DOI: 10.1111/tgis.12610

RESEARCH ARTICLE

Transactions in GIS WILEY

Cartographic reconstruction of building footprints from historical maps: A study on the Swiss Siegfried map

Magnus Heitzler | Lorenz Hurni

Institute of Cartography and Geoinformation, ETH Zürich, Zürich, Switzerland

Correspondence
Magnus Heitzler, Institute of Cartography and Geoinformation, ETH Zürich, Zürich 8092, Switzerland.
Email: magnus.heitzler@karto.baug.ethz.ch

Funding information
swissuniversities

Abstract

Extracting features from printed maps has been a challenge for decades; historical maps pose an even larger problem due to manual, inconsistent drawing or scribing, low printing quality, and geometrical distortions. In this article, a new workflow is introduced, consisting of a segmentation step and a vectorization step to acquire high-quality polygon representations of building footprints from the Siegfried map series. For segmentation, an ensemble of U-Nets is trained, yielding pixel-based predictions with an average intersection over union of 88.2% and an average precision of 98.55%. For vectorization, methods based on contour tracing and orientation-based clustering are proposed to approximate idealized polygonal representations. The workflow has been tested on 10 randomly selected map sheets from the Siegfried map, showing that the time required to manually correct these polygons drops to about 45 min per map sheet. Of this sample, approximately 10% of buildings required manual corrections. This workflow can serve as a blueprint for similar vectorization efforts.

1 | INTRODUCTION

The Topographic Atlas of Switzerland ("Siegfried map") is a comprehensive Swiss national map series that was published between 1872 and 1949. In the past, several attempts have been made to extract diverse features of interest from historical Swiss map series for different time steps (e.g., to understand change of wetlands [Kräuchi & Tschannen, 2015; Stuber & Bürgi, 2018], glaciers [Wiesmann, 2016], settlements [Hurni, Lorenz, & Oleggini, 2013], and accessibility [Fuhrer, 2016]).

Automated approaches are relatively rare or provide only rough results. The most comprehensive work on such an automated approach for a single feature class from the Siegfried map has probably been conducted by Leyk and Boesch (2009) for forest textures. Other notable efforts focused on the extraction of river bodies using color-segmentation approaches such as described by Leyk (2010), Leyk and Boesch (2010), and Iosifescu, Tsorlini, and Hurni (2016). However, these approaches are hardly able to distinguish between feature types of similar color (e.g., black buildings and labels) and show limited success in cases of color deterioration due to aging. Leyk, Boesch, and Weibel (2006) attempted to develop tailored algorithms for several different feature classes (forests, labels, etc.) for the Siegfried map, an approach that was later abandoned (Leyk & Boesch, 2009). Recently, research shifted towards the use of deep learning (DL) methods to extract features from historical maps (Uhl, Leyk, Chiang, Duan, & Knoblock, 2017) to avoid tediously crafting tailored algorithms. The results based on conventional architectures—such as LeNet-5 (Lecun, Bottou, Bengio, & Haffner, 1998)—look promising, but are still too blurry to be used to efficiently reconstruct footprints with cartographic quality.

Most of these approaches do not yield vector geometries but rather provide pixel-based predictions. For modern GIS and cartographic applications, a clean vector representation is desirable. Concatenating each pixel of a footprint contour is probably the simplest approach to obtain such a polygon. An example for such a result can be found in the work by Hecht, Herold, Behnisch, and Jehling (2018). Subsequently applying a generalization algorithm reduces the number of vertices but still does not yield a footprint of cartographic quality (i.e., one that is comprised of mostly parallel walls and perpendicular corners). A more sophisticated approach to select contour vertices has been described by Liu (2002), who analyzes the curvature of the contour in combination with the line subdivision method of Wall and Danielsson (1984). One method to obtain parallel walls and perpendicular corners is described by Hori and Okazaki (1992), and involves shifting vertices until a cost function is minimized. Another method is described by Frischknecht and Kanani (1998), who achieve similar results by formulating and solving a likelihood function based on geometrical constraints. Tools providing such functionality can sometimes be found as part of (commercial) software packages. However, they often lack descriptions of the underlying algorithms (see e.g., Hurni & Hutzler, 2008).

In this article, the complete workflow to generate high-quality building footprints for the Siegfried map is described, which can serve as a blueprint for similar efforts. This is achieved by making improvements to both main steps, segmentation, and vectorization. For segmentation, we employ an ensemble of 10 fully convolutional neural networks (FCNNs) to obtain precise pixel-wise predictions. For the vectorization process, we build on top of the method proposed by Liu (2002) to create a first coarse polygon for each footprint and introduce a new approach to obtain perpendicular corners and parallel walls using density-based clustering on transformed wall orientations. This approach is complemented with an additional cleaning step that is tailored to the elimination of common geometric artefacts. Furthermore, we introduce metrics to evaluate the quality of the obtained polygons and describe an iterative process that allows us to streamline training data generation.

The scanned map sheets of the Siegfried map used in this study were provided by the Federal Office of Topography Swisstopo, the Swiss national mapping agency. They were georeferenced using the methodology described by Heitzler et al. (2018). Overall, the Siegfried map consists of 3,903 map sheets, each with a width of 7,000 pixels and a height of 4,800 pixels. 3,098 sheets were published for scale 1:25,000 and scanned with a resolution of 1.25 m/pixel, mainly covering the Swiss plateau; 805 sheets were published for scale 1:50,000 and have a resolution of 2.50 m/pixel, mainly covering the Swiss alps.

2 | SEGMENTATION

FCNNs have shown impressive performance for image segmentation tasks (i.e., to determine whether a pixel belongs to a certain object or not). For training, they require a dataset that consists of input images (i.e., subsets of the input map sheet with the three RGB channels) and corresponding binary images. For this study, the

binary images consist of per-pixel information holding a one if the pixel belongs to a building, and zero otherwise.[1] Training data conforming to this format have been generated for 26 randomly selected map sheets for which building polygons have been manually digitized and which were subsequently rasterized.

Each map sheet has been subdivided into a grid of tiles with 200 × 200 pixels (Figure 1a). These dimensions are in line with common tile sizes for FCNNs (see e.g., Ronneberger, Fischer, & Brox, 2015; Shelhamer, Long, & Darrell, 2017). As each map sheet has a width of 7,000 pixels and a height of 4,800 pixels, this leads to 840 tiles per sheet. An example for such an input–output tuple is depicted in Figure 1b (input) and Figure 1c (output). The extent of the input tile is increased by 60 pixels on each side. The red square indicates the region for which predictions should be made. If only this inner region were provided for the model, it would be impossible to determine whether the cropped "C" at the right border is actually part of a building or part of a letter.

2.1 | Neural network architecture

The segmentation model is based on a modified version of the U-Net architecture introduced by Ronneberger et al. (2015). The version used is illustrated in Figure 2. The endpoints of this architecture are defined by the dimensions of the respective input and output tiles. The input tile is represented as a 320 × 320 × 3-dimensional matrix,[2] and the output tile is represented as a 200 × 200 × 1-dimensional matrix.

The actual ability of the U-Net to effectively carry out image segmentation lies in its two diametrical network paths, the contracting path (light purple area in Figure 2) and the up-sampling path (light green area in Figure 2). The contracting path consists of convolution layers and max-pooling layers, which generate increasingly higher abstract representations of the input image. The first layers of this path detect low-level features (e.g., corners, edges) and the further layers detect increasingly higher-level features (e.g., roads, buildings). The up-sampling path gradually combines these features on different granularities, which allows it to infer the class of a pixel based on its position and its surroundings.

