USING CONVOLUTIONAL NEURAL NETWORKS AND PATTERN MATCHING FOR DIGITIZATION OF PRINTED CIRCUIT DIAGRAMS

LUKAS FUCHS[1], MARC DIESSE[2], MATTHIAS WEBER[1], ARIF RASIM[2], JULIAN FEINAUER[2], VOLKER SCHMIDT[1]

[1] *Institute of Stochastics, Ulm University, 89069 Ulm, Germany*

[2] *pragmatic minds GmbH, 73230 Kirchheim unter Teck, Germany*

Abstract. The maintenance and repair of industrial machinery rely on circuit diagrams, which serve as essential references for troubleshooting and have to be updated when machinery is adapted. However, many circuit diagrams are not available in a proper data structure, i.e., they exist as as unstructured PDF files, rendered images or even photos. Existing methods for digitization often focus on isolated tasks, such as symbol detection, but fail to provide a comprehensive approach. This paper presents a novel pipeline for extracting the underlying graph structures of circuit diagrams, integrating image preprocessing, pattern matching, and graph extraction. A U-net model is employed for noise removal, followed by gray-box pattern matching for device classification and a final graph extraction step to reconstruct circuit connectivity. A detailed error analysis highlights the strengths and limitations of each pipeline component.

Keywords: Circuit diagram; Image processing; Convolutional neural network; Pattern recognition; Distortion analysis; Similarity measure

1. Introduction

The maintenance and repair of machinery are critical tasks in various industries, including manufacturing, transportation, and energy production. These machines internally are described by circuit diagrams, which serve as essential blueprints for understanding and troubleshooting electrical systems [23]. However, the preservation and interpretation of these circuit diagrams pose significant challenges due to their representation and age. There are attempts to introduce standards [3, 5, 18]. However, on the other hand, there is an evolution (or non-existence) of documentation standards. Often, these diagrams are available only as PDF files, with the underlying structure no longer accessible in a more usable format. Sometimes, these files dot not even contain a proper vector graphic, but only scanned images. Despite the fact, that most modern circuit diagrams are being created by means of specialized ECAD (electronic computer-aided design) software, there are significant challenges in leveraging this data. First, many legacy circuit diagrams stored in archives no longer have corresponding ECAD files, making it difficult to digitize them. Second, certain proprietary ECAD software, particularly older systems, may lack convenient interfaces for exporting data, limiting users to printed outputs, which can complicate the digital processing.

Traditional methods of interpreting circuit diagrams are labor-intensive and prone to human errors. Manual transcription and analysis are not only time-consuming but also susceptible to inaccuracies, especially in complex diagrams with extensive interconnections. However, this reconstruction process is crucial not only for troubleshooting, but also for adapting the machine. For instance, when a single device is exchanged, a new circuit diagram must be drawn to reflect the updated configuration. The ability to generate updated diagrams efficiently ensures that machinery can be modified and maintained with minimal disruption.

Several attempts have been made in the literature to address these tasks. In [24, 25], convolutional neural network (CNN) based proof-of-concepts for automatic detection of electrical devices in circuit diagrams are presented, demonstrating high accuracy for predefined symbol classes while identifying challenges with certain line styles. In [20], a Hough transformation is applied for line detection, showcasing its effectiveness but emphasizing the need for preprocessing to handle textual elements and noise. Furthermore, in [9], neural networks are explored for analyzing hand-drawn schematics, highlighting their robustness but also the trade-offs in computational efficiency and interpretability.

The motivation of the present paper is to combine the strengths of these individual approaches while overcoming their shortcomings, by developing a holistic pipeline that can accurately extract the underlying graphs from circuit diagrams, despite present text elements, different line styles and noise. The proposed pipeline consists of image preprocessing, pattern matching, and graph extraction. Initially, a neural network (NN), specifically a U-net [16, 27], is employed to remove unnecessary information such as noise, text, logos, and other extraneous elements from the circuit diagram. Following this, a gray-box pattern matching technique [7] classifies the devices within the diagram and a line detection method identifies wires within the diagram. Finally, a graph extraction approach identifies and extracts a graph-based representation of the circuit diagram, suitable for computing, e.g., connected components [14]. Furthermore, the present paper provides a comprehensive error analysis, showing the benefits and drawbacks of the individual components of the proposed pipeline, which aims to facilitate the digital preservation, analysis, and visualization of these diagrams, thereby supporting maintenance activities and reducing downtime.

The rest of this paper is structured as follows: First, in Section 2, the type of data considered is presented. Then, in Section 3, the graph extraction pipeline is stated, starting with background removal, followed by device and line detection, and lastly graph extraction. Finally, in Section 4, the precision of the individual steps of the pipeline is analyzed.

2. Description of data

For training and calibration of the proposed pipeline, openly available circuit diagram data was used. This training data was assembled from public repositories [2] in the common KiCad schematic format [13], which is widely used in many open hardware projects. KiCad is an open source ECAD software and was employed to plot the schematic circuit diagrams to PDF. These PDFs are vector graphics and thus infinitely resolved;

however, they are not easily accessible for the tasks performed within the pipeline due to their unstructured nature and lack of consistent structural standards in PDF formats [31]. More specifically, PDF files can include invisible lines, and objects observable as a single structural element can be composed of several non-hierarchically structured lines, etc. As a remedy, PDF files are converted into images and the methods described below are applied to these images rather than directly to PDFs. The images contain precisely the information observable for the human eye in a rendered PDF, i.e., the necessary details for accurate graph extraction.

Besides addressing the previous mentioned problems, considering rasterized images additionally enables the use of well-studied convolutional neural networks and pattern matching approaches for the circuit diagram extraction pipeline.

To calibrate the presented pipeline in a supervised setting and to quantify the quality of its results, we compute two binary images $x, y \in \{0,1\}^{m \times n}$ of height $m \in \mathbb{N} = \{1, 2, \ldots\}$ and width $n \in \mathbb{N}$ from each circuit diagram in its vector graphic representation. The image x represents simulated but realistic input data of the pipeline. This is, x includes not only wires and devices but also additional elements such as images, text, and gaps within lines of wires. In contrast, the image y contains only the essential information of the circuit diagram, namely the wires represented by connected lines and devices. The image y allows for the straight forward extraction of information. To generate these pairs of image data (x, y), we utilize a-priori knowledge of the structure of the PDF files generated by KiCad [13]. This is, the PDF files do not contain dashed lines, and text is properly decoded as text, not as individual line segments. This allows us to efficiently compute two versions of the circuit diagram, the image x containing the original text, newly introduced dashed lines and noise, and y containing neither of them. More precisely, to generate a pair $x, y \in \{0,1\}^{m \times n}$ of a raw input image and its corresponding preprocessed image, a given vector-based circuit diagram is processed as follows:

First, to generate x, connected lines are randomly replaced by dashed lines. This modification is applied independently to each line segment longer than 0.5 cm, with a probability of 0.5. The minimum length threshold ensures that small lines within devices, which are essential for preserving their meaning, remain unaltered. The dashed line style (a, b), where a denotes the length of the line segments and b the length of the gap, is chosen uniformly at random from the range $[0.015, 0.15]$ cm. This ensures diverse and realistic variations in line styles. The modified vector graphic is rendered to an image with a resolution uniformly drawn from $[150, 500]$ dpi to account for the large variability of scales and line widths of circuit diagrams in image representation. The rendering is performed as a binary (black and white) image. Formally, the resulting image I from a rendering is an element of $\{0,1\}^{m \times n}$, where m and n denote the dimensions of the image. A value of 0 corresponds to background, whereas a value of 1 corresponds to the foreground, i.e., a line, a device, text, etc. To simulate scanning artifacts, noise is added by randomly inverting each pixel's value with a probability of 0.02. This random process generates the network input x. To generate the corresponding clean image y, the text from the original vector graphic is removed before rendering the graphic at the same resolution. For a visualization of the result of this procedure, see Figure 4. Note that this procedure is random, thus, it is applied to each PDF page several times in order to generate a larger database. More precisely, first, 140 pages of PDF files of circuit diagrams derived from [13] are split into training and test data. Thereby, 90% of these data were selected as training data, in order to calibrate the pipeline. The remaining 10% of the data were set aside and only considered during validation, see Section 4. Then, from each of these PDF pages, 20 image pairs (x, y) are generated as stated above.

