An Automated Engineering Assistant: Learning Parsers for Technical Drawings

Dries Van Daele,[1] **Nicholas Decleyre,** [2] **Herman Dubois,** [2] **Wannes Meert** [1]
[1] KU Leuven, Dept. of Computer Science, Leuven, Belgium
[2] Saint-Gobain Mobility | Engineered Components (Seals), Kontich, Belgium
dries.vandaele@cs.kuleuven.be

Abstract

Manufacturing companies rely on technical drawings to develop new designs or adapt designs to customer preferences. The database of historical and novel technical drawings thus represents the knowledge that is core to their operations. With current methods, however, utilizing these drawings is mostly a manual and time consuming effort. In this work, we present a software tool that knows how to interpret various parts of the drawing and can translate this information to allow for automatic reasoning and machine learning on top of such a large database of technical drawings. For example, to find erroneous designs, to learn about patterns present in successful designs, etc. To achieve this, we propose a method that automatically learns a parser capable of interpreting technical drawings, using only limited expert interaction. The proposed method makes use of both neural methods and symbolic methods. Neural methods to interpret visual images and recognize parts of two-dimensional drawings. Symbolic methods to deal with the relational structure and understand the data encapsulated in complex tables present in the technical drawing. Furthermore, the output can be used, for example, to build a similarity based search algorithm. We showcase one deployed tool that is used to help engineers find relevant, previous designs more easily as they can now query the database using a partial design instead of through limited and tedious keyword searches. A partial design can be a part of the two-dimensional drawing, part of a table, part of the contained textual information, or combinations thereof.

Introduction

Technical drawings are the main method in engineering to visually communicate how a machine or component functions or is constructed. They are the result of a design process starting from a set of specifications that the final product needs to comply with. This design process follows a number of strict and soft rules (e.g., material choice as a function of temperature). Figure 1 shows a typical example containing both a 2D and 3D visualisation of the object, and a material list in tabular form specifying parts and properties. These drawings are laid out according to generally applied conventions, and engineering companies have a large database of previous designs, potentially going back decades.

These databases are often underutilized because previous designs can only be searched for by title or by using a limited set of textual annotations. Ideally, however, such a database can also be used to: (1) given a technical drawing, find other relevant drawings in a large database of previous designs; and (2) given a partial description or drawing, find designs that would complete the partial design. In this work we present an approach that can extract the knowledge in a technical drawing and thus improve the search capabilities significantly to achieve the aforementioned tasks and assist engineers during the design process.

In order to fully benefit from technical drawings, we need to extract their tabular as well as their visual information, and translate it to a representation that can be handled by automated systems. Furthermore, such a system should be able to deal with both recent digital drawings and historical analog drawings. The latter is important because a great amount of information is captured in legacy drawings. Ideally, extracting the information can be done using a parser, which is a small computer program. The main challenge is that writing and maintaining such a parser is time-consuming and costly. Furthermore, it is error prone since an expert has to explain subtle rules to an analyst or a programmer. The approach we present here will learn such parsers directly from expert annotations on the original drawing and allow its output to be used in automated tasks such as searching relevant designs.

Providing these annotations is a trivial task for domain experts. The number of drawings that require annotation is mainly dependent on the number of variations or templates that need to be recognized. Fortunately, since all technical drawings within an organisation are expected to be (loosely) based on a limited set of templates, the number of drawings that need to be annotated is also limited.

We present a hybrid approach that utilizes both neural methods and reasoning-based methods. This combination is necessary to capture the full range of information available in a drawing and provided by experts. Neural methods (e.g., deep convolutional neural networks) are used because they are the state-of-the-art in image recognition algorithms. Despite successes in image recognition, however, automatic analysis and processing of engineering drawings is still far from being complete (Moreno-García, Elyan, and Jayne 2018). This is in part because neural methods require

