Legend-Informed Symbol Recognition in Engineering Diagrams with Self-Supervised Learning

Antonia Hain[1] (✉), Simon Gölzhäuser[1], Nicolas Réhault[1], Thomas Brox[2], and Matthias Demant[1]

[1] Fraunhofer Institute for Solar Energy Systems ISE, Germany
{antonia.hain, simon.goelzhaeuser, nicolas.rehault, matthias.demant}@ise.fraunhofer.de
[2] University of Freiburg, Germany

Abstract. Engineering diagrams are vital documents in many industries. Historically stored as image data, conversion of such diagrams into modern formats is required for further use and adaptation. Therefore, research towards automated digitization has gained traction. To recognize symbols in the diagrams, recent studies rely on supervised learning, but large labeled datasets are difficult to acquire in industry settings. In this paper, we present a self-supervised approach towards automated recognition of engineering diagram symbols. We validate the method on diagrams from the building sector, where they are used for technical plant planning, installation, and monitoring. The method makes use of diagram legends, which show prototypical examples of the symbols occurring in the diagram. As the legend entries are unique, they can be used to learn embeddings through contrastive learning for a self-supervised classification of diagram symbols. The method circumvents most of the labeling efforts: all symbols are extracted from the set of diagrams with a symbol region detector trained on a synthetic dataset. Then, we train a symbol encoder by contrasting the symbols found inside the legends with each other. The encoder is subsequently used in a matching procedure that classifies unknown diagram symbols by comparing them to the legend examples. Furthermore, it can recognize when symbols do not appear in the legend at all. Generalizing beyond variations in diagram drawing style, this matching procedure achieves over 80% accuracy. The results demonstrate the potential of legends for engineering diagram digitization without the need to invest in labeled datasets.

Keywords: Engineering Diagram · Building Services · Heating, Ventilation and Air Conditioning · Contrastive Learning

1 Introduction

The operation of buildings, specifically heating and cooling systems, contributes significantly to carbon emissions, causing an estimated 26% of global emissions

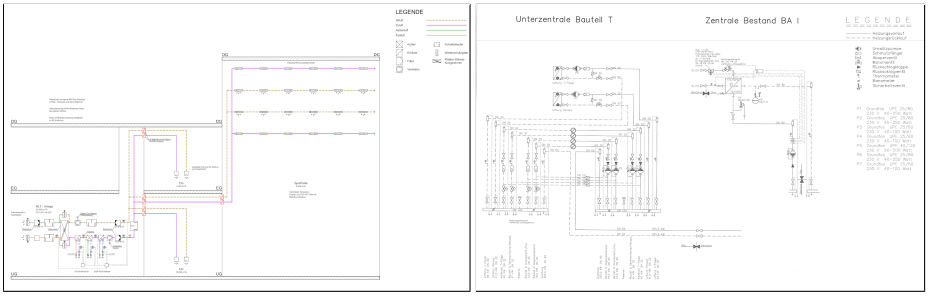


Fig. 1. Exemplary engineering diagrams with different style, coloring and symbols.

related to energy [11]. Therefore, steps towards renovation or operational optimization of heating, ventilation, and air conditioning (HVAC) systems must be taken promptly. However, much effort goes into planning such steps: Before deriving retrofit or optimization measures, existing systems are thoroughly analyzed by reviewing building data and transferring the often decades-old data into a modern CAD format. This entails high workloads for engineers and technicians.

An integral part of building data are engineering diagrams, which illustrate a system's components, using symbols, and its topology. Similar diagrams are used in other industry sectors such as process engineering or electrical engineering. The diagrams may also contain text, tables, or a legend explaining the symbols. Despite existing norms, their style and qualities vary strongly depending on age and source, as shown in Fig. 1. Diagrams may be hand-drawn, scanned or photographed from a print, or provided as a PDF file. In most cases, they therefore do not contain any semantic machine-readable information.

Though considerable advances have been made towards automated recognition and digitization of engineering diagrams, creating a solution that is robust to variation in depiction styles is difficult, especially in regard to diagram symbols. A lack of public and labeled datasets in the domain makes the training of machine learning models for this purpose additionally challenging.

We propose a novel pipeline for the recognition of symbols in engineering diagrams that aims to address these challenges. Instead of attempting to learn all-encompassing representations for a large number of symbols in a supervised manner, we leverage the diagram legends to train a symbol encoder. Our method allows for the recognition of symbols with just one reference example and without the need to predefine classes of symbols to recognize. Furthermore, the method is indifferent to variation in symbol depictions across diagrams, as long as each symbol looks distinguishable from other symbols within the same image.

The approach is a three-step procedure that works without annotations, except for the legend location: First, we generate synthetic data with the aim of representing the gist of what the diagrams look like. Then, we train a generalized symbol detector on the synthetic data, which extracts symbols from the diagrams and corresponding legends. We employ a self-supervised contrastive learning framework to subsequently train an encoder which learns symbol rep-

resentations based on the legends. The representations produced by the encoder are used to classify symbols in the diagrams by determining the nearest-neighbor legend symbol embedding for each diagram symbol embedding. To identify symbols that are not represented in the legend, we define an embedding distance threshold which adapts to each individual diagram.

Summarized, our contributions are:

- A pipeline for legend-based symbol recognition without symbol annotations
- A diagram symbol detection method, trained entirely on synthetic data
- An encoder model computing symbol embeddings, trained with legend symbols and contrastive learning
- A matching procedure selecting corresponding legend symbols for symbols in the diagrams, and rejecting symbols not present in the legend

2 Related Work

2.1 Engineering Diagram Digitization

Digitization of engineering and architectural diagrams, such as, for example, P&ID, chemical process flow and circuit diagrams, or floor plans, has been an active research area since at least the 1990s [19]. In recent years, diagram digitization research has increasingly shifted from hand-crafted features towards neural networks and deep learning. Extensive literature reviews have been comprised by Moreno-García et al. [19] and Jamieson et al. [12].