Each convolution layer consists of a set of $3 \times 3 \times n_{\text{input}}$-dimensional filter kernels that iteratively adapt their values via backpropagation until they are able to detect certain features on the input matrix. Here, n_{input} is the

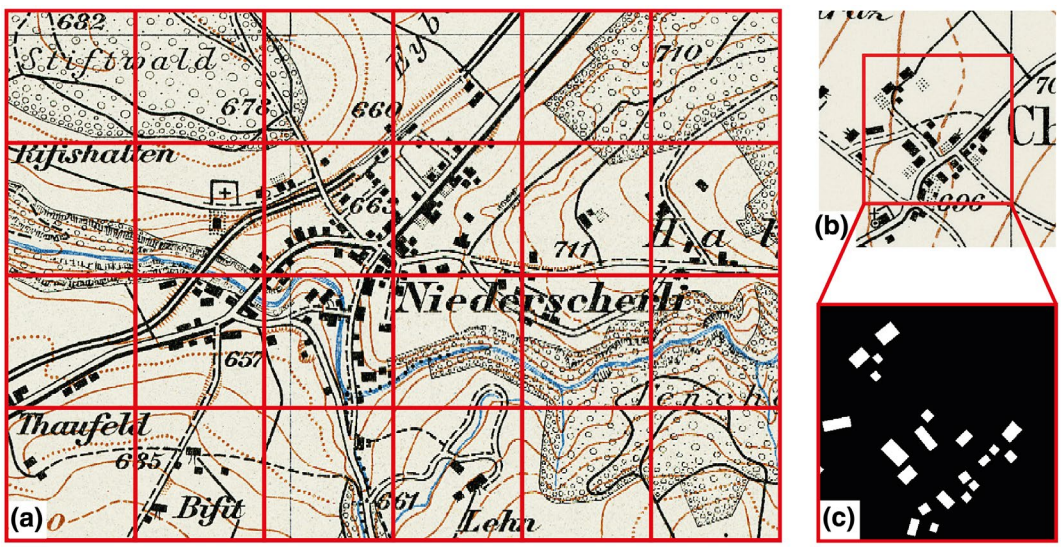

FIGURE 1 Tiling scheme used to train a fully convolutional neural network. The area to be analyzed by the FCNN is larger than the one for which predictions are to be made as this provides valuable context information. Geodata © Swisstopo

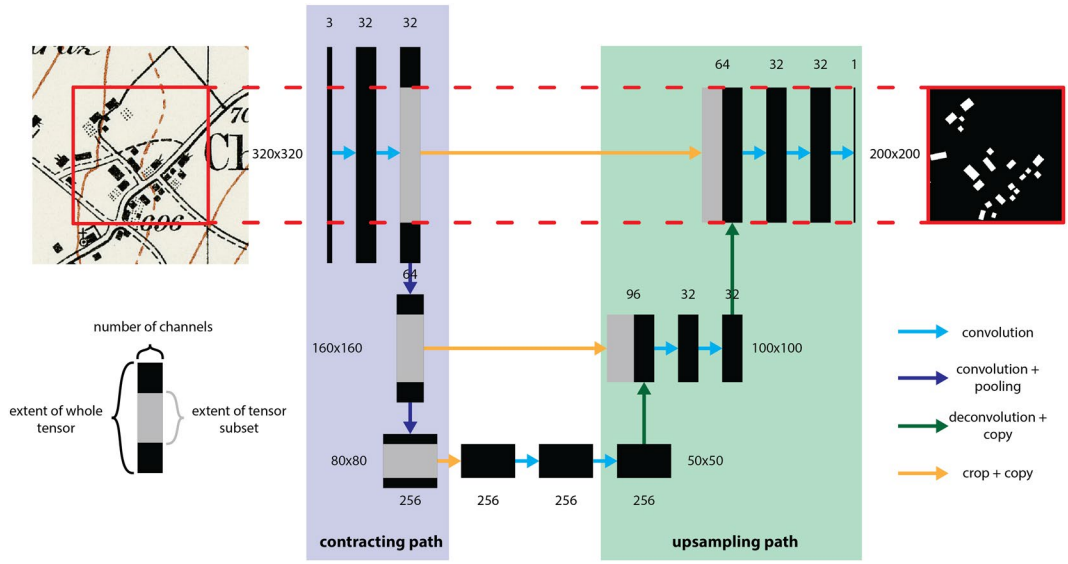

FIGURE 2 Adapted fully convolutional neural network based on the U-Net by Ronneberger et al. (2015). The contraction path (purple) and the up-sampling path (green) allow us to analyze the input tile at different levels of granularity. Geodata © Swisstopo

number of channels of the input matrix (e.g., 3 when analyzing the input RGB image). The number of filter kernels defines the number of channels of this output matrix. In the given network, 32 filter kernels were used, resulting in a matrix with dimensions 320 × 320 × 32.

Max-pooling layers subdivide the input matrix into groups of four neighboring cells and only keep that cell with the highest activation, effectively quartering the extent of the input matrix. Max-pooling layers are used in conjunction with convolution layers with a higher number of filter kernels. This allows us to reduce the extent of the input matrix, while increasing the number of channels.

In the given network, such max-pooling/convolution layers are used twice. The first layer creates a matrix with dimensions 160 × 160 × 64. The second layer creates a matrix with dimensions 80 × 80 × 256.

During the up-sampling path, so-called deconvolution layers are used to quadruple the extent of input layers. Deconvolution layers, in essence, work like ordinary convolution layers in that they train a set of filter kernels. The main difference lies in the matrix they operate on. For each cell of the input matrix, three neighboring empty cells are generated, leading to a matrix four times the size of the input matrix. The kernels of the deconvolution layer fill the empty cells based on the existing cells.

Since the extents of the intermediate layers of the contracting path are not a multiple of the target resolution of 200 × 200 pixels, it is necessary to crop the extent of each intermediate matrix. The matrix with an extent of 320 × 320 directly obtains the target extent of 200 × 200; the matrix with and extent of 160 × 160 obtains the extent of 100 × 100; and the matrix with the original extent of 80 × 80 obtains the extent of 50 × 50.

After applying two convolution layers to the latter matrix, the deconvolution layer with 32 kernels is applied, yielding a matrix with dimensions 100 × 100 × 32, which can be stacked directly with the respective cropped layer of the same extent. This leads to a combined matrix with dimensions 100 × 100 × 96. Afterwards, again two ordinary convolution layers are applied before another deconvolution layer yields a matrix with dimensions 200 × 200 × 32, which can be combined with the respective matrix with the same dimensions, forming a matrix with dimensions 200 × 200 × 64. Three subsequent convolution layers are applied, with the last having only one channel that holds the probability of each pixel being part of a building.

Each layer comes with a corresponding ReLU activation layer. Exceptions to this are the max-pooling layers that do not have any corresponding activation layers and the final convolution layer, which has an accompanying sigmoid activation layer. This ensures that the target probabilities are within the range [0, 1]. The vectorization model works better on a resolution higher than available on the source map sheets. Therefore, an up-sampling operation is included in the model as a final step (not depicted in Figure 2).

2.2 | Performance assessment

Training was carried out for 40 epochs, with a batch size of 6, using the Adam optimizer and binary cross-entropy as loss. The input data (21,840 tiles) was divided into 60% for training, 20% for validation, and 20% for testing. Overall, 10 models with different initial weights were trained to benefit from ensemble effects. For each model, the weights at that epoch with the smallest loss on the validation set were used to make predictions. The performance of these models on the test set is depicted in Table 1. The last line represents the performance of all models combined, by averaging their predictions. Two models (highlighted in orange) were not able to learn anything, and strictly classify any pixel as not being part of a building. The best value of the remaining models is highlighted in green for each measure. All measures were computed using scikit-learn, version 0.21 (Pedregosa et al., 2011). The prediction threshold is 50% for each measure, except for average precision, which accounts for the continuum of thresholds. These measures indicate satisfactory performance. The approach yields a low number of falsely recognized buildings (high precision value) and a high number of correctly recognized building pixels (high recall value), which is naturally reflected by the high f1 score. Accuracy has limited expressiveness as it is more sensitive to imbalanced data (only about 1% of the pixels belong to a building), and also shows high values if all pixels are classified as non-building. However, more complex measures that account for such issues and are commonly used in similar classification/segmentation tasks—such as the Jaccard index and average precision—also yield high values of 88.2 and 98.5%, respectively, indicating the good performance of the model. It is striking that the combined predictions performed best in every case except for recall, which is the reason that this ensemble approach was used to make the final predictions of the Siegfried map sheets.

It should be emphasized that these measures also have drawbacks for the specific case of vectorizing features from historical maps. There are infinitely many polygons that can represent a feature on a historical map

TABLE 1 Performance measures of the trained FCNN models

	Precision (%)	Recall (%)	f1 (%)	Accuracy (%)	Jaccard (IoU) (%)	Average precision (%)
1	93.142	91.918	92.526	99.850	86.091	98.002
2	93.824	92.051	92.929	99.858	86.793	98.230
3	94.156	92.251	93.194	99.864	87.255	98.324
4	87.365	87.475	87.420	99.746	77.651	94.551
5	93.604	92.544	93.071	99.861	87.040	98.282
6	92.778	92.177	92.477	99.848	86.006	97.935
7	93.568	93.175	93.371	99.866	87.567	98.301
8	94.251	92.547	93.391	99.868	87.602	98.392
9	0.000	0.000	0.000	98.990	0.000	1.010
10	0.000	0.000	0.000	98.990	0.000	1.011
	94.628	92.840	93.726	99.874	88.192	98.548

Note: Ensemble results are depicted in the last row. Models not able to learn are indicated in orange. Best performing models are indicated in green (excluding failed models).

appropriately. Yet, when predictions are made they are only checked against the one rasterized polygon drawn by the GIS operator. Hence, pixels that might very well be part of another valid polygon might erroneously be classified as a false positive or false negative when checked against the one given polygon. These issues mainly occur at the border pixels (the seam) of features. An example for this issue is depicted in Figure 3, and exemplified with the Jaccard measure also known as intersection over union (IoU) (Csurka, Larlus, & Perronnin, 2013).

