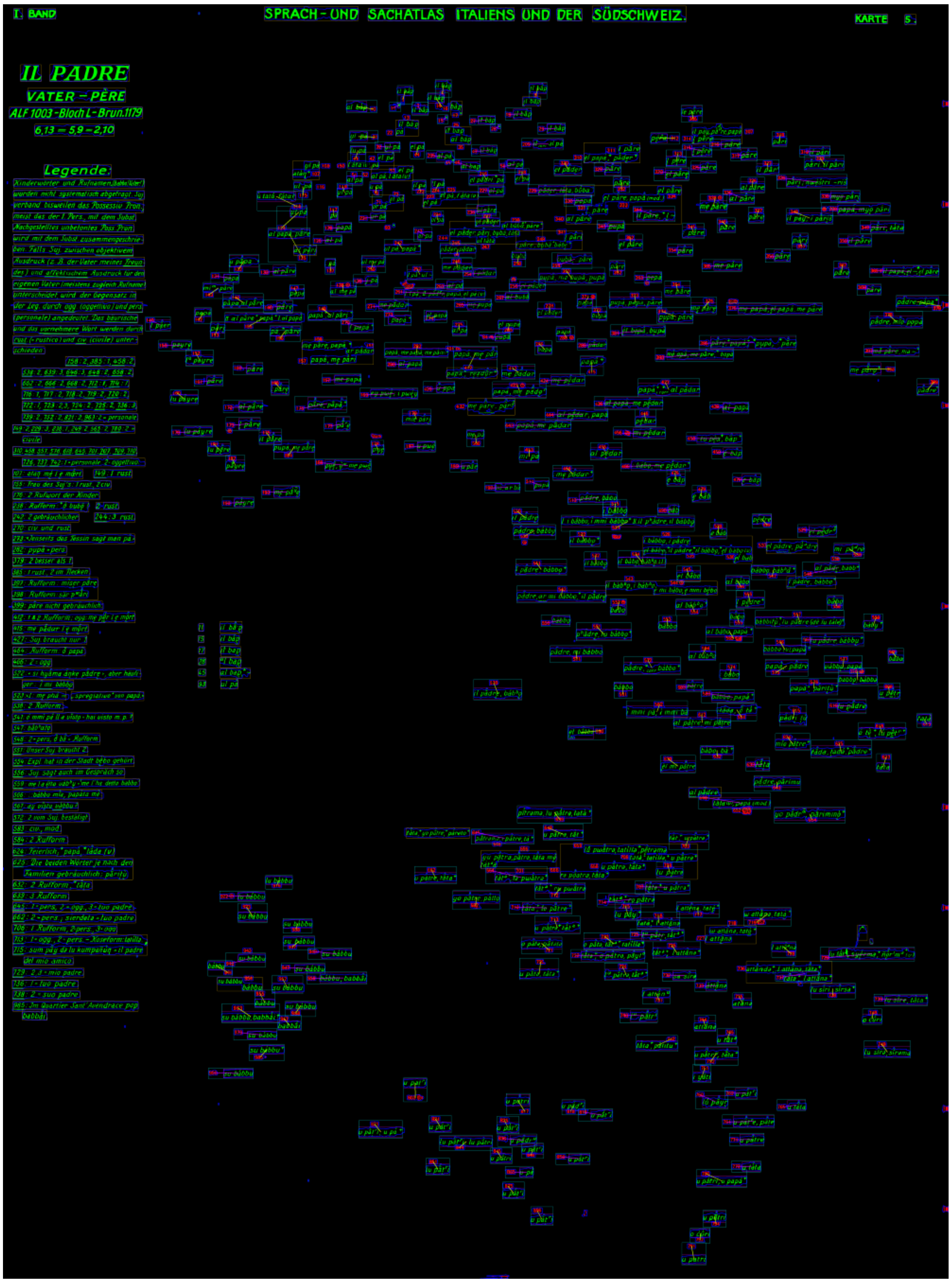

I. BAND
SPRACH- UND SACHATLAS ITALIENS UND DER SÜDSCHWEIZ.
KARTE 5.
IL PADRE
VATER – PÈRE
ALF 1003 - Bloch L - Brun. 1179
6,13 = 5,9 - 2,10
Legende:

Figure A.3.: Example of the Mapping Process of the Deep Learning Approach

List of Figures

Bibliography

[BB12] Bergstra, James ; Bengio, Yoshua: Random search for hyper-parameter optimization. In: *Journal of Machine Learning Research* 13 (2012), Nr. Feb, S. 281–305

[BK08] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008

[BKC17] Badrinarayanan, Vijay ; Kendall, Alex ; Cipolla, Roberto: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. In: *IEEE transactions on pattern analysis and machine intelligence* 39 (2017), Nr. 12, S. 2481–2495

[Can87] Canny, John: A computational approach to edge detection. In: *Readings in computer vision.* Elsevier, 1987, S. 184–203

[Cho18] Chollet, Francois: Deep learning with Python. (2018)

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep learning.* MIT press, 2016

[GDDM14] Girshick, Ross ; Donahue, Jeff ; Darrell, Trevor ; Malik, Jitendra: Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, S. 580–587

[GGOEO+17] Garcia-Garcia, Alberto ; Orts-Escolano, Sergio ; Oprea, Sergiu ; Villena-Martinez, Victor ; Garcia-Rodriguez, Jose: A review on deep learning techniques applied to semantic segmentation. In: *arXiv preprint arXiv:1704.06857* (2017)

[Gir15] Girshick, Ross: Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*, 2015, S. 1440–1448

[HGDG17] He, Kaiming ; Gkioxari, Georgia ; Dollár, Piotr ; Girshick, Ross: Mask r-cnn. In: *Proceedings of the IEEE international conference on computer vision*, 2017, S. 2961–2969

[Hou62] Hough, Paul V.: *Method and means for recognizing complex patterns.* Dezember 18 1962. – US Patent 3,069,654

[HSZ87] Haralick, Robert M. ; Sternberg, Stanley R. ; Zhuang, Xinhua: Image analysis using mathematical morphology. In: *IEEE transactions on pattern analysis and machine intelligence* (1987), Nr. 4, S. 532–550

[JJS28] Jaberg, Karl ; Jud, Jakob ; Scheuermeier, Paul: *Sprach-und Sachatlas Italiens und der Südschweiz.* Bd. 1. Ringier, 1928

[KB14] Kingma, Diederik P. ; Ba, Jimmy: Adam: A method for stochastic optimization. In: *arXiv preprint arXiv:1412.6980* (2014)

[KL14] Krefeld, Thomas ; Lücke, Stephan: *VerbaAlpina. Der alpine Kulturraum im Spiegel seiner Mehrsprachigkeit.* `http://dx.doi.org/10.5282/verba-alpina`, 2014. – Accessed: 29.06.2019

[KL18] Krefeld, Thomas ; Lücke, Stephan: *Verba Alpina (Projektpräsentation), First GeRDI Community Workshop.* `https://www.verba-alpina.gwi.uni-muenchen.de/wp-content/uploads/gerdi_presentation_community_workshop.pptx`, 2018. – Accessed: 25.09.2019

[KSH12] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*, 2012, S. 1097–1105

[Lag14] Laganière, Robert: *OpenCV Computer Vision Application Programming Cookbook Second Edition.* Packt Publishing Ltd, 2014

[LBB+98] LeCun, Yann ; Bottou, Léon ; Bengio, Yoshua ; Haffner, Patrick u. a.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[LBH15] LeCun, Yann ; Bengio, Yoshua ; Hinton, Geoffrey: Deep learning. In: *nature* 521 (2015), Nr. 7553, S. 436

[LRB15] Liu, Wei ; Rabinovich, Andrew ; Berg, Alexander C.: Parsenet: Looking wider to see better. In: *arXiv preprint arXiv:1506.04579* (2015)

[LSD15] Long, Jonathan ; Shelhamer, Evan ; Darrell, Trevor: Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, S. 3431–3440

[Lü19] Lücke, Stephan: *Unter der Haube – Ein Blick in den Maschinenraum von VerbaAlpina.* `https://www.verba-alpina.gwi.uni-muenchen.de/?author=5`, 2019. – Accessed: 26.09.2019

[MKS+13] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Graves, Alex ; Antonoglou, Ioannis ; Wierstra, Daan ; Riedmiller, Martin: Playing atari with deep reinforcement learning. In: *arXiv preprint arXiv:1312.5602* (2013)