3. Methods

This section describes the pipeline of graph extraction from circuit diagrams step by step. First, the image preprocessing procedure is outlined. The goal of this step is to clean the images by removing noise and irrelevant text, which facilitates faster and more robust subsequent processing. Additionally, dotted and dashed lines are replaced by connected lines to simplify line detection. Next, the pipeline identifies the devices in the circuit diagram, follwowed by line detection to identify connections between them. Finally, the extracted information is combined to generate a graph representing the connected devices.

3.1. **Text removal and dashed line connection.** The aim of this section is to introduce a neural network-based procedure to preprocess the image data, enhancing the effectiveness and robustness of subsequent pattern matching and graph extraction tasks. The goal is to produce "clean" images y from input images x that do not contain any elements not important for graph extraction such as, text, noise (appearing in scanned PDFs), or dashed lines. Recall that for an image $x \in \{0,1\}^{n\times m}$, a pixel of value 1 is considered as foreground, and a pixel of value 0 is considered to belong to the background. The neural network's task is to assign a value of 1 to pixels belonging to the actual circuit diagram and a value of 0 to all others. This encodes the removal of unnecessary elements and allows for conversion of dashed lines into continuous lines to improve line detection. This is particularly important because dashed lines can appear in various forms, making them challenging to handle with conventional methods, see Figure 1 for examples.

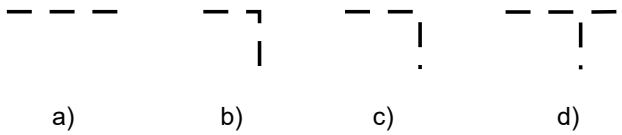


FIGURE 1. Exemplary types of dashed lines. (a) Straight dashed line, (b) dashed line with corner, (c) dashed line with gap at corner, (d) dashed junction with gap at junction.

The neural network $U\colon \{0,1\}^{m\times n} \to [0,1]^{m\times n}$ utilized in this paper is a U-net, which is similar to the network proposed in [27]. It was originally designed for classifying pixels in medical images, but it is now widely used for image segmentation in many different fields [17, 32]. In this context, the task of the present paper can be viewed as a classification problem, where each pixel in x is labeled as foreground or background. In particular, the U-net features a symmetric encoder-decoder structure that captures contextual information through downsampling and recovers spatial details via upsampling. Additionally, skip connections between the encoder and decoder layers allow the network to integrate low-level features with high-level context, improving the precision of pixel-wise outputs while mitigating issues like vanishing gradients. Thus, the neural network U is a high-parametric function, the parameters of which have to be fitted in order to perform the desired task, where the network training is done on the pairs (x, y) of raw and cleaned images introduced in Section 2.

As a loss function for training, the weighted mean squared loss (WMSE) is utilized, i.e., a function $\mathrm{WMSE}\colon \{0,1\}^{3\times m\times n} \to \mathbb{R}$ is used. More precisely, for any pair of training data $x, y \in \{0,1\}^{m\times n}$, and corresponding network output $U(x) = \tilde{y} \in [0,1]^{m\times n}$, the loss $\mathrm{WMSE}(x, y, \tilde{y})$ is given by

$$\mathrm{WMSE}(x, y, \tilde{y}) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (\tilde{y}_{ij} - y_{ij})^2 \cdot (0.01 + x_{ij} + y_{ij}), \tag{1}$$

where $x_{ij}, y_{ij}, \widetilde{y}_{ij}$ wirh $1 \le i \le m, 1 \le j \le n$ refer to the corresponding pixel values of the images $x, y, \tilde{y}$, respectively. Note that WMSE is used instead of the classical mean squared error in order to address the heavily imbalanced class problem, i.e., a large fraction of background pixels, which are almost meaningless for the circuit diagram, and a very low fraction of foreground pixels, which are of high importance. In order to train the network to reliably perform on a wide range of data, and even properly perform on slightly rotated inputs such as those arising from scans, the images x, y are rotated by an angle $\phi = \phi_1 + \phi_2$, where ϕ_1 and ϕ_2 are independently and uniformly drawn from the sets $\{0, 90, 180, 270\}$ and $[-10, 10]$, respectively. To preserve binary values, the so-called nearest-neighbor interpolation is applied during rotation [6].

Training of network parameters is done by minimizing the values of $\mathrm{WMSE}(x, y, \tilde{y})$ given in Eq. (1), using an Adam optimizer [11] with a learning rate of 0.01, where the network architecture is similar to the one described in [27], but with half the number of channels per layer. This is motivated by the rather simple task and the large input images which have to be loaded while training. Thus, by utilizing a reduced network complexity, a memory requirement is achieved that matches that of commercially available graphics cards. This allows for very fast training and application of this method. Note that the prediction of the network

U can take values between 0 and 1. Thus, to achieve a binary clean image during inference, the network output $U(y) \in [0,1]$ is pixel-wise rounded to the nearest integer in $\{0,1\}$.

3.2. **Object detection.** Circuit diagrams prominently feature devices alongside wire connections, see Figure 4a. These device elements range from basic devices, such as resistors, capacitors, and transistors, to more complex digital and analog integrated circuits (ICs), including, for example, microcontrollers, sensors and operational amplifiers. To derive a graph representation from a circuit diagram, the detection of those devices is a crucial step.

One common approach to object detection in computer vision is template matching [10, 12], where a template, that is a small image or a feature representation of the object to be detected, is systematically compared to cutouts taken from all over a given image. The goal is to find pixels in the target image that closely match the template. Often, this type of object detection is performed using neural networks [8] to handle high variations in object scale, rotation, and lighting conditions. However, in the present scenario, which involves simple black-and-white images, a traditional pattern-matching approach is chosen to maintain a fast white-box model with explainable errors.

Fortunately, there are industry standards for symbols used in circuit diagrams for elementary devices, like the American IEEE Standard 315 [4] and IEC 60617 [5], widely used in Europe. Hence, we can draw from a rich library of device templates to detect. Figure 2 depicts some examples.

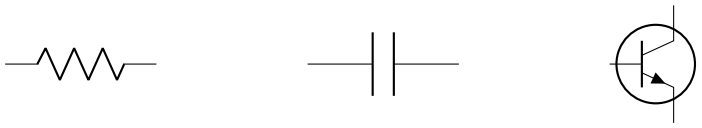


FIGURE 2. Examples of elementary circuit diagram devices as per IEEE Standard 315.