Figure 3a covers a small area, with building predictions of different quality. Most buildings have been detected sufficiently well. A minority of buildings show missing pieces (1), predominantly when graphical deficiencies are present, and one building has not been detected at all (2). A small area is shown in Figure 3b to illustrate the well-delineated borders of the detected buildings, which is an important property when carrying out cartographic vectorization. Figure 3c shows the IoU values for individual buildings. Buildings with IoU greater than 80% are generally well represented, while IoU smaller than 80% usually indicates major flaws such as large gaps within the feature or even 0% if the building is missing entirely (3). The main differences between prediction and ground truth are present at the seam of each feature. As long as this seam is only a few pixels wide (e.g., up to 3), the building can be considered to be detected sufficiently well. However, it also becomes apparent that IoU is positively biased toward larger features. As the width of the seam is independent of the feature area, smaller IoUs are calculated for smaller features—even if they sufficiently represent a building (i.e., the seam is being over-emphasized). For example, the features highlighted at (4) have a relatively low IoU (75.56 and 80.21%), although they show little deficiency, while the feature at (5) possesses two gaps that complicate vectorization, yet has a high IoU of 87.48%. Similar issues can arise for other measures as well, and it might very well be worth exploring how these measures can be adapted to better reflect the quality of predictions when carrying out segmentation/vectorization tasks on historical maps.

To obtain a better picture of the performance of these models, 10 sheets were randomly selected from the Siegfried map series, which have not been used in the training/testing procedure. Map sheets 1–8 are of scale 1:25,000 and map sheets 9 and 10 are of scale 1:50,000. For each of these map sheets, the most densely populated region has been cropped and added to Appendix A of this article. This should give the reader a concrete impression of the quality of the proposed approach. Note that the depicted regions usually contain relatively complex building shapes. Simpler footprints (e.g., rectangles) in rural regions are usually recognized and vectorized in higher quality. For these map sheets, we conducted an analysis to quantify the performance of the models in respect to other features of the map. Hence, the false positive pixels were manually classified according to the actually depicted feature type. The results are listed in Table 2. Only those feature types with more than a thousand pixels are listed. All remaining feature types are grouped in the "other" category (e.g., rare boulder symbols).

By far the highest risk of being confused with a building exists for hachures as a means to represent relief. Likely reasons for this are that they locally might resemble buildings as they are being depicted as dense black spots and that little training data for these regions have been provided to the DL models (Figure 4a). This issue

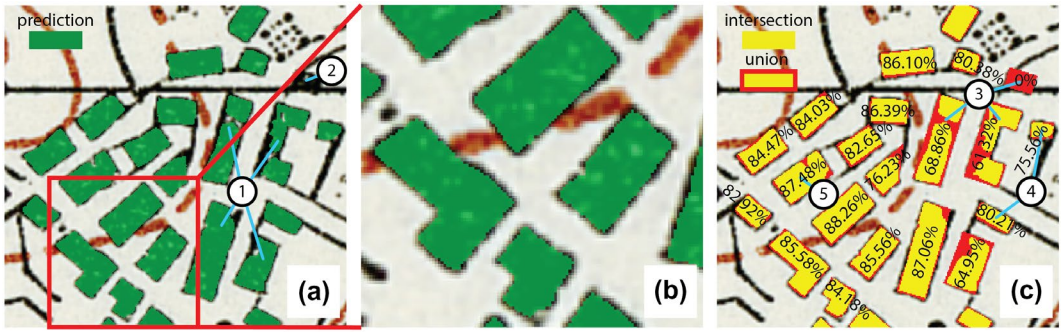

FIGURE 3 Example of segmentation results: (a) Pixel-based predictions for single buildings; (b) Detailed view on selected buildings with clearly delineated borders; and (c) IoU for single buildings. Geodata © Swisstopo

TABLE 2 Pixels classified as buildings and the actual feature class they belong to for 10 randomly selected map sheets

	Building	Hachure	Label	Border	Road	Height mark	Rail	Other
1	1,766,744	–	4,944	–	–	–	–	–
2	546,144	–	229	–	–	–	4,432	–
3	379,521	399	1,490	7	–	–	–	163
4	995,442	–	3,805	–	–	–	–	–
5	837,388	–	3,572	–	–	–	–	–
6	425,379	–	–	174	–	–	–	–
7	863,806	–	1,930	90	–	25	–	–
8	729,387	3,553	1,537	2,769	257	116	–	522
9	883,770	4,474	7,128	230	1,946	1,283	1,927	249
10	171,465	117,344	14,310	7,786	704	1,353	–	986
	7,599,046	125,770	38,945	11,056	2,907	2,777	6,359	1,920

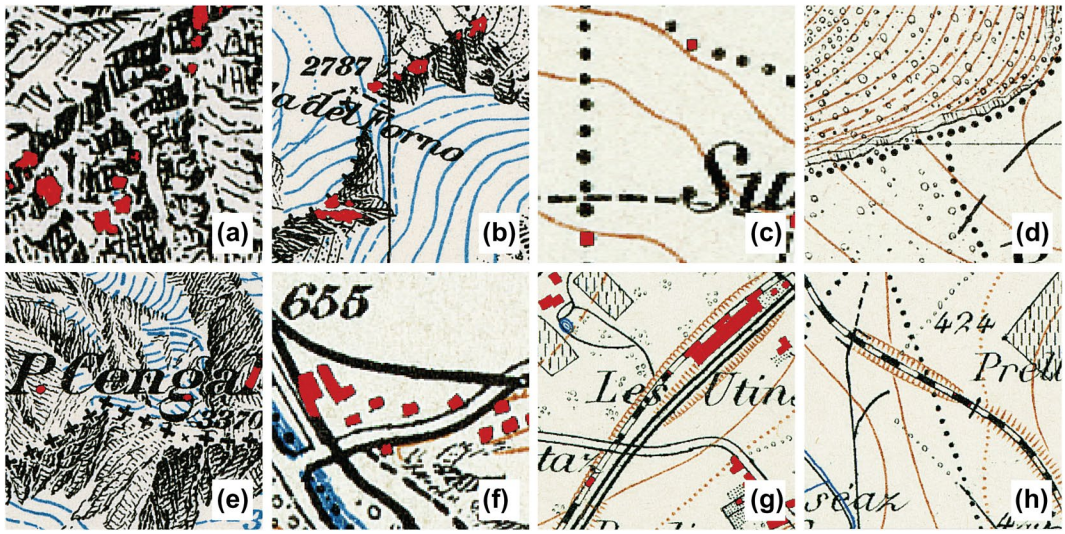

FIGURE 4 Examples for falsely recognized buildings: (a) Hachures in a mountainous region; (b) Administrative border represented as crosses superimposing hachures; (c/d) Administrative border represented as dots, partially misinterpreted as buildings; (e) Falsely recognized text segments. (f) Building extended along a neighboring road; and (g/h) Railways, partially misinterpreted as buildings. Geodata © Swisstopo

gets amplified if an administrative border symbolized by small crosses leads along a mountain ridge or if other such symbols are superimposed. The resulting black patches are likewise easily confused with buildings (Figure 4b), if they are not well separated from other textures (Figure 4e). A different symbolization for administrative borders is sequences of dots, which sometimes can take approximately rectangular shapes (Figure 4c). However, this seems primarily to be an issue of the quality of the underlying map sheet, as this problem is rare in regions of higher graphic quality (Figure 4d). More problematic are dots occurring in relative isolation, such as those indicating a height mark or that are part of a text (e.g., on the "i" or after an abbreviation), and pieces of text whose structure resembles those of buildings (Figure 4e). Texts with small font and consisting of thin lines are very rarely considered as buildings. Finally, roads and railways are sometimes likewise mistaken, although for different reasons.

Buildings bordering road symbols might be extended too much, leading to exaggerated building geometries. An extreme case of this issue is shown on the left of Figure 4f. Rail symbols are depicted as chains of black and white rectangles, of which some segments can be misinterpreted (Figure 4g). This seems to be largely dependent on the orientation of the rail and the surrounding area, since long sequences can be encountered that do not possess any issues (Figure 4h). Summarizing, of all pixels classified as buildings, only 2.4% have been falsely classified, with hachures making up the majority of 1.6%. While such an analysis gives a general impression of model performance, the internal processes of the neural networks leading to these results remain vague. At the time of writing, powerful analysis methods that shed light on the reasoning of DL models are scarce, as these models are widely regarded as being "black boxes" (O'Mahony et al., 2020).