[MN18] Minar, Matiur R. ; Naher, Jibon: Recent advances in deep learning: An overview. In: *arXiv preprint arXiv:1807.08169* (2018)

[MNA16] Milletari, Fausto ; Navab, Nassir ; Ahmadi, Seyed-Ahmad: V-net: Fully convolutional neural networks for volumetric medical image segmentation. In: *2016 Fourth International Conference on 3D Vision (3DV)* IEEE, 2016, S. 565–571

[ODZ+16] Oord, Aaron van d. ; Dieleman, Sander ; Zen, Heiga ; Simonyan, Karen ; Vinyals, Oriol ; Graves, Alex ; Kalchbrenner, Nal ; Senior, Andrew ; Kavukcuoglu, Koray: Wavenet: A generative model for raw audio. In: *arXiv preprint arXiv:1609.03499* (2016)

[PCD15] Pinheiro, Pedro O. ; Collobert, Ronan ; Dollár, Piotr: Learning to segment object candidates. In: *Advances in Neural Information Processing Systems*, 2015, S. 1990–1998

[Pow11] Powers, David M.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. (2011)

[RDGF16] Redmon, Joseph ; Divvala, Santosh ; Girshick, Ross ; Farhadi, Ali: You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 779–788

[RFB15] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention* Springer, 2015, S. 234–241

[RHGS15] Ren, Shaoqing ; He, Kaiming ; Girshick, Ross ; Sun, Jian: Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in neural information processing systems*, 2015, S. 91–99

[SHM+16] Silver, David ; Huang, Aja ; Maddison, Chris J. ; Guez, Arthur ; Sifre, Laurent ; Van Den Driessche, George ; Schrittwieser, Julian ; Antonoglou, Ioannis ; Panneershelvam, Veda ; Lanctot, Marc u. a.: Mastering the game of Go with deep neural networks and tree search. In: *nature* 529 (2016), Nr. 7587, S. 484

[SLV+17] Sudre, Carole H. ; Li, Wenqi ; Vercauteren, Tom ; Ourselin, Sebastien ; Cardoso, M J.: Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In: *Deep learning in medical image analysis and multimodal learning for clinical decision support.* Springer, 2017, S. 240–248

[SS04] Sezgin, Mehmet ; Sankur, Bülent: Survey over image thresholding techniques and quantitative performance evaluation. In: *Journal of Electronic imaging* 13 (2004), Nr. 1, S. 146–166

[Sze10] Szeliski, Richard: *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010

[Tea19a] Team, OpenCV: *Image Thresholding OpenCV.* `https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html`, 2019. – Accessed: 20.07.2019

[Tea19b] Team, OpenCV: *Open Source Computer Vision Library (OpenCV).* `https://opencv.org/`, 2019. – Accessed: 24.06.2019

[Tis17] Tisato, Graziano: *NavigAIS, AIS Digital Atlas and Navigation Software.* `http://www3.pd.istc.cnr.it/navigais-web/`, 2017. – Accessed: 25.09.2019

[Tod08] Todorovic, Dejan: Gestalt principles. In: *Scholarpedia* 3 (2008), Nr. 12, S. 5345

[Tri11] Tripathi, KP: A review on knowledge-based expert system: concept and architecture. In: *IJCA Special Issue on Artificial Intelligence Techniques-Novel Approaches & Practical Applications* 4 (2011), S. 19–23

[UVDSGS13] Uijlings, Jasper R. ; Van De Sande, Koen E. ; Gevers, Theo ; Smeulders, Arnold W.: Selective search for object recognition. In: *International journal of computer vision* 104 (2013), Nr. 2, S. 154–171

[WGSM16] Wong, Sebastien C. ; Gatt, Adam ; Stamatescu, Victor ; McDonnell, Mark D.: Understanding data augmentation for classification: when to warp? In: *2016 international conference on digital image computing: techniques and applications (DICTA)* IEEE, 2016, S. 1–6

[ZMCL16] Zhu, Hongyuan ; Meng, Fanman ; Cai, Jianfei ; Lu, Shijian: Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation. In: *Journal of Visual Communication and Image Representation* 34 (2016), S. 12–27