For template matching, a standard method is used which is based on the Pearson correlation coefficient (PCC), [12,30]. The similarity $\mathrm{PCC}(c,z) \in [-1,1]$ of a cutout $c \in \{0,1\}^{k\times l}$ and a template $z \in \{0,1\}^{k\times l}$ is given by

$$\mathrm{PCC}(c,z) = \frac{\sum_{i=1}^{k}\sum_{j=1}^{l}(c_{ij}-\bar{c})(z_{ij}-\bar{z})}{\left(\sum_{i=1}^{k}\sum_{j=1}^{l}(c_{ij}-\bar{c})^2\right)^{\frac{1}{2}}\left(\sum_{i=1}^{k}\sum_{j=1}^{l}(z_{ij}-\bar{z})^2\right)^{\frac{1}{2}}}, \tag{2}$$

where $\bar{c} = \frac{1}{kl}\sum_{i=1}^{k}\sum_{j=1}^{l} c_{ij}$, $\bar{z} = \frac{1}{kl}\sum_{i=1}^{k}\sum_{j=1}^{l} z_{ij}$, and c_{ij}, $z_{ij} \in \{0,1\}$ for $i \in \{1,\dots,k\}, j \in \{1,\dots,l\}$ denote the pixel values of $c, z \in \{0,1\}^{k\times l}$, respectively.

To detect occurrences of a given pattern $z \in \{0,1\}^{k\times l}$ within a binary image $I \in \{0,1\}^{m\times n}$, where $1 \le k < m$ and $1 \le l < n$, the values of $\mathrm{PCC}(c,z)$ are computed using a sliding window approach. More specifically, the sliding window extracts all possible cutouts c of size (k,l) from I, ensuring that the pattern z fits within each position. At each valid location (p,q) in the image, where $1 \le p \le m-k$ and $1 \le q \le n-l$, the corresponding cutout is defined as $c = I_{p:p+k,\,q:q+l} \in \{0,1\}^{k\times l}$, which consists of the pixels from row indices p to $p+k-1$ and column indices q to $q+l-1$. The value of $\mathrm{PCC}(c,z)$ for the cutout c and the pattern z is then used to quantify their similarity. By iterating over all valid positions $(p,q) \in \{1,\dots,m-k\} \times \{1,\dots,n-l\}$ of the cutout $c = I_{p:p+k,\,q:q+l}$, a correlation map is obtained, highlighting regions in I that closely match z. Note that the computation of the PCC across all valid positions can be efficiently implemented using convolution operations.

Recall that the values of $\mathrm{PCC}(c,z)$ belong to the interval $[-1,1]$, where -1 and 1 correspond to a positive and negative linear relation between the cutout c and the pattern z, respectively. Translated to our application on binary images and templates, this means that $\mathrm{PCC}(c,z)$ yields a value of 1 if $z = c$ and a value of

-1 if $z = 1 - c$. Thus, our criterion for detection is a value of $\mathrm{PCC}(c, z)$ higher than some threshold $t > 0$. To avoid false positives, i.e., falsely detected templates, the threshold of $t = 0.8$ was empirically chosen.

Given that multiple instances of the template may appear in the image and that a single template can yield several matches, we apply non-maximum suppression with a low intersection over union (IoU) threshold [26], where the value of $\mathrm{IoU}((p, q), (p', q'))$ for two cutouts (p, q) and (p', q') of size $k \cdot l$ is defined as the ratio of the number of pixels present in both cutouts to the number of pixels present in at least one of them, i.e.,

$$\mathrm{IoU}((p, q), (p', q')) = \frac{\max\{0, k - |p - p'|\} \max\{0, l - |q - q'|\}}{2kl - \max\{0, k - |p - p'|\} \max\{0, l - |q - q'|\}}. \tag{3}$$

Since devices will not overlap in a diagram, we chose a small threshold of 0.2, i.e., for two cutout positions (p, q) and (p', q') of an image I and a template z with $\mathrm{PCC}(I_{p:p+k,\, q:q+l}, z) > \mathrm{PCC}(I_{p':p'+k,\, q':q'+l}, z) > 0.8$ and $\mathrm{IoU}((p, q), (p', q')) < 0.2$, the matched template z at position (p', q') is neglected. If the values of $\mathrm{PCC}(I_{p:p+k,\, q:q+l}, z)$ and $\mathrm{PCC}(I_{p':p'+k,\, q':q'+l}, z)$ given in Eq. (2) are equal, we keep one of both cutout positions. The procedure described above filters the matches, ensuring that we obtain a refined list of detected devices that do not heavily overlap.

Note that the classic pattern matching procedure described above has shortcomings when applied to images that show a lot of noise or arbitrary rotations and resolution, compared to more robust approaches ranging from allowing a cleverly chosen set of template deformations [21] to feature matching methods with [22] or without [12, 28] the assistance of deep learning models. However, since we performed a preprocessing step to remove noise, and deformations in devices and variations in gray scale values do not appear in the considered cases, the simple and fast approach is chosen as described above. Moreover, this procedure has the advantage to provide clear and interpretable errors, as mismatches can be directly traced back to template discrepancies.

The templates utilized to demonstrate the pipeline were sourced from the KiCad symbolic libraries. Specifically, a subset of the most critical devices from the library `Device.kicad_sym` [1] was selected. In the detection pipeline, the templates were up- or downscaled with linear interpolation and scale factors $s \in \{0.7, 0.8, 0.9, 1.0, 1.2\}$. Furthermore, the templates were rotated to four distinct orientations, that is, 0°, 90°, 180° and 270°. To enhance scale robustness additionally, a morphological dilation operation was applied to both the image and templates using a $(3, 3)$ rectangular structuring element [15]. This process increased the thickness of all edges, improving the correlation of shapes on different scales.

Clearly, if the scope of the pipeline are circuit diagrams designed by a different ECAD software, other appropriate templates should be chosen to not lose a significant amount of detection recall, since there could be differences in the device symbol designs in spite of the industry standards. Another notable consideration is the computational cost associated with template matching through correlation and sliding window methods, particularly with high-resolution input images. To address this, it is advisable to limit the number of templates used in the matching process, since the computational cost linearly increases with the number of patterns. Thus, in practical applications, the list of templates used in pattern recognition should be wisely selected based on the specific field of application and relevant industry standards. However, it is important to note that the primary cost here is time, not memory – template matching typically requires modest memory resources and can be executed on standard hardware. As such, if real-time performance is not a priority, expanding the template set remains a viable strategy, particularly given the low risk of false positives when using a similarity threshold of t= 0.8.

For the quantitative analysis of the pipeline performed in Section 4, we selected templates for matching standard bipolar transistors, capacitors, ferrite coils, resistors, ground markings and diodes, which results in a set of 14 original templates and a set of 280 modified, i.e., scaled and rotated, templates.

3.3. **Line detection and clustering.** After extracting the electrical devices from the circuit diagram using the pattern matching method described in Section 3.2 (see also Figure 4), the next step is to identify the wires, which are represented by chains of horizontally or vertically aligned consecutive foreground pixels, called lines in the following. More precisely, for any integer $\ell > 1$, a (horizontal or vertical) line $A = \{a_1, \ldots, a_\ell\} \subset \mathbb{Z}^2 = \{\ldots, -1, 0, 1, \ldots\}^2$ is a set of ℓ pixel positions such that there exists an orientation

$o \in \left\{\binom{1}{0}, \binom{0}{1}\right\}$ satisfying $a_j - a_{j-1} = o$ for all $j \in \{2, \dots, \ell\}$. To detect these sets of pixel positions, a line detection algorithm based on binary morphological operations [15] is applied.