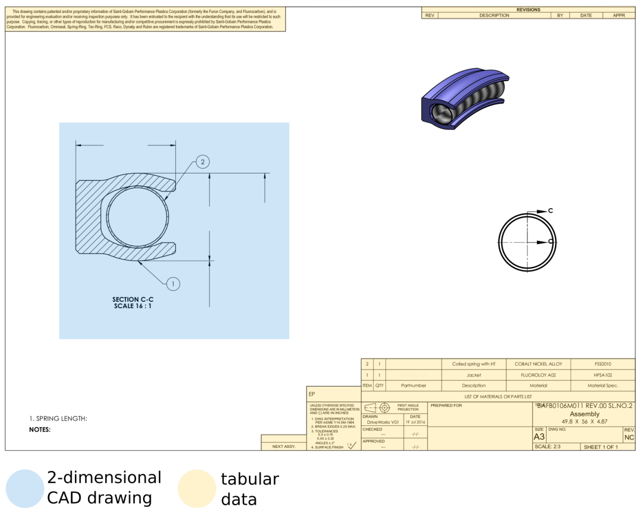


Figure 1: A technical drawing with highlights indicating the 2D CAD drawing and the tabular data

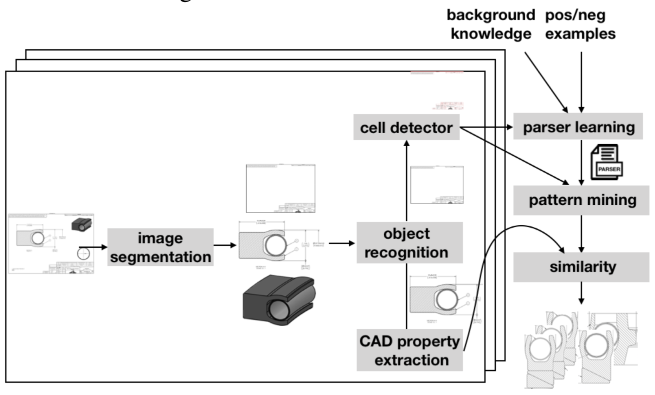


Figure 2: Overview of the technical drawing similarity proposal system.

large amounts of training data. While such data are not always available, an expert might be capable of summarizing part of the knowledge in just a few abstract concepts. To exploit this expert knowledge, we also utilize reasoning-based methods such as inductive logic programming (ILP) (De Raedt et al. 2008). Such a hybrid approach that combines data-driven methods with knowledge-driven methods is gaining in popularity since real-world tasks such as parsing technical designs tend to require Hybrid AI (Manhaeve et al. 2018; Mao et al. 2019). We have developed and deployed this system for learning to parse and search technical designs.

Its modular design is explained in more detail in the following sections. This work presents five contributions that enabled us to surpass the state-of-the-art in parsing technical designs: First, we introduce the use of ILP to learn parsers from both data and expert knowledge to interpret technical drawings. Second, we introduce a novel bootstrapping learning strategy for ILP that speeds up learning and increases accuracy. Third, we propose the use of a siamese deep learning architecture to meaningfully summarize CAD drawings. Fourth, we introduce a similarity measure to find related technical drawings in a large database. Finally, the efficacy of this method is demonstrated in a number of experiments on a real-world data set. As seen in Figure 2, these contributions are reflected in the modular structure of our implementation.

Identify Elements in a Technical Drawing

The first action is to identify the different elements in a drawing, thus tables and CAD drawings. To design and test the system, we have access to 5000 archived technical drawings that need interpreting. Archived technical drawings are digitized to varying degrees. Because of this, we consider the case where the technical drawing is represented as a bitmap image ($\approx 3300 \times 2300$ pixels)

Segment the Image

Segmenting the design into its different elements is achieved using conventional computer vision methods. The image is partitioned into its main segments using DBSCAN with $\epsilon = 30$ and minimum points set to 0.001% of total pixels, thus ≈ 75 points (Ester et al. 1996). Since a technical drawing employs white space to distinguish central layout elements, such a density-based method is highly effective. No errors were observed in the segmentation of the drawings.

Recognize Image Segments

Next, the system recognizes what each image segment represents by classifying them as one of three possible classes: 'tables', 'two-dimensional CAD drawings', and 'irrelevant' segments. Since the classes are visually distinct high predictive accuracy can be achieved with a small CNN classifier. This classifier is constructed using the PyTorch library (Paszke et al. 2017) and consists out of three convolution layers and three fully connected layers. It was trained against 318 randomly selected technical drawings that were annotated by an expert, for a total of 3000 image segments (318 tables, 372 CAD drawings, 2310 irrelevant segments). No classification errors were made on a randomly selected test set of 53 technical drawings containing 500 segments.