3 | VECTORIZATION

Before a polygonization effort is started, it needs to be defined which criteria the resulting polygons should meet. One should also take into consideration the mapping and drawing methods used when establishing the Siegfried map and the resulting accuracy (Swisstopo, 2019): geodetic fixed points were established by triangulation down to third order. The network was then densified by graphical triangulation directly on the survey reference map to provide additional points for plane table measurements. It is therefore obvious that only a small number of buildings (landmarks such as churches) were determined by triangulation, and the vast majority of buildings were measured by graphical forward reading. This resulted in manuscript maps which were used as drawing keys for the printing plates: copper engraving for the 1:25,000 scale maps and lithography for the 1:50,000 scale maps. Despite this tedious work, the geometrical and graphical accuracy cannot be compared to today's topographical maps. Internal guidelines permitted an average deviation over 10 points of 35 m, with a tolerated limit of maximum 75 m for the 1:50,000 map and 12.5/30 m for the 1:25,000 map, respectively. Visual inspection and comparison with the current topographical map, however, shows that there are still cases above the maximum limit. The reversed engraving on copper and limestone, respectively, was state-of-the-art technology in the 19th and early 20th century, but the buildings on the printed Siegfried maps nevertheless obtained blurry borders rather than clearly delineated walls. This is also due to the three-color printing and the RGB scanning from a not-color-separated (i.e., printed) original, resulting in color fringes on the edges. Modern standards would demand such borders being straight lines rather than perfectly reconstructing the border pixels. Hence, idealized polygons are being created by vectorization. For the same reason, it is desirable to depict right angles and parallel walls, even in cases where these properties cannot be unambiguously derived from the drawn buildings. Finally, the number of vertices per footprint polygon should be as low as possible, while sufficiently representing the general shape. The major steps of how these geometries were obtained are shown in Figure 5.

3.1 | Corner point detection

For each detected building, a distinct contour is generated using the Canny (1986) algorithm. This contour is then analyzed to find characteristic corner points in the step corner point detection (Figure 5, no. 2). Corner point detection is based on the idea of contour tracing. For each pixel along the contour, the angle between the next and

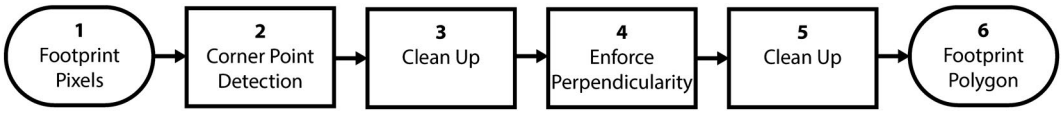

FIGURE 5 Major steps of the polygonization process

previous pixel is calculated. A threshold can be applied to the resulting sequence of values to distinguish between corners (high angle) and straight or slightly curved walls (low angle). Determining this threshold is hardly possible when working on these data directly, as the (absolute) angles can only take the values 0°, 45°, and 90°. An example for such a contour is given by the blue sequence in Figure 6. Applying a threshold to filter out only 90° angles (red bar) yields too few corners (yellow pixels in Figure 6a). Applying a threshold to filter out 45° and 90° angles (green bar) yields too many corners (yellow pixels in Figure 6b). Hence, the angles are smoothed using two mean filters of size four, resulting in the orange sequence in Figure 6.

Applying a threshold of 10° on this sequence better approximates pixels that are part of a corner (Figure 6c). This approach seems to be related to the workflow described by Liu (2002), who states that they measure the "angluar difference between the slopes of two line segments fitting to data around each curve point of distance *k*." It is not described which value *k* takes and how this fitting procedure is being carried out. We assume that both approaches are identical if $k = 1$ (i.e., the distance of one pixel along the contour). We conducted experiments with several values for *k* and smoothing operations, but did not find a suitable way to: (1) unambigously determine corner pixels; and (2) calculate a suitable position of a corner vertex for each cluster. An example for corners that were not detected due to a particularly curved wall is depicted in Figure 7c.

A method similar to the approach of Wall and Danielsson (1984) is then used to compensate for these shortcomings. During this step, the sequence of pixels between each two neighboring corners is traversed in order to detect such false negatives. This process is illustrated in Figures 7a and b. For each corner candidate (orange dot), a polygon is constructed consisting of all the neighboring pixels until and including the next and previous corner pixels (red dots). If the quotient of the area and the circumference of this polygon are larger than a certain threshold,

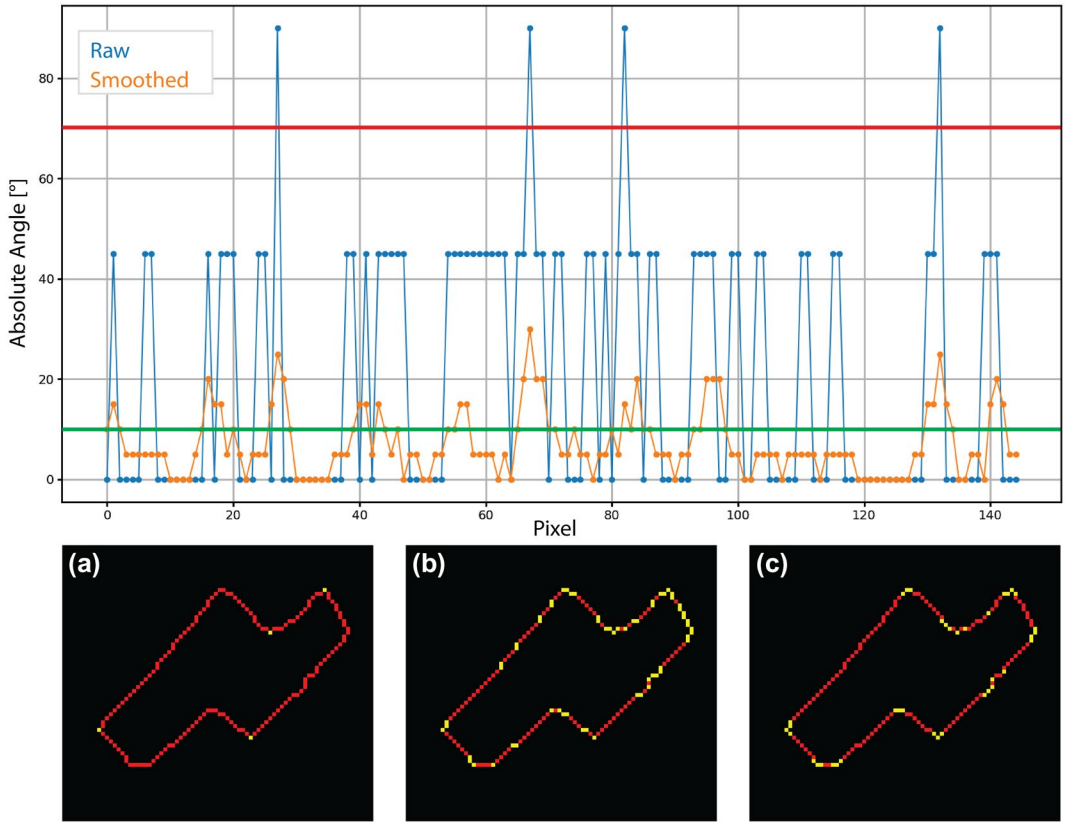

FIGURE 6 A graph representing the differences in two adjacent angles along the contour of a detected building. Thresholds were applied to the raw and smoothed values to determine corners. Geodata © Swisstopo

the pixel is considered to be a corner, else it is considered to be part of a wall. This threshold was set to 0.3 pixels. In the example of Figure 7, the algorithm successfully detected two additional corners. The same principle is used to determine whether an existing corner pixel should be kept or discarded. In this case, the threshold was set to 0.2 pixels (i.e., false positives are treated with lower tolerance than false negatives to reduce the number of vertices). The polygon vertices are calculated by intersecting two neighboring straight walls. We found this approach to be more precise than calculating the vertices based on cluster pixels.

3.2 | Clean up

As depicted in Figure 7d, small artefacts can occur that either stem directly from the vectorization step or might be propagated due to an inaccurate segmentation. The most common artefacts are separately treated during this step to obtain a cleaner geometrical representation. How they are being resolved is shown in Figures 8a–d. Small jumps (Figure 8a) are very small discontinuities along an otherwise (roughly) straight wall. Peaks (Figure 8b) are small, typically triangular structures that either point inward to or outward from a polygon. These are either closed (in the former case) or changed to a rectangle. The rationale behind this is as follows. Both types of peaks usually occur as a consequence of false negatives during the segmentation step, leading to pixels missing inside the actual building (inward peaks) or pixels missing for small structures attached to a building (outward peaks). In both cases, the geometry is regarded as being incomplete and thus the missing structures are estimated. Beveled corners (Figure 8c) are another common artefact that can easily be fixed by extending the neighboring, longer, walls. Figure 9a shows how the artefact present in Figure 7d was fixed by this rule. Finally, Figure 8d shows oblique walls (i.e., walls that do not align orthogonally to two neighboring parallel walls). This step is only being carried out in the second clean-up step (Figure 5, no. 5) after parallel walls and perpendicular corners have been enforced.