Line detection algorithm. For detecting horizontal lines, we use a horizontal opening operation [15], i.e. any foreground pixel which is not included in the union of all sets of k horizontally consecutive foreground pixels is replaced with a background pixel. For vertical lines, a similar vertical opening is performed, where the number k must be chosen as the pixel length of the shortest line to detect. We set $k = \left\lfloor \frac{m}{160} \right\rfloor$ for horizontal lines and $k = \left\lfloor \frac{n}{160} \right\rfloor$ for vertical lines, where m, n denote width and height of the input image, respectively, and $\lfloor r \rfloor = \max\{\ell \in \mathbb{N} : \ell \leq r\}$ is the largest integer smaller than or equal to $r > 0$. It should be noted that this procedure is again efficiently computable, using convolutions.

Since wires are represented by straight lines in images, identifying the start and end coordinates of them is now straightforward. For cases involving lines that are far from being perfectly straight, a more robust approach would be required. Typically, this involves applying the Hough transform [29] to identify line candidates.

Note that the procedure described above generates duplicated lines if lines within the original image are thicker than one pixel. Furthermore, as a consequence of the preprocessing steps stated in Section 3.1, there can be foreground noise in the neighborhood of the lines, also resulting in superfluous line detections. To counteract this, lines that are encompassed in other lines are removed as a postprocessing step. Here, the term "encompassed" means the following: A line $A \subset \mathbb{Z}^2$ is encompassed in a line $B \subset \mathbb{Z}^2$ if A and B share the same orientation, the size of A is less than or equal to the size of B, and the distance of A and B, i.e., the smallest distance between the pixel positions in A and B is smaller or equal than a threshold of 5.

A further postprocessing step is the removal of lines with a length shorter than or equal to a threshold of 5 pixels.

Classification of line crossings. To derive connectivity information of wires, it remains to cluster the family of detected lines into connected components. For this, it is important to note that there is a variety of junction styles, with different meanings, see Figure 3. Thus, in the following, we employ another (small) CNN in to determine the type of line crossings.

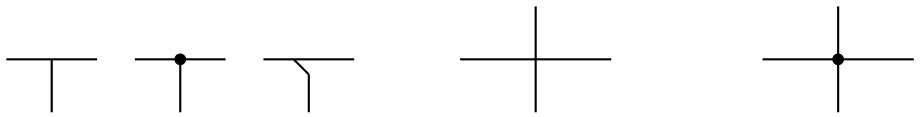


FIGURE 3. Different variants of junctions between wires. Left: The first three images show so-called T-junctions, representing connected wires, using either a straight intersection (without or with dot), or a diagonal connection. For line crossings, two different notations are present in the image data. Middle: Plain crossing of unconnected wires (without dot). Right: crossing of connected wires (with dot).

The utilized CNN is denoted by $S\colon \{0,1\}^{50\times 50} \to [0,1]^{10}$ and processes 50×50 pixel cutouts $c \in \{0,1\}^{50\times 50}$ centered at intersections of lines. More precisely, if the line detection algorithm detects a junction of two lines at pixel $(p, q) \in \mathbb{Z}^2$ in the image I, a cutout $c = I_{p-25:p+25, q-25:q+25}$ is computed. For pixel positions (p, q) which would result in cutouts that exceed the original image I, we apply zero padding [19]. The network S is trained to classify a cutout c to belong to one of 10 different classes, representing the five junctions shown in Figure 3, the four corner junctions arising from the connected version of Figures 1b and 1c (each with rotations by 0°, 90°, 180°, and 270°), and a class representing the case that no junction is present in c.

The architecture of the network $S\colon \{0,1\}^{50\times 50} \to [0,1]^{10}$ can be given as the superposition of simple (almost everywhere) differentiable functions, called layers, i.e.,

$$S = \text{Softmax} \circ \text{FC} \circ \text{Dropout} \circ \text{Flatten} \circ \text{Pool2} \circ \text{ReLU} \circ \text{Conv2} \circ \text{Pool1} \circ \text{ReLU} \circ \text{Conv1}, \tag{4}$$

where the functions appearing on the right-hand side of Eq. (4) will be explained below in detail. Observe that the network S processes an input image $c \in \{0,1\}^{50\times 50}$ and successively transforms it to a vector of length 10, representing the probability that c belongs to the individual classes. For simplicity of the notation of the individual layers, we implicitly assume, that c is a matrix of shape $(50, 50, 1)$ instead of $(50, 50)$.

Layers of the classification network. The convolutional layer Conv1: $\mathbb{R}^{50\times 50\times 1} \to \mathbb{R}^{48\times 48\times 32}$ in Eq. (4) consists of 32 filters of size $3\times 3\times 1$, whereas the convolutional layer Conv2: $\mathbb{R}^{24\times 24\times 32} \to \mathbb{R}^{22\times 22\times 64}$ consists of 64 filters of size $3 \times 3 \times 32$. Both convolutional layers utilize a stride of 1 and no padding. Formally, for any integers $h, w > 2$ and $d, t \geq 1$, a convolution layer Conv: $\mathbb{R}^{h\times w\times d} \to \mathbb{R}^{h-2\times w-2\times t}$ is given by

$$\mathrm{Conv}(X)_{j,k,l} = \sum_{\alpha=0}^{2}\sum_{\beta=0}^{2}\sum_{\gamma=1}^{d} W_{\alpha+1,\beta+1,\gamma,l} X_{j+\alpha,k+\beta,\gamma}, \tag{5}$$

where $X \in \mathbb{R}^{h\times w\times d}$ is some input matrix, $\mathrm{Conv}(X)_{j,k,l}$ denotes the value of $\mathrm{Conv}(X)$ at entry $(j,k,l) \in \{1,\dots,h-2\}\times\{1,\dots,w-2\}\times\{1,\dots,t\}$, and $W \in \mathbb{R}^{3\times 3\times d\times t}$ is a matrix of the trainable parameters. Then, an activation layer follows after each convolutional layer in Eq. (4), which is represented by the activation function ReLU: $\mathbb{R} \to [0,\infty)$ with $\mathrm{ReLU}(r) = \max\{0, r\}$ for each $r \in \mathbb{R}$. The non-linearity of this function allows the network to represent non-linear, complex correlations present in the training data. Furthermore, the max-pooling layers Pool1: $\mathbb{R}^{48\times 48\times 32} \to \mathbb{R}^{24\times 24\times 32}$ and Pool2: $\mathbb{R}^{22\times 22\times 64} \to \mathbb{R}^{11\times 11\times 64}$ in Eq. (4) reduce the spatial dimensions of their inputs. Both functions perform downsampling using a 2×2 region with stride 2, selecting the maximum value within each pooling region. Formally, for any integers $h, w \in \{2, 4, \dots\}$ and $d \geq 1$, a pooling layer Pool: $\mathbb{R}^{h\times w\times d} \to \mathbb{R}^{h/2\times w/2\times d}$ is given by

$$\mathrm{Pool}(X)_{j,k,l} = \max\{\, X_{2j-1,2k-1,l}, X_{2j-1,2k,l}, X_{2j,2k-1,l}, X_{2j,2k,l} \,\}, \tag{6}$$

where $X \in \mathbb{R}^{h\times w\times d}$ and $(j,k,l) \in \{1,\dots,h/2\}\times\{1,\dots,w/2\}\times\{1,\dots,d\}$.