Alongside a variety of other relevant data, like text and lines, one essential structure to identify in diagrams are symbols, which require a recognition approach specialized to the data at hand. Recently, mostly supervised learning has been applied to locate and recognize diagram symbols, often with popular object detectors [31] [34] [14], and occasionally with segmentation networks [24] or a two-step pipeline, where a localization module is followed by classification [21] [35]. Various studies have demonstrated high precision and recall; however, for a practical application setting, we identify some drawbacks of this approach. The training needs extensive amounts of data, which has to be labeled by experts. In fact, even finding enough and varied data can be a challenge, where possible solutions include mining images from other scientific publications [32], gathering data via web search [31] or simulation with synthetic data [21]. Aside from this roadblock, strong class imbalance has been identified as another issue in training symbol identification models [5].

Few contributions have attempted to classify symbols without a large training dataset. Paliwal et al. [22] represent symbols as graphs and create embeddings of both the graph and the visual representation. Their method recognizes 25 classes of symbols based on just one example each and still performs comparably to other work. One inherent limitation that remains for this method as described, and in general for supervised learning methods, is that all relevant symbols must be predefined. For each class, sufficiently varied samples must be found which cover

the expected distribution, as diagrams in the field show many different variations of the same component types.

While, nowadays, the classification aspect of the symbols is almost always addressed with supervised deep learning, the detection of symbol locations is still sometimes tackled with traditional computer vision techniques [22] [35]. This methodology has the advantage of not requiring large amounts of training data, and in that sense, being unsupervised. However, human-engineered feature extractors have the downside of being vulnerable to data variations that were not explicitly addressed in the design of the method.

Few works explore using the diagram legend for symbol identification. In the domain of map digitization, Samet et al. [28] have studied legends for the identification of symbols with traditional template matching. In the engineering drawing domain, Joy and Mounsef [13] used legend symbols to create training data for an object detector. Sarkar et al. [29] used the legend to classify symbols, testing both SIFT [18] descriptors and a convolutional neural network trained with labeled symbols, in which SIFT achieved better results. Symbol localization was performed with a Faster R-CNN [26] that learned from annotated data. In this work, we also utilize a Faster R-CNN for detection, but trained with synthetic data, bypassing the need for symbol annotation. Similarly, we classify the symbols using the legend, but use a self-supervised learning approach to train a symbol encoder.

Newly, foundation models such as large language models (LLMs) have been on the rise, showing an impressive body of knowledge that can be readily applied to many domains. While we are not aware of any publications using LLMs to digitize complex engineering diagrams, the application of such models may seem intuitive. For this scenario, we however see some challenges: First, many diagram images have thousands of pixels on both axes, but cannot be downsized without rendering small symbols unrecognizable. Therefore, an LLM would have to process a diagram in many small excerpts, which is costly due to the models' complexity. Further, an LLM would need to return accurate symbol locations to allow for subsequent reconstruction of the full diagram, which current models are not specifically trained for. In a pre-test run by us with GPT 4-o [20], we also found that the model's knowledge without finetuning is not sufficient to recognize symbols beyond common types. The model also tends to overlook symbols and to produce false results rather than admitting uncertainty. For these reasons, while engineering diagram recognition with LLMs does not seem out of reach with proper model finetuning and appropriate prompts to address these hindrances, we explore a more lightweight approach in this work.

2.2 Non-supervised Representation Learning

Unsupervised, semi-supervised and self-supervised learning advance steadily, in part motivated by the effort and cost of data labeling. While semi-supervised learning still makes use of a small set of labeled data, unsupervised and self-supervised methods aim at utilizing data without any labels. A popular technique is *representation learning*. The goal of representation learning is to distill data

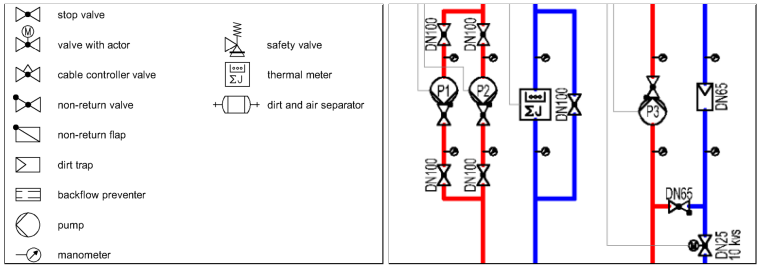


Fig. 2. Legend depicting different kinds of symbols (left) and an excerpt of the diagram in which the legend is placed (right).

into a less complex set of features preserving the key identifiers of each data point, allowing for comparison of data points with simple computations.

Generative models are one popular branch of representation learning which synthesize data similar to a given distribution [23]. To do so, the models implicitly learn to embed the key features of the data, allowing for the use of the embeddings in tasks such as classification. Well-known models include Generative Adversarial Networks (GANs) [6] and autoencoders.

Another field in representation learning is contrastive learning, where the model is instructed on which points in the data are similar and which are different. The data points are passed through a twin network [16] of identical architecture and weights. Similar data points then ought to produce similar embeddings, and dissimilar points different embeddings. This is enforced by, for example, using a contrastive loss [7] or triplet loss [30], which we also use in our approach. Without labels, samples "similar" to a reference data point can be created with augmentation. In recent years, the contrastive learning framework has been adapted for more complex and data-intensive scenarios, e.g. through facilitating the use of many dissimilar pairings with momentum contrast [9] and memory banks [33] or by embedding the input images in a patch-wise manner [2] [8]. For industrial anomaly detection, it has also been shown that patch embeddings can be taken from intermediate layers of pre-trained networks, presenting an alternative to training a specialized network on labeled domain data [27].