Of course, it is possible that some of the detected artefacts are in fact correctly recognized building parts and that the clean-up step leads to false corrections. In practice, however, it is far more common that these geometries are actual artefacts and thus clean up does more good than harm to the end result. The main threshold below

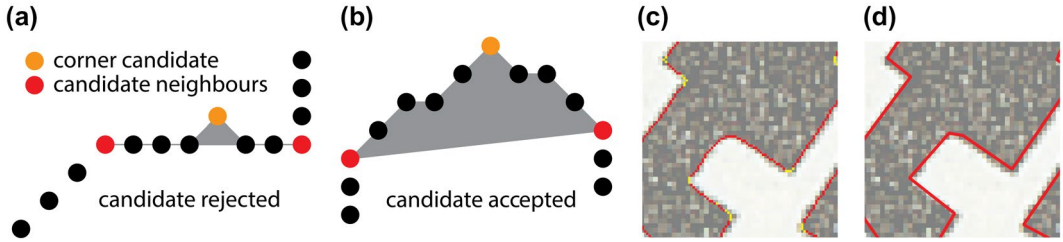

FIGURE 7 Illustration of the concept (a/b) and example results (c/d) when applying the method of Wall and Danielsson (1984). Pixels are regarded as polygon vertices if the ratio of the area and the circumference spanned by two corners and a corner candidate is larger than a certain threshold. Geodata © Swisstopo

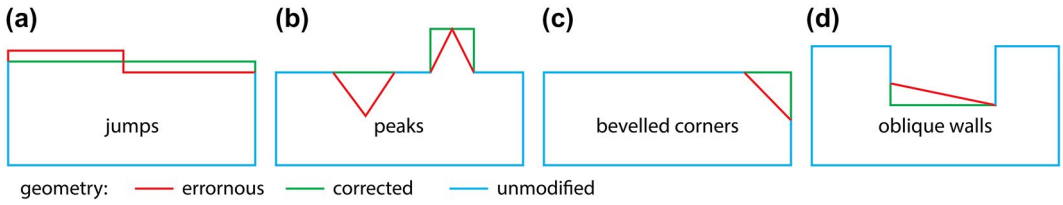

FIGURE 8 The clean-up step consists of several algorithms that are tailored to remove common artefacts

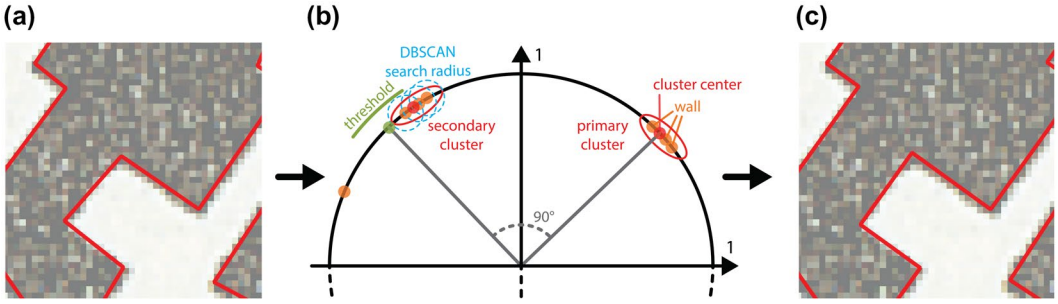

FIGURE 9 The slope of each wall is plotted on a unit circle as a point. Walls with similar slopes are clustered and their orientations are unified to obtain perpendicular corners and parallel walls. Geodata © Swisstopo

TABLE 3 Different length thresholds in pixels depending on the artefact type and clean-up step

	Clean-up step 1				Clean-up step 2			
	Small	**Medium**	**Large**	**Very large**	**Small**	**Medium**	**Large**	**Very large**
Jumps (pixels)	1	2	3	4	1	2	3	4
Beveled corners (pixels)	4	6	8	10	16	24	24	30
Peaks (pixels)	12	12	12	12	36	36	18	18
Oblique walls (pixels)	—	—	—	—	12	12	12	12
	Enforce perpendicularity				Building size (pixels)			
	Small	Medium	Large	Very large	Small	0–99		
Angle threshold (°)	30	30	30	25	Medium	100–799		
Search radius (—)	0.25	0.25	0.25	0.25	Large	800–1,599		
Ratio threshold (pixels)	0.6	0.6	0.6	0.5	Very large	≥1,600		

which the length of a wall (i.e., red lines in Figure 8) has to fall to be considered for fixing is listed in Table 3. This value depends on the artefact type, the area of the building, and whether it is being executed in the first or second clean-up stage.

3.3 | Enforce perpendicularity

Perpendicularity is enforced using a method that is considered simpler than the ones described by Hori and Okazaki (1992) and Frischknecht and Kanani (1998). Rather than iteratively solving an optimization problem, this method carries out rectification by simple geometrical means. The principle is depicted in Figure 9b. In a first step, the angle of each wall is being determined and then plotted on a unit circle (orange points in Figure 9b). Walls with similar angles are therefore represented as points close to each other.[3] In a second step, these points are being clustered with the DBSCAN algorithm (Ester, Kriegel, Sander, & Xu, 1996). Hence, walls with similar angles are grouped together and the average angle is calculated (red circles). Each wall within a group gets assigned this average angle (red points), effectively unifying their orientation. In a third step, the cluster whose sum of all included wall lengths is highest is picked as the primary cluster. For this primary cluster, it is determined whether there is a

corresponding secondary cluster approximately 90° apart, which holds the orthogonally oriented walls. If found, all walls contained in that secondary cluster are being oriented to be exactly 90° apart from the orientation of the primary cluster (green point). This way, perpendicularity and parallelism are obtained. This algorithm requires two inputs, the search radius of DBSCAN (exemplified with blue dashed circles) and a threshold defining how far the secondary cluster can be located apart from the point 90° away from the main cluster (green line). In addition, it makes use of the ratio of area to circumference, as described in the corner point detection step, to only adjust those walls which would not deviate too much from their original position. The concrete values of these parameters can be found in Table 3. The result of the process is exemplified in Figure 9c.

4 | RESULTS

For evaluation, the results of the 10 randomly picked map sheets already described in Section 2 were investigated. Flaws in the resulting geometries were manually corrected, leading to an additional, corrected dataset. The source dataset and the corrected dataset were then automatically compared to determine the quality of the vectorization procedure, yielding the attributes listed in Table 4. The following list describes the meaning of each attribute. The last row holds the accumulated attributes and weighted averages.

- # Buildings predicted: Number of all automatically generated building footprints.
- # Buildings unchanged: A building in the source dataset is considered unchanged if there exists a corresponding building in the corrected dataset that covers the exact same area (i.e., the symmetrical difference between the two polygons is empty).
- # Buildings changed: A building in the source dataset is considered changed if there exists at least one corresponding building in the corrected dataset that covers partially the same area (i.e., the intersection between the two is non-empty and they are not identical; that is, the symmetrical difference between the two is non-empty).
- # Buildings deleted: A building in the source dataset is considered deleted if there exists no polygon in the corrected dataset that at least partially covers the same area (i.e., the intersection between the two polygons is empty).
- # Buildings added: A building in the target dataset is considered added if there exists no polygon in the source dataset that at least partially covers the same area (i.e., the intersection between the two polygons is empty).
- # Buildings corrected: Change in the number of buildings due to corrections.
- IoU: The intersection and union datasets are created based on the source and corrected datasets using the respective GIS-based overlay operations. For both resulting datasets, the areas of their features were summed up. Dividing the accumulated intersection areas by the accumulated union areas yields the IoU.
- Area predicted: Accumulated area of the predicted buildings. The unit pixels have been added as using another meter alone complicates the comparison of sheets of vectorization produced for different scales.
- Area changed: Change in the area due to correction.
- Vertices predicted: The number of vertices the buildings of the sheet consist of.
- Vertices changed: Change in the number of vertices due to correction.
- Correction time: Total time required to correct a sheet.
- Correction speed: Speed to correct building footprints.