The flatten layer Flatten: $\mathbb{R}^{11\times 11\times 64} \to \mathbb{R}^{11\cdot 11\cdot 64}$ in Eq. (4) reshapes the input matrix into a vector of size $11 \cdot 11 \cdot 64 = 7744$, and the dropout layer Dropout: $\mathbb{R}^{7744} \to \mathbb{R}^{7744}$ introduces regularization to prevent the network S from overfitting, which is given by $\mathrm{Dropout}(X)_j = w_j \cdot X_j$ for any input vector $X \in \mathbb{R}^{7744}$ and $j \in \{1,\dots 7744\}$, where $w_1,\dots,w_{7744} \in \{0,1\}$ are independently sampled realizations of a Bernoulli distributed random variable with parameter 0.5.

The fully connected layer FC: $\mathbb{R}^{7744} \to \mathbb{R}^{10}$ in Eq. (4) is a linear transformation given by $\mathrm{FC}(X) = WX + b$ for any input vector $X \in \mathbb{R}^{7744}$, where $W \in \mathbb{R}^{10\times 7744}$ is a trainable weight matrix and $b \in \mathbb{R}^{10}$ is a trainable bias vector. Finally, for any input vector $X = (X_1,\dots,X_{10}) \in \mathbb{R}^{10}$ and $j \in \{1,\dots,10\}$, the softmax activation layer Softmax: $\mathbb{R}^{10} \to [0,1]^{10}$ in Eq. (4) is defined as

$$\mathrm{Softmax}(X)_j = \frac{e^{X_j}}{\sum_{k=1}^{10} e^{X_k}}. \tag{7}$$

This ensures that the outputs are within the set $[0,1]^{10}$. Further details on the layer architecture of CNNs can be found, e.g., in [19].

Training of network parameters. To train the parameters of the classification network S, i.e. the parameters of the convolutional layers and the linear layer, we generate synthetic training data consisting of pairs (c, g) of image data and junction label. Thereby $c \in \{0,1\}^{50\times 50}$ is a (binary) image of a junction, and $g = (g_1,\dots,g_{10}) \in \{0,1\}^{10}$ represents the corresponding intersection class as one hot encoding, i.e., $g_i = 1$ if and only if the intersection class of c is the i-th class. In total, we generate 500 samples for each of the 9 connection classes by applying random augmentations to the line crossings. More precisely, we apply random translations to the line coordinates by up to 3 pixels and we modify the size of dot markings by uniformly drawing their radius from the interval $[2, 4]$. Furthermore, for all intersection classes we add noise close to foreground by switching any background pixel, having a pixel distance of less than one to foreground, into a foreground pixel with a probability of 0.03. Finally, we generate 1 000 samples for the undefined class by randomly placing rectangles, triangles, and off-center lines to capture ambiguous cases. The network is then trained for multi-class classification using the categorical cross-entropy loss function

$L\colon \{0,1\}^{10} \times (0,1)^{10} \to \mathbb{R}$ given by

$$L(g, S(c)) = -\sum_{i=1}^{10} g_i \log(S(c)_i), \tag{8}$$

for any pair $(c, g) \in \{0,1\}^{50\times 50} \times \{0,1\}^{10}$ of training data, where $\log\colon (0,\infty) \to \mathbb{R}$ is the natural logarithm.

Clustering of detected lines. First, cutouts containing pairs of possibly intersecting lines are determined, which will be classified by the network S given in Eq. (4). Therefore, for each pair (A, B) of detected lines, where A is a horizontal line and B is a vertical line, the pair (a, b) of the closest pixel positions $a \in A$ and $b \in B$ is computed. If the distance between a and b is larger than 25 pixels, the lines A and B are considered as non-intersecting. Otherwise, a cutout c centered at $\mu = \frac{a+b}{2}$ is computed , i.e., c is given by $c = I_{\lfloor\mu_1\rfloor-16:\lfloor\mu_1\rfloor+16,\lfloor\mu_2\rfloor-16:\lfloor\mu_2\rfloor+16}$, where μ_1, μ_2 are the coordinates of $\mu = (\mu_1, \mu_2) \in \mathbb{R}^2$. The network S is then used to determine whether the wires depicted in c are connected or not. For the remainder of the paper, by slight abuse of terminology, the term wire will denote a connected component of lines.

3.4. **Connectivity graph.** In order to suitably represent the set of devices and wires, as well as connections between them, we will use some basic notions from graph theory [14]. That is, we consider a graph $G = (V, E)$ with a (non-empty) set of nodes V containing all detected devices and wires of a circuit diagram. Furthermore, the set $E \subset V \times V$ of edges of the graph contains pairs of nodes, representing connections between devices and wires. More precisely, $(u, v) \in E$ if and only if a wire $u \in V$ and a device $v \in V$ are connected with each other, which means, that a connection-point of the wire, i.e. any endpoint of some line in the component has a distance of less than 5 pixels to a foreground or background pixel of the device. Finally, we delete all detected wires that are not contained in any edge of the graph G, which can be due to false positives in the line detection step.

On the one hand, this representation is low-dimensional and, on the other hand, it allows for the direct application of algorithms from the well-established field of graph theory. As a result, there exist several efficient algorithms for computing properties of graphs that are important in practical applications. For example, for troubleshooting electrical devices, these algorithms can identify all devices that are interconnected or determine the shortest paths connecting them.

Furthermore, for validation purposes, we will utilize graph similarity measures to check the quality of the pipeline proposed in the present paper, since these measures are capable of displaying global graph features, such as connectivity of devices. In particular, in Section 4 below, we will compare graphs generated by hand labeling with those generated by the pipeline presented.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Then, the so-called graph edit distance (GED) is given by

$$\mathrm{GED}(G, G') = \frac{|V \cap V'| + |E \cap E'|}{|V \cup V'| + |E \cup E'|}, \tag{9}$$

where the symbols $\cap, \cup$ are used for the intersection and union of sets, respectively, and $|\cdot|$ denotes cardinality. Intuitively, in our case, the quantity $\mathrm{GED}(G, G')$ given in Eq. (9) describes the fraction of correctly detected wires and devices, as well as connections between these objects. To further investigate where some low precision of the graph detected by our pipeline might come from, we consider the precision $\mathrm{N_p}(G, G')$ and the recall $\mathrm{N_r}(G, G')$ of nodes, which are given by

$$\mathrm{N_p}(G, G') = \frac{|V \cap V'|}{|V'|}, \quad \text{and} \quad \mathrm{N_r}(G, G') = \frac{|V \cap V'|}{|V|}, \tag{10}$$

assuming that $|V|, |V'| > 0$. Moreover, assuming that $|E|, |E'| > 0$. we consider the precision $\mathrm{E_p}(G, G')$ and the recall $\mathrm{E_r}(G, G')$ of edges, which are given by

$$\mathrm{E_p}(G, G') = \frac{|E \cap E'|}{|E'|} \quad \text{and} \quad \mathrm{E_r}(G, G') = \frac{|E \cap E'|}{|E|}. \tag{11}$$