3 Legend-Informed Symbol Recognition

3.1 Properties of Engineering Diagrams and Legends

Legends are present in many engineering diagrams and provide useful information regarding the component symbols in the diagram. As seen in the examples in Fig. 1, legends are table-like structures that are usually provided on the side of the diagrams. For simplicity, we will call symbols found inside the bounds of the legend *legend symbols*, and the symbols outside the legend *diagram symbols*.

Fig. 2 shows an exemplary legend and an excerpt of the corresponding diagram, taken from the same image file. Within one diagram, legend symbols often

share similarities in drawing style and coloring. One legend may contain multiple visually similar symbols showing variations of the same component.

From the excerpt, it is evident that matching the diagram symbols to the legend symbols is not trivial: Although the legend contains all symbols in the excerpt and is intuitively usable to human readers, the diagram symbols are not drawn in the exact same way as they appear in the legend. Aside from rotation and scaling differences, outlines may be thicker, thinner, or blurry, and the appearance is additionally changed by the lines connecting the symbols. This challenges more traditional computer vision methods like template matching. As the symbols contain a lot of whitespace and only a few lines, we are also wary of using autoencoders or patch embeddings - even minor image transformations, such as translation or scaling, could be punished strongly in a classical autoencoder setting, and small patches of the images may not always be meaningful.

However, we observe the following properties of legends that make contrastive learning an interesting option for this data:

(1) Each symbol is unique within the legend
(2) Diagram symbols (mostly) appear in the legend

Property (1) can be exploited to mine positive and negative pairings out of the given data, and property (2) can subsequently be used to find labels for the diagram symbols. It is important to note that while, in our observation, property (1) is nearly always true, there are exceptions to (2). Real-life data is imperfect; many legends are indeed incomplete and additional objects appear in the diagrams. However, we find that most legends do represent the majority of diagram symbols, such that applying the method will nevertheless considerably decrease the effort needed for digitizing the documents.

3.2 Method Overview

The proposed legend-informed symbol recognition pipeline is summarized in Fig. 3 and consists of three main parts: First, synthetic data emulating the symbols and diagrams is created using simple drawing functions and randomization. Then, symbols in the diagrams are detected using a generalized symbol region detector, i.e. an object detection neural network that is indifferent to the specific classes of the symbols. After symbol detection, an embedding encoder is trained on the legend symbols with contrastive learning. Finally, we compute embeddings for all symbols with this encoder, which enables for comparison of diagram symbols and the legend counterparts.

Crucially, the method can be trained with minimal annotation effort: The symbol detector is trained entirely on synthetic data, whose creation requires only superficial knowledge of the diagrams' appearance. The embedding encoder also needs knowledge of only two things: (1) the locations of all symbols in the diagram - which can be determined using the detector - and (2) the location of the legend itself, to group the symbols into diagram and legend symbols. Therefore, the only annotations we use are those of the legend locations. Even

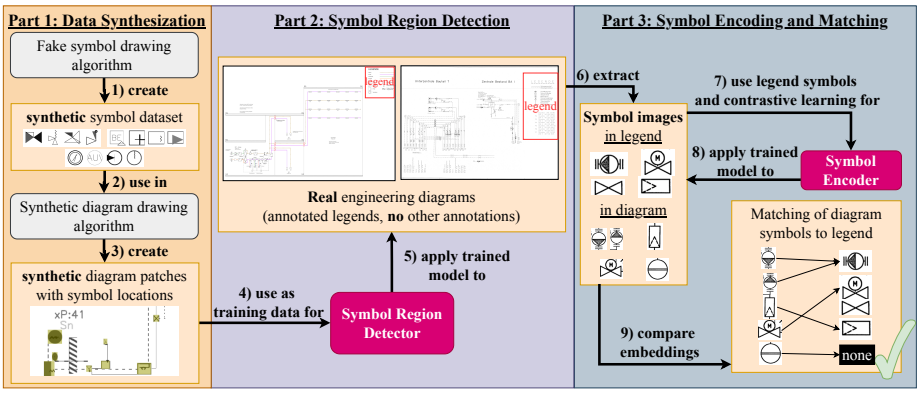


Fig. 3. Overview of the pipeline. Data is shown in orange, basic algorithms in gray and the neural networks in pink boxes. The method's steps are written along the arrows.

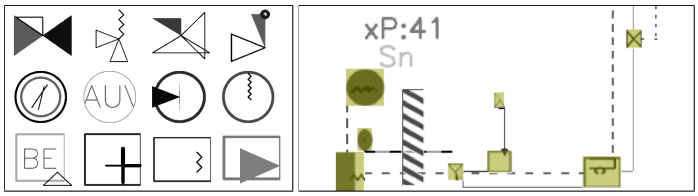


Fig. 4. Synthetic data symbols (left) and an exemplary synthetic diagram excerpt with symbols highlighted in yellow (right).

this can likely be determined automatically, e.g. with a table detection method. Given the low effort needed to point out the legend and the immediate tradeoff in method stability, we however assume here that the legend location is known.

3.3 Synthetic Data Creation

Symbol Creation: For the symbol detector training, we first create a set of symbol-like illustrations using randomized drawing algorithms. The synthetic data is designed to simulate various component types without representing specific symbols. Rather than focusing on the details, the detector is encouraged to recognize what symbols generally look like and in which contexts they appear. Examples can be viewed in Fig. 4 (left). Each image follows one of three basic shape types which imitate the symbol shapes typically seen in the diagrams. This main shape is drawn first and is either a combination of triangles (top row), a circle (middle row) or a rectangle (bottom row). Straight and zig-zag lines or other basic geometric shapes are randomly added to the symbols. Circle and rectangle components may also include letters. Parameters like line strength, gray value, font scale and size are all randomized within pre-set bounds.