The number of buildings generated ranges from around 600 for rural areas to around 3,000 for sheets covering larger settlements (e.g., the city of Thun, see Appendix A1). Across all sheets, around 90% of the nearly 15,658 buildings did not require any manual corrections, 673 footprints required manual changes, and 773 footprints have been deleted as not being considered a building. Another 311 buildings needed to be added as they were not recognized. The corrected dataset therefore contained 462 less buildings than in the source dataset. The overall

TABLE 4 Performance measures and correction time for 10 randomly selected map sheets of the Siegfried map

	# Buildings predicted	# Buildings unchanged	# Buildings changed	# Buildings deleted	# Buildings added	# Buildings corrected	IOU (%)	Area predicted	Area changed	# Vertices predicted	# Vertices changed	Correction time (min)	Correction speed (buildings/min)
1	3,169	3,009	122	38	76	38	96.19	1,634,897 px 638,631 m^2	+46,613 px +18,208 m^2	13,733	+508	78	40.63
2	580	517	53	10	9	−1	95.70	522,176 px 203,975 m^2	+6,197 px +2,421 m^2	3,051	+86	25	23.2
3	596	552	31	13	9	−4	95.71	354,509 px 138,480 m^2	+8,040 px +3,141 m^2	2,606	−15	22	27.09
4	2,092	2,007	52	33	40	7	96.84	915,627 px 357,667 m^2	+16,144 px +6,306 m^2	8,648	+62	44	47.55
5	1,700	1,631	47	22	42	20	96.51	776,031 px 303,137 m^2	+15,480 px +6,047 m^2	7,278	+87	44	38.64
6	785	777	5	3	0	−3	99.40	394,602 px 154,141 m^2	+936 px +366 m^2	3,308	−4	9	87.22
7	1,280	1,188	71	21	7	−14	96.15	806,392 px 314,997 m^2	+23,342 px +9,118 m^2	5,329	−116	34	37.65
8	1,203	1,095	45	63	11	−52	96.73	686,093 px 268,005 m^2	−57 px −22 m^2	5,055	−118	40	30.08
9	3,166	2,849	215	102	109	7	90.56	800,247 px 1,250,386 m^2	+32,105 px +50,164 m^2	12,921	−87	88	35.98
10	1,087	587	32	468	8	−460	52.83	277,424 px 433,476 m^2	−122,260 px −191,031 m^2	4,871	−2,263	30	36.23
	15,658	14,212 (90.77%)	673 (4.3%)	773 (4.94%)	311 (1.99%)	−462 (−2.95%)	92.33	7,167,997 px 4,062,895 m^2	+26,539 px (+0.37%) −95,283 m^2 (−2.35%)	66,800	−1,860 (−2.78%)	414	37.82

IoU amounts to 92.33%, indicating a satisfactory vectorization performance. Note that this value is larger than for the pixel-wise predictions on the test set described in Section 2. It is suspected that the reason for this lies in the more tolerant assessment of the vectorization results, as multiple footprint polygons may be sufficiently representing a depicted building; automatic pixel-wise comparisons do not show such tolerance.

Overall, approximately 7 million pixels (4 km^2) of building footprints consisting of 66,800 vertices have been generated. While the area in terms of pixels has remained nearly unchanged, around 0.1 km^2 and 1,860 vertices were removed due to correction. This difference stems from the smaller scale of sheets 9 and 10. Correcting all sheets amounted to approximately 7 hr in total (41 min/sheet), with an average investigation/correction rate of around 38 buildings/min. An example of typical vectorization flaws—and how they have been corrected—is depicted in Figure 10. For comparison, 100 buildings for each of the 10 map sheets were digitized from scratch, starting at the northwest corner and continuously working eastward. On average, around 10 buildings could be vectorized per minute, with large deviations in urban areas with complex footprints (~5 buildings/min) and rural areas with simple shapes (~14 buildings/min).

To some degree, these numbers are a matter of subjectivity, as cartographers and GIS specialists might exhibit varying correction performance and their perception on which flaws require correction might differ. However, there are certain sheet properties that are well worth discussing. Sheets with a larger amount of buildings tend to show a better correction performance (buildings/min). This is likely due to the fact that manually scanning each part of a map sheet has to be accomplished irrespective of the number of buildings, leading to a relatively poor performance for sheets with few buildings. However, there is clearly an outlier to this suspected rule. Sheet 6 could be corrected with a relatively high speed of 87 buildings/min. This sheet has a very high print quality, with buildings being cleanly delineated from the background, shows only a few features with similar shapes to buildings (except for labels), and nearly all buildings have simple geometries. This not only makes investigating the sheet easy, it also leads to good vectorization results with very few corrections necessary, which is reflected by a high IoU of 99% and few changes for area and number of vertices. In contrast, sheet 10, whose bad predictions were already discussed before, lost more than 44% in area and 46% of the generated vertices due to correction, which mainly comprised deleting polygons in mountainous areas.

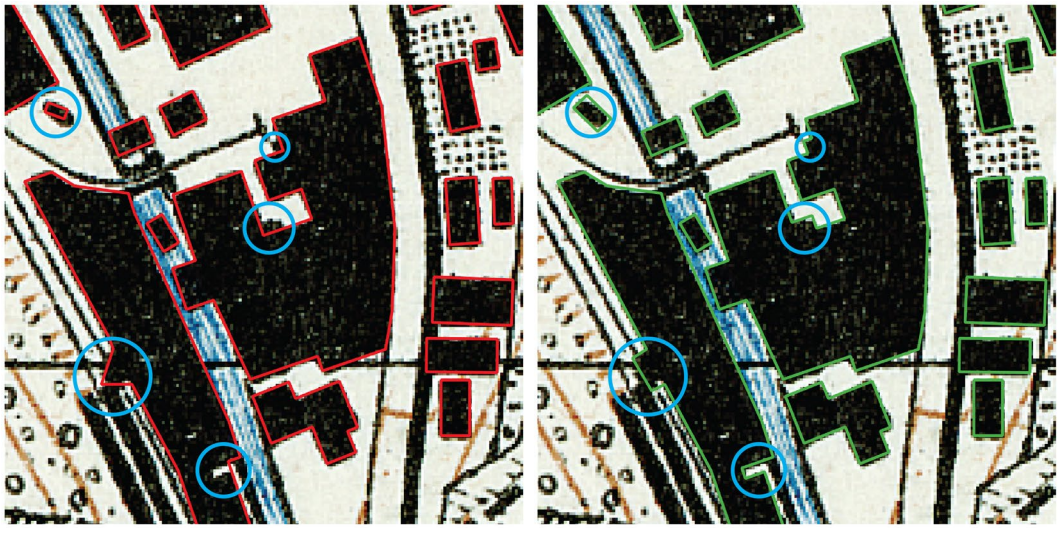

FIGURE 10 Example of geometrical flaws in the vectorization output (left) and the corresponding corrected polygons (right). Geodata © Swisstopo

5 | DISCUSSION

The results of the proposed vectorization approach look promising, especially when quantifying their quality based on the 10 sample map sheets. However, it is likely that the models will not generalize well in areas that have not been sufficiently represented in the training dataset or are of low graphical quality. A more sophisticated assessment has to be undertaken to determine such issues. A first indication is given by sheet 10 listed in Table 3, for which the models did not yield good results for hachures. This might require retraining the models based on an adapted training dataset that specifically incorporates mountainous areas.

In fact, the described approach can be embedded in a more generic process that allows us to iteratively improve the vectorization results (Figure 11). The methods and results described represent the latest implementation of this process. Originally, six map sheets were manually generated (Figure 11, no. 1) that served as the first training dataset (Figure 11, no. 2). These sheets were used to develop and test several different approaches in the automated segmentation (Figure 11, no. 3) and automated vectorization (Figure 11, no. 4) steps. These first algorithms, however, did not yield results that were regarded as being of great value to other researchers. Hence, the resulting footprint polygons have been manually corrected (Figure 11, no. 5) and the segmentation and vectorization models have been gradually improved. This way, the data pool has been iteratively enlarged while experimenting with different algorithmic approaches. Ultimately, 26 map sheets originated from these preliminary iterations that were used to train the most recent segmentation model, whose vectorized results were finally published (Figure 11, no. 6). This resulted in approximately 6 million buildings, covering all Siegfried map sheets, which were made available to other researchers via a geoportal.

If no more iterations of this process were to be conducted, it would take approximately 2,900 hr or 363 8-hr working days to manually correct all sheets, assuming a conservative correction time per sheet of 45 min. Since this still represents a significant effort, it is considered to publish the data under a Creative Commons Share Alike, or similar, license, obligating users of this data to publish any corrections they make. Such a license has to be chosen with care, to avoid collisions with existing usage rights, such as those imposed by Swisstopo, the copyright holder of the Siegfried map.