Finally, as a measure of global similarity of two graphs G and G', the numbers of their connected components are compared to each other. A connected component $C \subset V$ of a graph $G = (V, E)$ is a subset of

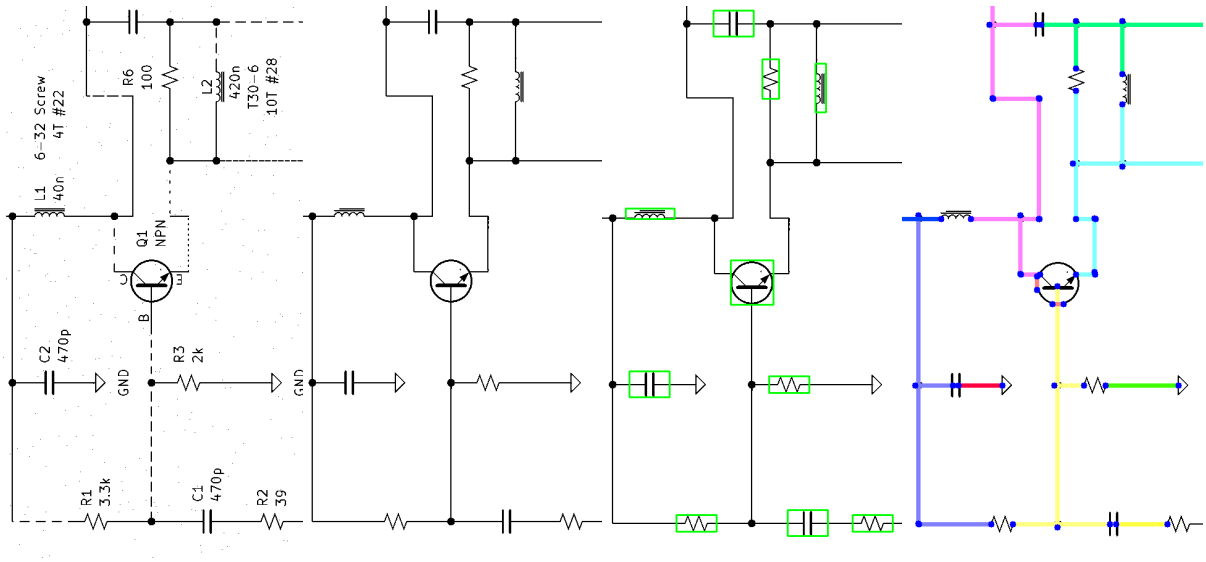
nodes such that for any $u, v \in C$, there exists a path of connected nodes $u_0, u_1, \ldots, u_\ell \in C$ from u to v, i.e., $u = u_0$, $u_\ell = v$ and $(u_i, u_{i+1}) \in E$ for each $i \in \{0, 1, \ldots, \ell - 1\}$, and there is no path between any $u \in C$ and any $v \in V \setminus C$. The number of connected components of a graph G will be denoted by $c(G)$. To compare two graphs G and G', we consider the relative number $\mathrm{NC}(G, G')$ of connected components of G and G', defined as

$$\mathrm{NC}(G, G') = \frac{c(G')}{c(G)}\,. \tag{12}$$

4. Results and discussion

Before quantifying the overall graph extraction quality, the quality of the individual steps considered in Section 3 is analyzed separately.

4.1. **Quality of the U-net output.** Recall that the pipeline starts with a U-net based preprocessing of circuit diagrams in order to remove unnecessary text and noise, and to connect dashed line segments. A visual impression of the results achieved can be obtained from Figure 4, where a high extraction quality can be observed.



(a) (b) (c) (d)

Figure 4. Pipeline overview. (a) Raw circuit diagram image x, (b) corresponding clean ground truth image y, (c) predicted clean circuit diagram $\tilde{y}$ produced by the U-net preprocessing step of Section 3.1, with highlighted devices detected by the template matching algorithm of Section 3.2, (d) same prediction $\tilde{y}$ with highlighted wires, identified using the line detection algorithm of Section 3.3.

To assess the quality of the network output more quantitatively, we analyze the various types of errors individually. This analysis is done on test data, not used for network training. On these data, the U-net stated in Section 3.1 successfully removes 99.8% of black pixels attributed to noise and 99.8% of black pixels associated with text. Additionally, 96.5% of white pixels in dashed lines are accurately replaced by black pixels. Moreover, 100.0% of background pixels remain correctly unmodified. A visualization of these results is given in Figure 5, where it becomes clear that the task of detecting dashed lines is the hardest one for the U-net.

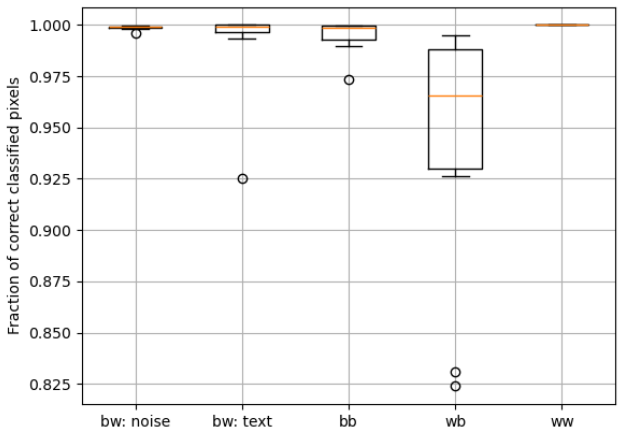


FIGURE 5. Precision per pixel class. The quality of the U-net preprocessing with respect to black pixels that should be set to white (bw), black pixels that should remain unchanged (bb), white pixels that should be set to black (wb), and white background pixels that should not be changed (ww). The classification of black pixels that should be classified as white pixels is further distinguished into pixels belonging to noise (bw: noise) and pixels belonging to text (bw: text). Note that the box plots shown in this figure correspond to results achieved on test data, not used for network training.

However, even though the U-net achieves a high accuracy over all pixels, in circuit diagrams not all pixels are equally important. For example, there are cases where the value of a single pixel determines whether two devices are connected or not. The quality measure considered in Figure 5 cannot reflect this. Thus, as a second quality measure, the resulting correct detection of lines and devices is analyzed.

4.2. **Synthetic distortion of input images.** To assess the precision of the proposed pipeline and further analyze the quality of the pipeline with respect to different qualities of the image data, its performance was analyzed under synthetic distortions, where the distortions included three types: (1) noise of varying magnitudes added to the input image, (2) scaling applied to the input image, and (3) increased gap sizes within dashed lines.

Varying magnitude of noise. For the analysis of the robustness of the noise, the test data were modified by changing each background pixel to a foreground pixel with a certain probability $p \in [0, 0.2]$, while keeping all other parameters (scaling and gap size) constant. Figure 6a presents the resulting pipeline precision. Specifically, the metrics include the mean fraction of correctly classified foreground pixels in the ground truth image (orange), the mean fraction of correctly classified lines in the line detection step (blue), and the mean fraction of devices accurately identified in the pattern matching step (green). To evaluate the number of correctly identified lines and devices, these elements were manually labeled. A detected line was considered correct if its start and endpoint matched those of a labeled line within a tolerance of 10 pixels with respect to the maximum metric. Likewise, a device was classified as correctly detected if the intersection over union between a detected template and the corresponding (pre-labeled) bounding box exceeded the value of 0.8, see Figures 4c and 4d.