Diagram Creation: A diagram is synthesized by placing a number of synthetic symbols on an invisible grid on a square canvas. We then connect the bounding boxes of random, but most often spatially close, symbols with lines of different thicknesses and shades, some of which are dashed like in real diagrams. Additionally, randomly generated rectangles, some with cross-hatching, and random text fragments are placed on the canvas to simulate other common parts of the diagrams to be ignored by the detector. Fig 4 (right) shows an exemplary synthetic diagram excerpt.

3.4 Symbol Localization

To find the symbols, we use an off-the-shelf object detector, such as Faster R-CNN [26] or a detector from the YOLO series [25]. The network only knows one foreground class and is trained exclusively on the synthetic diagrams. During inference on real data, we first detect text elements in the diagram with the CRAFT text detector [1] and remove them. Due to the large size of the diagrams, the networks cannot process them as one. Instead, we employ a sliding window approach, in which square excerpts are taken from the diagram and evaluated by the detectors one by one. Since the diagrams often contain a lot of whitespace, the approach skips excerpts which contain only one unique gray value, and as such appear empty, to avoid unnecessary forward passes and speed up processing.

3.5 Diagram-to-Legend Symbol Matching

To learn an embedding function which can be used to match diagram symbols to the legend symbols, we employ a contrastive learning framework using the legend, and a lightweight convolutional encoder, described in the following subsections.

Contrastive Learning on Legend Symbols: Property (1) as given in Section 3.1 is the backbone for the training of our method. Knowing that legend symbols are unique allows for the design of a self-supervised method which learns to find similarities between different augmentations of the same legend symbol, and differences between one legend symbol and others. Specifically, we use a triplet loss [30], in which each triplet is drawn from the legends in the dataset. The loss function is

$$L(a, p, n) = \max(0, d(a, p) - d(a, n) + m) \tag{1}$$

where a denotes the embedding of an anchor symbol, p and n respectively denote embeddings of "positive"/"negative" symbols that match/do not match the anchor, $d(x, y)$ refers to the Euclidean distance between two embeddings x and y, and m is a predefined margin that is sought to be enforced between the distances of positive pairs and negative pairs.

To form a triplet, a random legend symbol is retrieved from the dataset to serve as the anchor, then augmented and embedded using the encoder. The same symbol with different augmentations is used for the positive, matching,

symbol. For the negative example, another symbol from the same legend is taken and augmented. As we are dealing with data points that would be considered highly similar in a broader image classification task, the encoder must learn to distinguish small details. Therefore, instead of retrieving the negative symbol randomly, we first embed all other symbols within the anchor's legend, and then pick a random symbol within the three closest matches. This is to ensure that the algorithm drives apart the most similar symbols, without getting stuck contrasting against the same neighbor over and over.

Augmentations are crucial for successful contrastive training [3]. Our approach particularly relies on extensive augmentation, since the anchor and the positive sample are based on the same symbol. To reflect all the potential differences in depiction between legend and diagram symbols described in Section 3.1, a variety of augmentations are randomly applied, such as rotation, erosion and dilation, padding and cropping, addition of lines, color shade changes, and noise.

Encoder Network: The encoder is a lightweight neural network with three convolutional layers (using a 7×7 kernel in the first layer and 3×3 kernels after), with ReLU activation functions and 2D Max Pooling placed in between. The block of convolutional layers is followed by a linear layer, which is then normalized to a unit vector, becoming the symbol's embedding. The normalization step ensures that the maximal distance between two embeddings is known and a reasonable margin for the triplet loss function can be chosen.

Matching Procedure: Diagram symbols are matched to the legend symbols with the smallest Euclidean distance in the embedding space. To recognize false-positive symbol detections and symbols not represented in the legend, we add a threshold to our embedding matching method. If no legend symbol embedding is within this distance, the diagram symbol is rejected. Depending on diagram style and quality, the legend symbols may be more or less similar to the diagram symbols. To account for this, the threshold t adapts to each diagram, s.t.

$$t = m + \min_{q \in Q; l \in L} (d(q, l)) \tag{2}$$

where m is the margin used in the triplet loss training, $d(q, l)$ is the Euclidean distance between symbol embeddings q and l, Q is the set of diagram (query) symbols embeddings, and L is the set of legend symbol embeddings within the diagram. In other words, the threshold is the sum of the desired minimal embedding distance between different symbols, and the smallest distance between any diagram and legend symbol embeddings in the respective diagram.

4 Experiments

4.1 Experimental Setup

Data: Our dataset consists of 141 diagrams, acquired from industrial partners[3]. The diagrams were converted from PDF format to grayscale images with 300 dpi to preserve detail. This results in image sizes of around 2300-7900 pixels on the smaller axis and 3300-27,500 pixels on the larger axis, with a median of around 30 MP in total. The diagrams contain five to 800 symbols, with a median of 105.

A test set of 21 diagrams was created to reflect diagrams that are likely encountered in daily use. Due to the sliding window processing, this amounted to over 8000 input patches for the symbol detection, excluding the ignored white-space excerpts.

We selected the test diagrams manually to ensure that various drawing styles, symbols, qualities and complexities were covered. Some diagrams have the same source and were thus drawn with similar templates. The full dataset contains around 20 different drawing styles, but is imbalanced regarding their prevalence. For example, one template was highly common with 57% of diagrams drawn in this style, while other styles only occurred once or twice. Random test data selection would therefore likely favor styles that happened to be common in our data, while not every dataset may contain such imbalances. Note that two diagrams being drawn with the same template does not necessarily entail that the set of symbols used in both diagrams or their legends is the same.