One important issue when training a neural network for building segmentation (and likely for many other feature types as well) is the relatively small amount of training data available. In fact, in the whole dataset (training, validation, testing), only about 1% of all pixels actually belong to a building, leading to highly imbalanced classes. This increases the risk of a model not being able to distinguish buildings from other features, as high accuracies can be obtained by classifying each pixel as not belonging to a building. This issue is exemplified by models 9 and 10 in Table 1. A recent survey paper (Johnson & Khoshgoftaar, 2019) thoroughly discusses three strategies to address imbalanced training datasets: data-level methods (e.g., over- and under-sampling), algorithm-level methods (e.g., class weighting), and hybrid approaches combining both methods. While we consider a thorough investigation of such methods in the context of historical maps worthwhile, we found a relatively straightforward solution to reduce the risk of models ignoring the minority class (e.g., buildings). After a few iterations of experimenting and frequently obtaining unsuccessful models, we provided two additional map sheets that cover cities, which possess

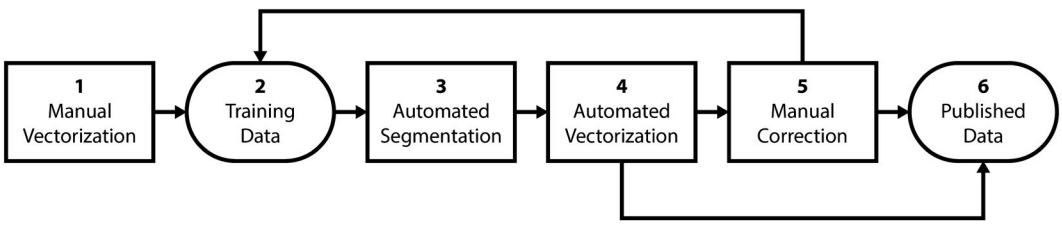

FIGURE 11 High-level view of the iterative workflow to gradually generate better building footprints for the Siegfried map series

a considerable number of tiles containing many building pixels. For example, of the 840 tiles per sheet, 111 and 68 tiles, respectively, cover more than 20% of building pixels. This seems to give the models sufficient information to "get on the right track."[4] First tests indicate that the probability of this issue arising can be reduced considerably when the training dataset is sampled in such a way that more pixels are covered by the target feature class as supported by the plugin described in Outlook.

The question remains whether DL is superior to other extraction algorithms that rely on more traditional computer vision techniques, such as color segmentation, morphological operators, and template matching, or their combination. While the literature rarely provides enough information to re-implement existing approaches (see for example, the vague workflow explanations by Frischknecht and Kanani 1998 and Hecht et al. 2018), a comparison on a higher level is possible. In fact, O'Mahony et al. (2020) provide a thorough discussion on this subject. They conclude that DL is indeed superior to conventional techniques, primarily because it automates the tedious step of manual feature engineering,[5] which requires "a long trial and error process" and fine-tuning the "plethora of parameters by the CV engineer." In contrast, DL is considered overkill for many simple tasks, it might not perform well if the image to be analyzed is not adequately represented in the training dataset, assembling a sufficiently large dataset for training might be challenging, and DL models are often regarded as black boxes. Finally, they state that combining both DL and traditional computer vision can be the best choice.

Each of these considerations is relevant for the specific case of extracting buildings or other features from historical maps. Creating a classifier using conventional computer vision techniques via feature engineering in the context of historical maps is very tedious, as the engineer not only needs to analyze the feature to be predicted, but also needs to account for all those feature types that are likely to be mistaken for it. In addition, the developed workflow needs to be modified once it is realized that it is not performing well for certain map sheets. The latter is also true for a DL pipeline. However, this issue can be solved by simply adding data for the areas in question. Subsequently, (re-)training such models only requires processing time. This shifts somewhat the human involvement from software development to data preparation. From a practical point of view, this is beneficial as data preparation can be carried out by operators with basic knowledge of GIS/cartography, while software development requires more training. Following a hybrid approach can likewise be very beneficial. For example, small holes in predictions can be removed with conventional closing operations, considerably improving prediction quality with little programming effort.

However, we see a considerable limitation of the DL approach in that it only yields pixel-wise predictions, while ultimately vector products are desired. In a way, the vectorization procedure described in this article can be compared with traditional computer vision techniques since much effort has to be invested in "feature engineering" (e.g., treating the several cases in the clean-up step). Recent advances in DL-based polygonization models (see e.g., Li, Wegner, & Lucchi, 2019) aim to replace this task.

Ultimately, all these considerations require further investigation. This encompasses the exploration of different network architectures and their suitability regarding different feature types. While, for example, similar results could already be obtained for the extraction of hydrological features, features with a higher degree of complexity, such as labels, might require entirely different approaches or might be detected more easily with more conventional methods such as templates. A thorough analysis on the suitability of different approaches—depending on feature type—might be worthwhile.

6 | OUTLOOK

Although the proposed approach yields promising results and can likely be easily adapted to other feature classes, there are still some hurdles that limit its application in practice. First, GIS experts and cartographers usually are not familiar with machine learning techniques, and easy-to-use interfaces between both packages are rare. Second, there is typically not much flexibility when assembling an appropriate training dataset. Working on the

sheet level is often insufficient when more fine-grained adjustments are required (e.g., to limit the number of empty tiles in the training dataset). Third, historical map series are usually multidimensional in nature, which requires rarely supported multidimensional navigation and editing capabilities. For example, in the Siegfried map series, the same area can be covered by sheets published in different years, for different scales, and even different versions of the same sheet (e.g., changes in symbolization, improved localization of features).

These considerations led to the development of a plugin for the free and open-source software QGIS (see Figure 12). The bar chart at the bottom of Figure 12 shows the number of sheets for each year, including the number of corrected sheets (green segments). Clicking on one of the bars loads the relevant sheets and vectorization results in the map window shown in the center. The yellow area indicates automatically generated vectorization results; the green area indicates already corrected vector data that is suitable for training. Training data is simply defined by a point dataset, which can be created either using common digitizing tools or using algorithms from the processing toolbox (e.g., randomly create points in area). In Figure 12, each point has been symbolized according to the context region (outer square) and prediction region (inner square). At the time of writing, an external script needs to be executed to carry out training. However, it is intended that all required functionality can be accessed from within QGIS in the future.

The first results using this plugin are shown in Figure 13 for the extraction of hydrological features, following the same workflow as being carried out for extracting buildings. The main differences are that multiple features are predicted at the same time: streams, rivers, and lakes (merged to one class due to very similar symbolization), and wetlands (not shown). Vectorization of the results can be carried out by basic raster-to-vector conversion and subsequently applying a generalization algorithm (in this case, Douglas–Peucker). Lakes and rivers are differentiated based on morphological characteristics (circumference, area, etc.) during the vectorization step.

From a conceptual point of view, it is aimed at addressing two main issues, which will likewise be integrated into the plugin. First, the development of algorithms to carry out topologically correct matching of temporally and spatially aligned features (e.g., the same road in two time steps should be represented as identical polylines).

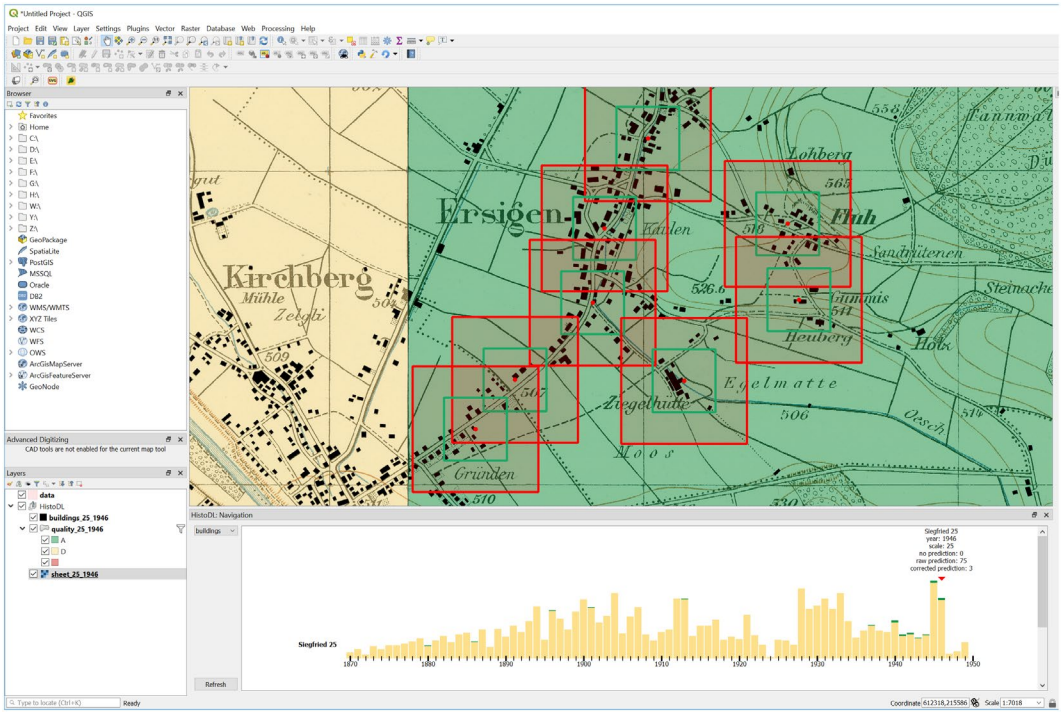

FIGURE 12 Prototypical QGIS plugin to support the vectorization of historical maps. Geodata © Swisstopo