Although the pixel-wise classification performance does not show a significant decrease under increasing noise levels, the quality of line detection and device detection decreases significantly at extreme but unrealistic noise levels. This degradation occurs primarily because the noise removal network is trained using a loss function that considers only local (pixel-wise) losses. It does not account for global properties such as line connectivity, which are critical for accurate line and device detection.

Effect of scaling. Similarly, to evaluate the effect of scaling, the same validation image was rendered at different resolutions ranging from 75 to 500 dpi, while keeping other distortion magnitudes unchanged. The results obtained in this case are shown in Figure 6b, where the performance of the pipeline decreases significantly at very low resolutions. For example, at 75 dpi, the pixel-wise error is still low; however, since lines are rendered with a width of just one pixel, device and line detection show a strong decrease in accuracy.

Impact of gap size. Finally, the impact of gap size in dashed lines was studied. The validation image was modified to include dashed lines, where each line had a straight segment of 0.15 cm and gaps of size δ cm for some $\delta > 0$, with a probability of 0.5 for any given line being dashed. The parameter δ was varied within the range $[0\text{cm}, 0.7\text{cm}]$, see Figure 6c for the results obtained in this case. Note that the green line, corresponding to device detection, is omitted here because, as expected, variations of gap size do not have any notable effect on pattern matching quality. It can be observed that until the gap sizes are less than four times the length of the straight segment, the line detection quality remains unaffected.

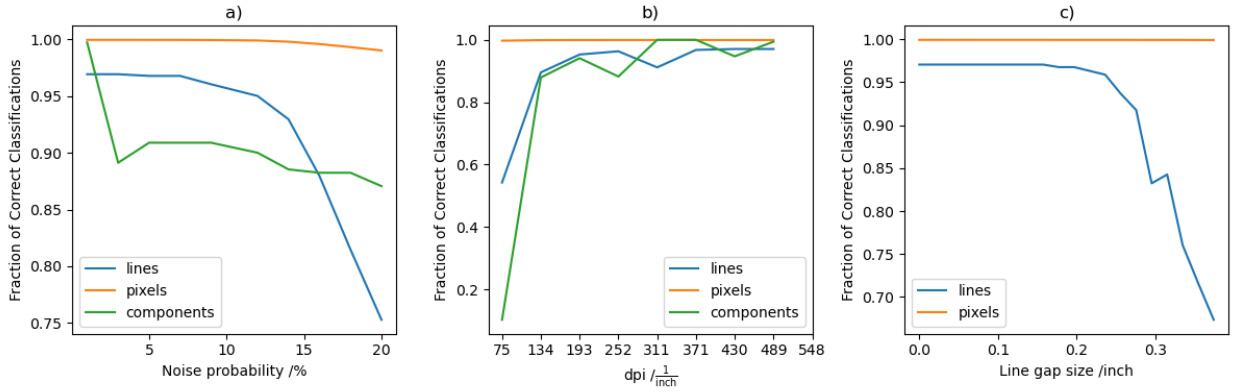


FIGURE 6. Distortion analysis. The influence of different magnitudes of noise (a), resolutions (b) and gap sizes in dashed lines (c) are shown, for the pixel-wise classification (orange), the detection of lines (blue) and the detection of electrical devices (green). Since the gap size of lines does not influence the detection of devices, the respective curve is neglected in (c).

4.3. **Similarity of graphs.** To obtain a quality metric that evaluates more than just the local detection of individual wires and devices, we will consider the similarity of graphs G, G' extracted from the circuit diagram (see Section 3.4), once by hand and once by the presented pipeline. Based on these graph representations, we compute graph similarity measures that capture not only local similarities, such as the number of correctly detected devices and wires, but also global aspects, including correctly detected connections between wires and devices as well as the overall connectivity. Formal definitions of these measures have been provided in Section 3.4. For node comparison similar criteria as described above were used. Two device nodes are considered equal if the center points of the bounding boxes differ by no more than 10 pixel with respect to the maximum metric. Two wire nodes are considered equal if they share a line, where lines are regarded as equal if start and endpoint match within a tolerance of 10 pixels as above.

Figure 7 visualizes graph similarities obtained for the validation image under different levels of added noise (see Figure 6a). The node precision $N_\text{p}(G, G')$ introduced in Eq. (10), which represents the fraction of detected wires and devices in G' that are also present in the ground truth graph G compared to the number of detected wires and devices, remains very high even for noisy images. This indicates that the proposed pipeline rarely detects nonexistent devices or wires. To be more precise, false positives in line detection can happen frequently if the noise level is high but this does not impact the resulting graph, since we can delete isolated wire nodes without losing information. For the node recall $N_\text{r}(G, G')$ given in Eq. (10), it turns out

that $0.9 < N_r(G, G') < 0.98$, which is slightly lower but still high. Recall that $N_r(G, G')$ reflects the ratio of correctly detected wires and devices in G' compared to those in the ground truth graph G and is directly related to the blue and green lines in Figure 6a.

Edge precision $E_p(G, G')$ and edge recall $E_r(G, G')$, introduced in Eq. (11) and visualized in Figure 7a in green and red, respectively, measure the accuracy of detected connections rather than individual wires and devices. Considering these similarity measures is crucial because they give insight into the performance of the connection classifier, as there would be a missing graph connections if a wire node, i.e. a connected component of lines is not fully detected. The results shown in Figure 7a indicate that edge precision $E_p(G, G')$ is extremely high, which means that false connections are rarely detected. However, edge recall $E_r(G, G')$, which is the fraction of ground truth edges correctly identified, declines significantly in the presence of extreme noise. Nevertheless, for reasonable noise levels, the edge recall $E_r(G, G')$ remains above 98%.

Figure 7b illustrates the normalized graph edit distance $\mathrm{GED}(G, G')$ given in Eq. (9), representing the number of modifications required to transform a graph obtained from the proposed pipeline into the corresponding ground truth graph. This measure effectively summarizes the results shown in Figure 7a. The distribution of values of $\mathrm{GED}(G, G')$ reveals that for low noise levels, extreme outliers are absent, demonstrating consistent performance.

Finally, Figure 7c shows the relative number $\mathrm{NC}(G, G')$ of connected components of the extracted graph G' and the ground truth graph G, in dependence of the noise added. Specifically, $\mathrm{NC}(G, G')$ is the normalized number of connected components in G', indicating how much more components G' has compared to G. Note that in all but one case, G' has more connected components than G. For circuit diagrams with less than 9% noise, the number of connected components in G' generally matches that of G. However, at extreme noise in the validation image, connectivity in G' decreases significantly, leading to more disconnected components.

In summary, the proposed pipeline performs well, not only on clean circuit diagrams, such as those directly extracted from PDFs, with an accuracy exceeding 95% across all metrics, but also on images with reasonable distortion levels. Despite being trained only on local features, such as individual noise, wires, and devices, the pipeline achieves a high precision even with respect to global similarity measures like connectivity.