We evaluate multiple scenarios designed to reflect different use cases: (1) As the symbol matching is self-supervised, we test its effectiveness on the data it was bootstrapped on, by testing on seven diagrams whose legends were also used in training. This models a case in which the method is trained to digitize one set of diagrams without considering future use on unseen data. (2) Another third of the test diagrams was excluded from training, but follows similar templates as training diagrams. This models a case where the symbol recognition method is already trained and a user wants to apply it on new incoming data made with the same software or templates as training diagrams. (3) The final third of the test set were neither used in training, nor drawn with a known template, therefore representing a case where a user wants to use the pre-trained method on entirely new incoming diagrams, e.g., from a different building project, made by a collaborator who uses a different template, or which are hand-drawn.

In the human annotations created with the VGG Image Annotator [4] for evaluation purposes, the test diagram legends contain three to 35 symbols, the diagrams themselves between 34 and 484 symbols. Note that in the full pipeline, with bounding boxes generated by the detector, these values may differ due to detection errors.

Detector: As the symbol region detector, we use a Faster R-CNN with MobileNetV3-Large [10] backbone, pre-trained on the COCO dataset [17]. 10,000 fake

[3] Due to industrial restrictions, the dataset and code for this project can not be published at this time.

symbols were created for each of the three types - triangles, circle and rectangle - as basis for the synthetic diagrams. The synthetic diagrams are created on-the-fly during training. We use around 34,000 synthetic samples with a batch size of 16, therefore training for around 2125 steps. We train with Adam Optimizer [15] and evaluate precision and recall.

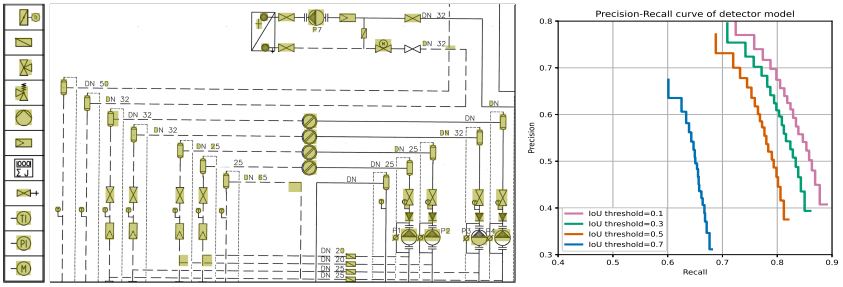
Encoder and Matching: The encoder transforms 64×64 grayscale images into a 15-dimensional embedding vector. We parameterize the triplet loss with margin $m = 0.5$ and train with Adam Optimizer and a batch size of 64. Training stops when the symbol-to-legend assignments in the test set have changed less than 3% per epoch, averaged over the last 50 epochs. For evaluation, we build a confusion matrix of all symbol-to-legend assignments. Symbols not appearing in the legend, which are therefore not to be matched, are classified as a special symbol type which we call *rejects*, and considered separately in the statistics. The results are quantified in classification accuracy, precision and recall, both over the entire test dataset and averaged over the individual diagrams. To separate the performance of the symbol matching procedure from that of the detector, our evaluation considers two cases: One, where near-perfect detections are assumed, using manually annotated bounding boxes, and one for the entire pipeline, using detections made by the object detector.

The results are compared to those of a SIFT matching procedure implemented as proposed in [29]. This baseline approach matches symbols by computing a similarity score between 0 and 1 for each potential diagram-legend match. We enable for rejections by setting a threshold for this similarity. Here, we assume a hypothetical optimal thresholding method, and use the threshold yielding the best accuracy for each individual diagram, which we determine experimentally. For an additional baseline comparison that is not influenced by the rejections, we ignore all reject symbols, remove the rejection mechanisms and report the symbol matching accuracy in the absence of a negative class.

4.2 Experimental Results

Detector: Qualitative and quantitative detection results can be viewed in Fig. 5. The detector achieves up to around 72% precision and recall when the necessary IoU to match a detected box to the ground truth is 0.5. The precision-recall curves show that the metrics are strongly affected by changes in this threshold. False positive detections typically occur at text fragments that were not found by the text detector, and line crossings. Symbols that are not detected are most often very large or small, or drawn in a style that differs from the average symbol, for example more intricate drawings with little whitespace.

Encoder and Matching: Table 1 shows quantitative results of our symbol encoding and the matching algorithm, and of the baseline SIFT approach, on the full test set. With accurate symbol detections, sourced from human annotations, our procedure for matching/rejection achieves an overall accuracy of 83.7%, with



(a) (b) (c)

Fig. 5. Symbol region detection results, highlighted in yellow, on (a) a legend and (b) a diagram excerpt (a text detector is used to remove text beforehand, but the original image is displayed here for readibility), and (c) the precision-recall curves on the test set. Each curve shows the precision and recall values modulated by the confidence threshold that is used to filter detections. The diagram shows curves for multiple Intersection over Union (IoU) thresholds, influencing which boxes are considered a match to a ground truth box.

precision and recall of 94.6% and 82.7% for the symbols which are represented in the legend. The precision and recall of rejects are 59.3% and 87.4%. Notably, precision is therefore much higher for symbols listed in the legends compared to the reject symbols. Averaged over the diagrams instead of over all symbols, the numbers are similar, but the sample standard deviations indicate strong differences across the diagrams: On some diagrams, near-perfect accuracy is reached, and around 70% on others. Our method thereby considerably outperforms the SIFT approach on all markers with this data, achieving over 20pp higher accuracy.