In case a table is recognized, we additionally identify the cells by applying a contour detection algorithm provided by the OpenCV library (Suzuki and Abe 1985). All cells are then passed to the *parser learning*-module. In case a two-dimensional CAD drawing is recognized, the image data is passed on to the *CAD property extraction*-module.

Extract Properties in Tables

The data contained in a technical drawing is laid out in a manner that facilitates human interpretation. Tabular data in particular tends to be organised both spatially and through explicit annotation. Common examples of spatial structuring involve assigning related cells to common rows or columns, while positioning unrelated cells further from one another. Particularly useful are cells that contain unambiguous keywords such as attribute names. These are helpful to gain insight in the structure of a table. They serve as anchors to cells that are less distinctive but can easily be described relative to them.

The application at hand does not only require us to parse a table, but also demands that we learn how to interpret its spatial organisation. A small computer program is required to parse these custom drawings. Programming and maintaining a parser for each type of drawing is not only an expensive and time consuming task, but also error-prone. First, the structure of such technical drawings needs to be explained to a non-expert, i.e. a programmer, who interprets the instructions. Second, the tables are typically not simple rectangular tables. They thus require a non-trivial parser that is difficult to understand. Third, a design can deviate slightly or change over time requiring periodic maintenance and potentially leading to software erosion. Ideally these programs would be derived directly from the expert's knowledge, and be easily updated when new designs appear. This is possible by means of machine learning techniques that learn programs from examples. The examples in this setting are obtained by annotating technical drawings, a task that is trivial for a domain expert.

The highly relational nature of tabular data and the ease with which tables can be sensibly navigated by visiting adjacent cells suggests the use of Inductive Logic Programming (ILP). ILP systems are particularly suitable for learning small programs from a limited amount of complex input data. When learning the programs covered in this work using ILP, we benefitted in particular from the ability to learn recursive definitions (e.g., row $n+1$ is defined by row n) and reuse learned target labels (e.g., first learning what a header row is helps to define what a content row is).

Learn Parsers Using Inductive Logic Programming

ILP learns a logic program, which consists of a set of definite clauses. Each definite clause can be interpreted as a rule. A clause is of the form: $h(a, X) \,\text{:-}\, b_1(a, X),\ b_2(X)$. where $h(a, X)$, $b_1(a, X)$, and $b_2(X)$ are literals whose arguments can either be constants (a) or logical variables (X). Constants are denoted using lowercase letters or numbers, while variables are uppercase letters. Disjunction is represented using ';' and conjunction using ','.

An ILP system learns a set of definite clauses from relational data. Given background knowledge B, positive examples E^+ and negative examples E^-, it attempts to construct a program H consisting of definite clauses such that $B \wedge H$ entail all, or as many as possible, examples in E^+, and none, or as few as possible of those in E^-.

We thus need to supply three types of inputs. First, a set of training examples E, containing the properties that identify a cell:

- *Cell text:* The textual contents of each cell. Tesseract 4.0 is used to recognize cell contents (Smith 2007).
- *Cell location:* The cell's bounding box information (i.e. (x,y) coordinates and cell width and height).

Second, a label for each cell (e.g., author, bill of materials, quantity). A cell can be annotated with multiple labels (e.g., a cell can be a quantity in the bill of materials). Depending on which target label we want to learn, we split the set of examples E in a tuple (E^+, E^-) where E^+ contains the examples associated with a cell that has the target label and E^- those examples that do not. For standard ILP, the learning task is defined for one target label, so we repeat the standard ILP task for each label in the set of labels.

Third, we can provide background knowledge B that contains generally applicable knowledge for the problem at hand and remains unchanged across examples. In this case we provide:

- *Relative cell positions.* Relations capturing which cells are adjacent to each other, and in which direction (horizontally or vertically) based on their bounding boxes.
- *Numerical order.* The successor relationship. Although not essential, it is useful for learning concise, recursive rules.