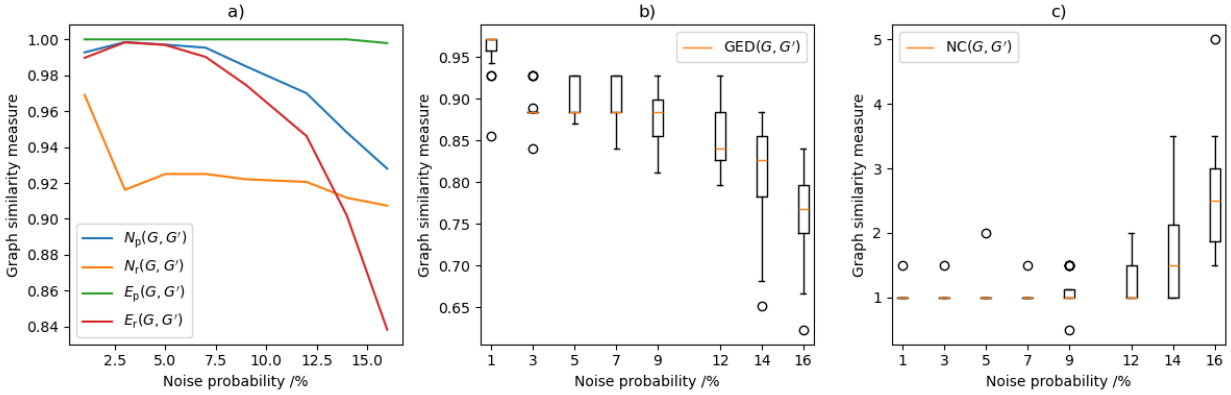


FIGURE 7. Graph similarity measures. Quantitative similarity of graphs G extracted by hand from the validation circuit diagram and graphs G' extracted by the proposed pipeline in dependence of the noise added to the image. The similarity is quantified by means of mean node/edge recall/precision (a), normalized graph edit distance (b), and the relative number of connected components (c). The underlying data corresponds to the validation data of Figure 6a.

5. Conclusion and outlook

A pipeline for automatically extracting graph representations of connected components from circuit diagrams is proposed. The pipeline integrates image processing, pattern-matching-based device detection, and

skeletonization-based line detection. While the methodologies are demonstrated using image data derived from vector graphic-based circuit diagrams, application to image data arising from scanning is straight forward. The method focuses on interpretable methods, especially suitable for fast and large scale applications. On the other hand, the methods presented in this paper, especially the pattern matching, have some drawbacks, when considering rotations. In our future work, we aim to use a combination of edge detection and a light-weighted CNN-based regression to predict and correct small rotations within the image in order to overcome this. Future work will also include exploring an alternative to the current line detection method where, instead of relying on morphological operations to extract vertical and horizontal lines, a more rotation-invariant approach could involve skeletonizing the entire image. Subsequently, foreground pixels with more than two neighbors - referred to as crossing points - would be removed to isolate individual line segments. These segments could then be grouped through clustering techniques. Afterwards, the original crossing points from the non-skeletonized image could be reanalyzed using a crossing classifier to identify connected components. This pipeline would then offer improved robustness to image rotation, although it would likely require a more complex model for accurate classification of the crossing points.

References

[1] Kicad symbols. `https://gitlab.com/kicad/libraries/kicad-symbols/`.

[2] Open source hardware park. `https://oshpark.com/shared_projects/`.

[3] IEEE standard for graphic symbols for electrical and electronics diagrams (including reference designation letters). *IEEE Std 315-1975 (Reaffirmed 1993)*, pages 1–176, 1975.

[4] American national standard - supplement to graphic symbols for electrical and electronics diagrams. *ANSI/IEEE Std 315A-1986*, pages 1–64, 1986.

[5] Graphical symbols for diagrams. *IEC 60617-DB*, 2025.

[6] K. Alomar, H. I. Aysel, and X. Cai. Data augmentation in classification and segmentation: A survey and new strategies. *Journal of Imaging*, 9(2):46, 2023.

[7] A. Apostolico and Z. Galil. *Pattern Matching Algorithms*. Oxford University Press, 1997.

[8] J. K. Basu, D. Bhattacharyya, and T. Kim. Use of artificial neural network in pattern recognition. *International journal of software engineering and its applications*, 4(2), 2010.

[9] J. Bayer, A. K. Roy, and A. Dengel. Instance segmentation based graph extraction for handwritten circuit diagram images. *arXiv preprint arXiv:2301.03155*, 2023.

[10] C. M. Bishop and N. M. Nasrabadi. *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] S. Bock, J. Goppold, and M. Weiß. An improvement of the convergence proof of the adam-optimizer. *arXiv preprint arXiv:1804.10587*, 2018.

[12] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. J. Wiley & Sons, 2009.

[13] J.-P. Charras, F. Tappero, and W. Stambaugh. *KiCad Complete Reference Manual*. 12th Media Services, 2018.

[14] R. Diestel. *Graph Theory*. Springer, 2024.

[15] E. Dougherty. *Mathematical Morphology in Image Processing*. CRC Press, 1992.

[16] G. Du, X. Cao, J. Liang, X. Chen, and Y. Zhan. Medical image segmentation based on U-net: A review. *Journal of Imaging Science & Technology*, 64(2):0710, 2020.

[17] O. Furat, T. Kirstein, T. Leißner, K. Bachmann, J. Gutzmer, U. A. Peuker, and V. Schmidt. Multidimensional characterization of particle morphology and mineralogical composition using CT data and R-vine copulas. *Minerals Engineering*, 206:108520, 2024.

[18] D. J. Garland and F. W. Stainer. *Modern Electronic Maintenance Principles*. Elsevier, 2016.

[19] I. Goodfellow. *Deep Learning*. MIT Press, 2016.

[20] C. R. Kelly and J. M. Cole. Digitizing images of electrical-circuit schematics. *APL Machine Learning*, 2(1):016109, 2024.

[21] H. Y. Kim and S. A. de Araújo. Grayscale template-matching invariant to rotation, scale, translation, brightness and contrast. In D. Mery and L. Rueda, editors, *Advances in Image and Video Technology*, pages 100–113. Springer, 2007.

[22] J. Kim, J. Kim, S. Choi, M. A. Hasan, and C. Kim. Robust template matching using scale-adaptive deep convolutional features. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 708–711. IEEE, 2017.

[23] S. Larick. Functional schematic diagrams. *Proceedings of the IRE*, 34(12):1005–1007, 1946.

[24] X. Li and X. Liu. Optimizing parameter extraction in grid information models based on improved convolutional neural networks. *Electronics*, 13(14):2717, 2024.

[25] S. Mani, M. A. Haddad, D. Constantini, W. Douhard, Q. Li, and L. Poirier. Automatic digitization of engineering diagrams using deep learning and graph search. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 673–679, 2020.

[26] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 850–855, 2006.

[27] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[29] R. Szeliski. *Computer Bision: Algorithms and Applications*. Springer, 2022.

[30] R. E. Woods and R. C. Gonzalez. *Digital Image Processing*. Prentice Hall, 3rd edition, 2021.

[31] Q. Zhang, V. S.-J. Huang, B. Wang, J. Zhang, Z. Wang, H. Liang, S. Wang, M. Lin, C. He, and W. Zhang. Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction. *arXiv preprint arXiv:2410.21169*, 2024.

[32] Z. Zhang, Q. Liu, and Y. Wang. Road extraction by deep residual U-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, 2018.