Using the symbol detections of our object detector, the overall accuracy is slightly lower at 80.9%. In this scenario, the precision discrepancy between the true positive symbols and rejects was much smaller. Upon visual inspection, this is because false positive symbol detections - which are not present in the human annotations - are rejected very reliably, raising the average performance on the reject group of symbols. However, both precision and recall for the symbols listed in the legend are about 10% lower than in the reference case, and detections with inaccurate bounding boxes can cause errors. Our proposed method again achieves higher scores than the SIFT method, with over 18pp higher accuracy. A visual result of the entire inference pipeline with both symbol detection and matching is displayed in Fig. 6.

If the need for rejections is removed, our method correctly matches 93.3% (annotation bounding boxes) and 86.0% (detector bounding boxes) of the symbols, more than SIFT with 66.4% and 56.3%, respectively.

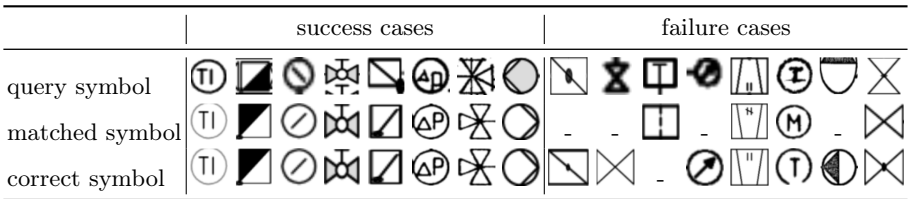
Table 2 shows exemplary success and failure cases. Evidently, the symbols are rarely confused with each other and most errors stem from the rejection mechanism. For example, symbols are falsely rejected if their diagram representation diverges strongly from the legend representation, e.g. due to lower quality or alterations in drawing style. Outside of the legend, most symbol types are

Table 1. Quantitative symbol matching results with our method and a baseline SIFT approach. Column 1 states whether the symbol bounding boxes used for training and evaluation were sourced from human annotations or from the object detector. Column 2 states whether the metrics were averaged over all symbol types in the test set, or averaged over the diagrams, and whether rejects were left out from the statistic.
Acc. = accuracy, P_S/R_S = precision/recall of symbols matching one of the legend symbols, P_R/R_R = precision/recall of rejects.

BBoxes	**Statistic**	*Acc.*	P_S	R_S	P_R	R_R
Proposed Method						
Annotation	overall	**83.7%**	**94.6%**	**82.7%**	**59.3%**	**87.4%**
Annotation	diagram average (±stddev)	**83.4%** (±10.5pp)	**95.2%** (±6.2pp)	**81.9%** (±11.5pp)	**57.0%** (±29.1pp)	**83.7%** (±21.4pp)
Detector	overall	**80.9%**	**85.1%**	**71.4%**	**77.5%**	**91.8%**
Detector	diagram average (±stddev)	**81.0%** (±8.2pp)	**84.4%** (±11.2pp)	**71.4%** (±12.4pp)	**77.3%** (±12.9pp)	**90.2%** (±12.5pp)
Annotation	overall, no rejects	**93.3%**	-	-	-	-
Detector	overall, no rejects	**86.0%**	-	-	-	-
SIFT-based Method [29] (Baseline)						
Annotation	overall	62.0%	63.8%	62.7%	55.7%	59.4%
Annotation	diagram average (±stddev)	63.2% (±15.6pp)	67.6% (±16.4pp)	59.8% (±22.3pp)	58.5% (±24.3pp)	52.4% (±32.4pp)
Detector	overall	61.7%	50.3%	44.0%	71.8%	82.2%
Detector	diagram average (±stddev)	62.3% (±16.0pp)	52.7% (±18.5pp)	42.8% (±25.2pp)	71.0% (±16.8pp)	82.6% (±11.6pp)
Annotation	overall, no rejects	66.4%	-	-	-	-
Detector	overall, no rejects	56.3%	-	-	-	-

represented with coherent drawing styles, such that in this case, a number of instances of the same symbol type may be misclassified. Therefore, errors are not evenly distributed across symbols, but focus on specific types. Conversely, reject symbols are most likely falsely matched if a similar symbol is listed in the legend. However, the perceived general intra-diagram symbol similarity did not inherently cause issues: For example, one diagram contained exclusively rectangular symbols with little variation, which were matched with 97% accuracy. Regarding the three potential use cases reflected in the test data as described in Section 4.1, we do not observe a clear trend outside the inter-diagram standard deviation: On diagrams also seen in training (case 1), the method achieved an average accuracy of 82.1% / 83.8% using the annotated/detector bounding boxes, respectively. On diagrams with similar templates to those seen in train-

Table 2. Exemplary symbol matching success and failure cases. Each column, top to bottom, shows a query (diagram) symbol, the legend match chosen by our method, and the correct match in the legend. "-" indicate rejections, where no legend symbol was found within threshold distance / the symbol type was not listed in the legend.



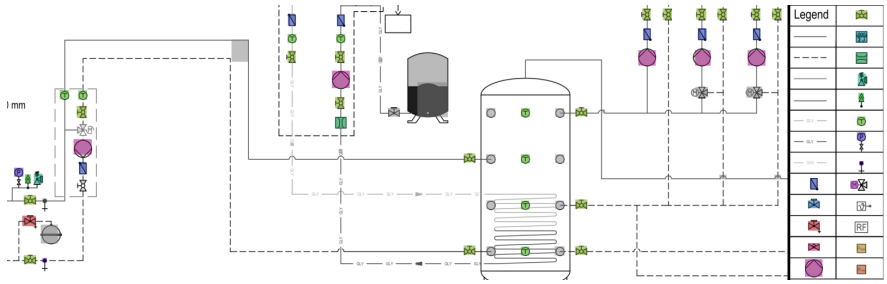


Fig. 6. Result of entire detection and matching pipeline on a diagram excerpt and the corresponding legend. Diagram symbols are marked in the color of the matched legend symbols. Gray boxes indicate detections which were not matched to any legend symbol.

ing (case 2), accuracy was 87.5% / 79.2%. On test diagrams with a style fully unknown to the model (case 3), accuracy was 80.5% / 78.7%.