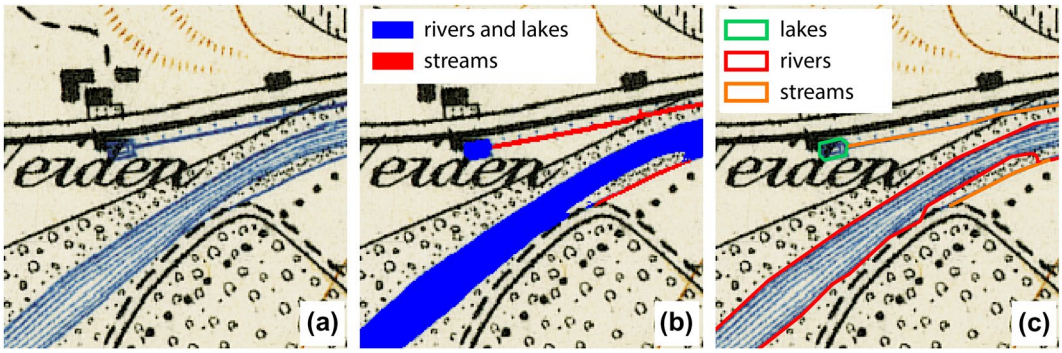

FIGURE 13 Example of a multiclass segmentation/vectorization approach: (a) Original map; (b) Automatically segmented map; and (c) Automatically vectorized segmented map. Geodata © Swisstopo

Second, different deep learning methods need to be explored, such as emerging polygonization networks that would combine segmentation and vectorization into a single step. In the long term, it is aimed to make the plugin accessible to the scientific community to enable other groups to carry out vectorization efforts on historical maps with little knowledge of deep learning.

7 | CONCLUDING REMARKS

This article describes a workflow that efficiently allows generating high-quality building footprint polygons from the Siegfried map. This is achieved by training an ensemble of 10 neural networks based on the U-Net architecture that effectively allows us to distinguish building pixels from non-building pixels. Subsequently, the detected buildings are being polygonized using a set of tailored algorithms, which, in essence: (a) detect corners by analyzing the gradient of each footprint contour; (b) enforce right angles by analyzing the orientation of each wall; and (c) remove common geometrical artefacts. These steps can be embedded into a more generic process to gradually accelerate the generation of training data. The nearly 6 million resulting building footprint polygons are being made available to the scientific community of Switzerland by integrating them into a geoportal. Future efforts focus primarily on the development of a plugin for QGIS enabling professionals with little deep learning knowledge to carry out vectorization on other historical map series and feature types, as well as the improvement of topological properties in space and time of the resulting vector datasets.

ORCID

Magnus Heitzler https://orcid.org/0000-0002-9021-4170
Lorenz Hurni https://orcid.org/0000-0002-0453-8743

ENDNOTES

[1]Multiple binary channels are possible, each holding information for a different feature class.

[2]In the deep learning literature, such high-dimensional matrices are sometimes referred to as tensors.

[3]Note that the absolute angles are plotted on the unit circle and therefore only the upper two quadrants are populated. These values are afterwards linearly stretched to cover the whole unit circle, which is necessary to account for border cases.

[4]As a side note, models that are not able to differentiate can usually be spotted early during training. After a very few epochs (usually two or three), their accuracy will tend to not change anymore and will be equal to the fraction of non-building pixels to building pixels (again, see accuracy for models 9 and 10 in Table 1). Training can then be cancelled and one of the mentioned strategies can be implemented.

[5]According to O'Mahony et al. (2020), feature engineering is the process of determining features of interest and writing algorithms that extract these features. The features are then searched for in the image and used for classification.

REFERENCES

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *8*(6), 679–698.

Csurka, G., Larlus, D., & Perronnin, F. (2013). What is a good evaluation measure for semantic segmentation? In *Proceedings of the British Machine Vision Conference*, Bristol, UK (pp. 32.1–32.11).

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, OR. Menlo Park, CA: AAAI.

Frischknecht, S., & Kanani, E. (1998). Automatic interpretation of scanned topographic maps: A raster-based approach. In K. Tombre & A. K. Chhabra (Eds.), *Graphics recognition: Algorithms and systems* (pp. 207–220). Berlin, Germany: Springer.

Fuhrer, R. (2016). Reconstructing Western Europe's transport system 1500–1950. In *Proceedings of the 14th T2M International Conference*, Mexico City, Mexico. Paris, France: T2M: Transport, Traffic & Mobility.

Hecht, R., Herold, H., Behnisch, M., & Jehling, M. (2018). Mapping long-term dynamics of population and dwellings based on a multi-temporal analysis of urban morphologies. *ISPRS International Journal of Geo-Information*, *8*(1), 2.

Heitzler, M., Gkonos, C., Tsorlini, A., & Hurni, L. (2018). A modular process to improve the georeferencing of the Siegfried map. *e-Perimetron*, *13*(2), 96–100.

Hori, O., & Okazaki, A. (1992). High quality vectorization based on a generic object model. In H. S. Baird, H. Bunke, & K. Yamamoto (Eds.), *Structured document image analysis* (pp. 325–339). Berlin, Germany: Springer.

Hurni, L., & Hutzler, E. (2008). Implementation kartographischer Funktionen als Adobe Illustrator™-Plugins. *Kartographische Nachrichten: Fachzeitschrift für Geoinformation und Visualisierung*, *58*, 255.

Hurni, L., Lorenz, C., & Oleggini, L. (2013). Cartographic reconstruction of historic settlement development by means of modern geodata. In *Proceedings of the 26th International Cartographic Conference*, Dresden, Germany.

Iosifescu, I., Tsorlini, A., & Hurni, L. (2016). Towards a comprehensive methodology for automatic vectorization of raster historical maps. *e-Perimetron*, *11*(2), 57–76.

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 27.

Kräuchi, N. T., & Tschannen, M. (2015). Ja zur Gewässerrevitalisierung – (k)eine Frage der Fruchtfolgeflächenverluste. *Schweizerische Zeitschrift für Forstwesen*, *166*(4), 213–218.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Leyk, S. (2010). Segmentation of colour layers in historical maps based on hierarchical colour sampling. In J.-M. Ogier, W. Liu, & J. Lladós (Eds.), *Graphics recognition: Achievements, challenges, and evolution* (pp. 231–241). Berlin, Germany: Springer.

Leyk, S., & Boesch, R. (2009). Extracting composite cartographic area features in low-quality maps. *Cartography & Geographic Information Science*, *36*(1), 71–79.

Leyk, S., & Boesch, R. (2010). Colors of the past: Color image segmentation in historical topographic maps based on homogeneity. *GeoInformatica*, *14*(1), 1–21.

Leyk, S., Boesch, R., & Weibel, R. (2006). Saliency and semantic processing: Extracting forest cover from historical topographic maps. *Pattern Recognition*, *39*(5), 953–968.

Li, Z., Wegner, J. D., & Lucchi, A. (2019). Topological map extraction from overhead images. In *Proceedings of the 2019 International Conference on Computer Vision*, Seoul, South Korea. Piscataway, NJ: IEEE.

Liu, Y. (2002). An automation system: Generation of digital map data from pictorial map resources. *Pattern Recognition*, *35*(9), 1973–1987.

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., … Walsh, J. (2020). Deep learning vs. traditional computer vision. In *Proceedings of the 2019 International Computer Vision Conference* (Vol. 1, pp. 128–144). Piscataway, NJ: IEEE.

Pedregosa, P., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional networks for biomedical image segmentation*. Cham, Switzerland: Springer.

Shelhamer, E., Long, J., & Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *39*(4), 640–651.

Stuber, M. B., & Bürgi, M. (2018). *Vom "eroberten Land" zum Renaturierungsprojekt: Geschichte der Feuchtgebiete in der Schweiz seit 1700*. Bern, Switzerland: Haupt Verlag.

Swisstopo. (2019). *Hintergrundinformation zur Siegfriedkarte*. Retrieved from https://www.swisstopo.admin.ch/de/wissen-fakten/karten-und-mehr/historische-kartenwerke/siegfriedkarte.html

Uhl, J. H., Leyk, S., Chiang, Y.-Y., Duan, W., & Knoblock, C. A. (2017). Extracting human settlement footprint from historical topographic map series using context-based machine learning. In *Proceedings of the Eighth International Conference of Pattern Recognition Systems*, Madrid, Spain. Piscataway, NJ: IEEE.

Wall, K., & Danielsson, P.-E. (1984). A fast sequential method for polygonal approximation of digitized curves. *Computer Vision, Graphics, & Image Processing*, *28*(2), 220–227.

Wiesmann, S. (2016). *Gletscheroberflächenentwicklung – Aspekte der Datenakquisition und der Visualisierung*. ETH Zürich, Zürich, Switzerland. Retrieved from https://doi.org/10.3929/ethz-a-010814643

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section.

How to cite this article: Heitzler M, Hurni L. Cartographic reconstruction of building footprints from historical maps: A study on the Swiss Siegfried map. *Transactions in GIS*. 2020;24:442–461. https://doi.org/10.1111/tgis.12610