5 Discussion

Our symbol detector demonstrates promising results on complex data, despite having been trained only on synthetic data that was created based on superficial observation of the diagrams' properties. However, the bounding box locations appear to be inaccurate, as seen by the influence of the IoU threshold on the precision-recall curve. Detecting and removing text elements was essential, as the symbol detector did not reliably ignore leftover text, despite seeing text in the training data. Failure to detect very large or small symbols may be explained by limitations of the object detector or the sliding window detection method. However, the detector also struggled with symbols of uncommon style, indicating that it may rely on the look of the symbols rather than their context, such as lines. On the other hand, legend symbols, which appear without this context, were therefore detected as reliably as the diagram symbols. This is important because missing a legend symbol could lead to many diagram symbols not being matched. To this end, high detection recall is also preferred over precision, since

false positives can still be removed in the matching process. In general, our symbol detection strategy serves as a proof of concept for a scenario where labeled data cannot be obtained at all. Notable improvements are likely possible if, for example, datasets similar to the application area are available, the data is less diverse, or labeling a small amount of data is feasible. Finetuning of the text detection component to technical documentation texts and diagram data could also be explored to improve text removal results and, consequently, those of the following pipeline steps.

The symbol-to-legend matching trained with contrastive learning correctly assigns a majority of symbols with little confusion between symbols on both the diagrams used in training and diagrams that were not available at training time, where we observed no clear trend comparing various test scenarios. On our data, the method outperforms a SIFT approach as used in previous literature addressing the challenge of recognizing diagram symbols with a legend in a non-supervised manner. In many difficult cases, such as low-quality symbols, hand-drawn symbols or accidentally cropped symbols due to bounding box detection errors, the correct match is still found with sufficient confidence. If those difficulties are even more pronounced, errors are however more likely to occur. One thing to keep in mind is that this promising performance is likely dependent on the number and diversity of diagrams available for the initial training.

We find that the main challenge is identifying rejects, i.e. symbols not listed in the legend, without causing too many false rejections in the process. As seen in the results of an experiment case where the rejects are removed, over 90% accuracy may be possible when the rejection mechanism becomes unnecessary. While in our scenario, our thresholding method has shown good success, the results varied strongly depending on the diagram and factors such as the ratio between the amount of symbols represented in the legend and those which were not. Overall, the method tended to reject more symbols than necessary.

We also find that the matching and rejection procedure should be able to adapt to each diagram and that a universal threshold is likely unsuited: Dependent on factors such as the difference between the legend depictions and those used in the actual diagrams, the actual embedding distances varied and did not always adhere to the triplet loss margin.

An inherent challenge is posed by compound symbols. In our data, this kind of symbol is not consistently depicted in the legends: Sometimes, a compound symbol is given its own legend explanation; an example can be seen in Fig. 2, where the symbol *valve with actor* is a compound of the *stop valve* and a motor symbol. In other cases, often even within the same legend, compound symbols are explained only through their separate building blocks. As a result, the same building block may appear in a legend multiple times, softening the uniqueness property of the legend symbols. However, we find that our data did not contain enough such cases to strongly disturb the training. For these reasons, we decide against attempting to merge compound symbols automatically, which the detector usually identifies as separate entities. Having the user indicate which symbols

should be merged after the recognition process would be a practical solution to this challenge and can likely be done with reasonable effort.

Overall, the results show that the method can be highly beneficial for users in the field despite the mentioned shortcomings. Manual labeling of the diagram symbols is tedious, error-prone, and takes twenty to thirty minutes per diagram in our data, while our method processed the test diagrams in an average of less than fifteen seconds on a machine equipped with a NVIDIA Quadro RTX 8000 (48GB) GPU. If the legend is highly incomplete or there is none at all, the encoder embeddings could instead be used to cluster symbols and, thereby, recognize which symbols show the same component. In a practical setting, a user could be presented with each recognized symbol along with the corresponding original excerpts of the diagram image, which can facilitate cross-checking of the results and correction of errors. The user could also manually add missing symbols to the legend, which are likely common symbols known to experts in the field, and may have therefore been omitted. Because the encoder is very lightweight, the matching could be recalculated with an adapted legend almost instantly. Another possible avenue for expansion and improvement of the method may be to subsequently integrate an adequate human-in-the-loop approach which finetunes the recognition based on such user feedback.

6 Conclusion and Outlook

We presented a pipeline for automated symbol recognition in engineering diagrams, based on diagram symbol legends. Our method needs minimal labeling and delivered promising results on real diagrams from the building sector, where data is often proprietary and labeling is costly. We found that most symbols can be localized with a model trained only on synthetic data, and that the uniqueness of symbols in a legend can be utilized for a contrastive learning approach for symbol embeddings. This subsequently enabled us to identify symbols by matching them to the legend. The approach successfully recognized symbols on test diagrams even of unseen drawing style, and is not limited to a predefined set of symbol classes. For a practical setting, this means that the method could be applied to new diagrams outside of the distribution available for training. Therefore, we demonstrated the potential to achieve a notable decrease in the effort needed for diagram digitization. Future research could explore alternative symbol region detection strategies or symbol matching algorithms, with a focus on reliable separation between the symbols represented in the legend from those which are not. In practice, the method will be validated by HVAC engineers in a web-based application that we developed, providing functionalities to digitize, modify, and complete diagrams.

Although we designed our method for the building sector, similar problems could conceivably be addressed in other fields with a similar approach. We therefore believe that the methodology can be valuable in any field facing high workload with digitization of schematics and diagrams, if the data contains legends.

Acknowledgments. This study is funded by a PhD scholarship granted to the first author by the German Federal Environmental Foundation (DBU), and by the German Federal Ministry for Economic Affairs and Climate Action through the project DiMASH (FKZ 03EN1067 A). We thank Serdar Yilmaz of Scherr+Klimke AG as well as Simon Glatz and Niklas Broghammer of Maurer Energie- und Ingenieurleistungen GmbH & Co. KG for provisioning the engineering diagrams used in this study.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Baek, Y., Lee, B., Han, D., Yun, S., Lee, H.: Character region awareness for text detection. In: CVPR. pp. 9365–9374 (2019)
2. Caron, M., Touvron, H., Misra, I., Jegou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: ICCV (2021)
3. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: III, H.D., Singh, A. (eds.) ICML. Proceedings of Machine Learning Research, vol. 119, pp. 1597–1607 (2020)
4. Dutta, A., Zisserman, A.: The VIA annotation software for images, audio and video. In: ACM Multimedia. p. 2276–2279 (2019)
5. Elyan, E., Jamieson, L., Ali-Gombe, A.: Deep learning for symbols detection and classification in engineering drawings. Neural Networks **129**, 91–102 (2020)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. (eds.) NIPS. vol. 27, pp. 2672–2680 (2014)
7. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: CVPR. vol. 2, pp. 1735–1742 (2006)
8. Hamilton, M., Zhang, Z., Hariharan, B., Snavely, N., Freeman, W.T.: Unsupervised semantic segmentation by distilling feature correspondences. In: ICLR (2022)
9. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: CVPR. pp. 9729–9738 (2020)
10. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3. In: ICCV. pp. 1314–1324 (2019)
11. IEA: Energy systems buildings database (2024), `https://www.iea.org/energy-system/buildings`, last accessed 2025/03/11
12. Jamieson, L., Moreno-García, C., Elyan, E.: A review of deep learning methods for digitisation of complex documents and engineering diagrams. Artificial Intelligence Review **57**, 136 (2024)
13. Joy, J., Mounsef, J.: Automation of material takeoff using computer vision. In: IAICT. pp. 196–200 (2021)
14. Kim, H., Lee, W., Kim, M., Moon, Y., Lee, T., Cho, M., Mun, D.: Deep-learning-based recognition of symbols and texts at an industrially applicable level from images of high-density piping and instrumentation diagrams. Expert Systems with Applications **183**, 115337 (2021)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) ICLR (2015)

16. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: ICML (2015)
17. Lin, T.Y., Maire, M., Belongie, S.J., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV. pp. 740–755 (2014)
18. Lowe, D.: Object recognition from local scale-invariant features. In: ICCV. vol. 2, pp. 1150–1157 (1999)
19. Moreno-García, C.F., Elyan, E., Jayne, C.: New trends on digitisation of complex engineering drawings. Neural Comput. Appl. **31**(6), 1695–1712 (2019)
20. OpenAI: Hello gpt-4o (2024), `https://openai.com/index/hello-gpt-4o/`, last accessed 2025/03/11
21. Paliwal, S., Jain, A., Sharma, M., Vig, L.: Digitize-pid: Automatic digitization of piping and instrumentation diagrams. In: Gupta, M., Ramakrishnan, G. (eds.) PAKDD. pp. 168–180 (2021)
22. Paliwal, S., Sharma, M., Vig, L.: Ossr-pid: One-shot symbol recognition in p&id sheets using path sampling and gcn. In: IJCNN. pp. 1–8 (2021)
23. Payandeh, A., Baghaei, K.T., Fayyazsanavi, P., Ramezani, S.B., Chen, Z., Rahimi, S.: Deep representation learning: Fundamentals, technologies, applications, and open challenges. IEEE Access **11**, 137621–137659 (2023)
24. Rahul, R., Paliwal, S., Sharma, M., Vig, L.: Automatic information extraction from piping and instrumentation diagrams. In: Marsico, M.D., di Baja, G.S., Fred, A.L.N. (eds.) ICPRAM. pp. 163–172 (2019)
25. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR. pp. 779–788 (2016)
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) NIPS. vol. 28, pp. 91–99 (2015)
27. Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T., Gehler, P.: Towards total recall in industrial anomaly detection. In: CVPR. pp. 14318–14328 (2022)
28. Samet, H., Soffer, A.: Magellan: Map acquisition of geographic labels by legend analysis. International Journal on Document Analysis and Recognition **1**, 89–101 (1998)
29. Sarkar, S., Pandey, P.K., Kar, S.: Automatic detection and classification of symbols in engineering drawings. arXiv preprint arXiv:2204.13277 (2022)
30. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: CVPR. pp. 815–823 (2015)
31. Stinner, F., Wiecek, M., Baranski, M., Kümpel, A., Müller, D.: Automatic digital twin data model generation of building energy systems from piping and instrumentation diagrams. In: ECOS. pp. 2854–1865 (2021)
32. Theisen, M.F., Flores, K.N., Schulze Balhorn, L., Schweidtmann, A.M.: Digitization of chemical process flow diagrams using deep convolutional neural networks. Digital Chemical Engineering **6**, 100072 (2023)
33. Wu, Z., Xiong, Y., Yu, S.X., Lin, D.: Unsupervised feature learning via non-parametric instance discrimination. In: CVPR. pp. 3733–3742 (2018)
34. Xiao, X., Li, Z., Zhao, S., Yang, L., Zhao, F., Ge, C.: Improved p&id symbol detection algorithm based on yolov5 network. In: SMC. pp. 120–126 (2023)
35. Yu, E.S., Cha, J.M., Lee, T., Kim, J., Mun, D.: Features recognition from piping and instrumentation diagrams in image format using a deep learning network. Energies **12**(23), 4425 (2019)