The output of ILP, the program H, is a set of definite clauses like '$author(A) \,\text{:-}\, cell\_contains(A, drawn).$' which can be read as the rule 'Cell A contains the author if it contains the word $drawn$'.

Improved Learning with Bootstrapping

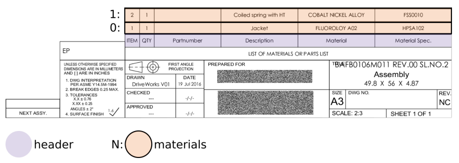
The ease with which an effective parser can be learned is expected to vary across target labels. We propose a bootstrapping extension that supports the construction of sophisticated programs by allowing them to employ the simpler ones in their definition. This is loosely inspired by the ideas raised by Dechter et al. (2013), but applied to the ILP setting.

This corresponds to a variation of the previously discussed ILP set-up where a dependency graph G is used. The nodes in this directed acyclic graph each represent a possible target label and the edges represent dependencies between those labels. A dependency indicates that one target label might have a natural description in function of another. Our method automatically constructs a sensible dependency graph. First, standard ILP is applied to learn programs for each target. Then, targets are ranked according to, first, ascending F_1 score on the training data and, second, the size of the program in number of literals. Each target in the ranking then has all subsequent targets as its dependencies. Finally, ILP with bootstrapping learns targets in the order specified by a correct evaluation order of G, and extends the background knowledge B for each target with the programs constructed to parse its dependent target labels. When learning program H using bootstrapping to capture a particular target label l, we define its extended background knowledge $B' = B \wedge (\bigwedge_{i \in descendants(G,l)} H_i)$, where H_i is the program trained for target label i.

Experiment Set-Up

The ILP system Aleph (Srinivasan 2001) is used to learn possibly recursive programs that parse the chosen targets from the tabular data, ranging from the document's author and its approval date to the attributes covered in the materials table and its indexed components.

To collect training and testing data, i.e. a set of fully labeled technical drawings, we built a data labeling tool with a web-based graphical interface to support domain experts in labeling drawings. Using this tool, 30 technical drawings with on average 50 cells were labeled with 14 different labels. For each target label, examples that contain that label



(a) A table excerpt from a technical drawing. Its header and materials are highlighted.

```
% Materials hypothesis                                          % Header hypothesis
materials(A,B) :-          materials(A,B) :-                    header(A) :-
  zero(A),                   succ(C,A),                           above_below(A,B),
  above_below(B,C),          above_below(B,D),                    cell_contains(B, 'LIST').
  header(C).                 materials(C,D).
```

(b) header/1 covers any cell located directly above a cell containing the word 'LIST'. materials/2 parses the indexed parts of the materials table. Its first argument is the index and its second argument represents the cell. materials/2 consists of two clauses. The first clause anchors the table by considering row 0 to consist of the cells above the header. It employs header/1 in its definition. The second, recursive clause indicates that the index is incremented whenever a row is located above another.

Figure 3: Figure a provides an illustration of the materials table and its header. Listing b shows the associated program learned using bootstrapping.

label	ILP	GBM	
	F_1	F_1	incorrect drawings
description	1	0.9164	20%
material	1	0.9355	20%
index	1	0.9749	7%
number	1	0.9749	7%
quantity	1	0.9926	7%
materialspec	1	1	0%
all labels			36.7%

Table 1: Comparison of the performance of our proposed approach versus a gradient boosting model in identifying the columns of the bill of materials.

form its positive example set, while negative examples are automatically derived by taking the complement of all possible examples for that target with its positive example set.

When inducing programs, we employed a proof depth of 12, a clause length of 5, and an upper bound of 60,000 on the number of nodes that could be explored during clause learning.

Learned Parser Programs versus Patterns

To verify that learning programs instead of patterns is preferable for this task, we compared ILP with Gradient Boosting Machines (GBM) and Multilayer perceptrons (MLP). GBM came out best performing. GBM is a state-of-the-art machine learning model with a strong track record in real-world use cases (Anghel et al. 2018). Table 1 compares the performance of ILP with bootstrapping against that of GBM when learning models to recognize the columns in the bill of materials. The models were trained using leave-one-out cross-validation. GBM (multiclass classification using XG-

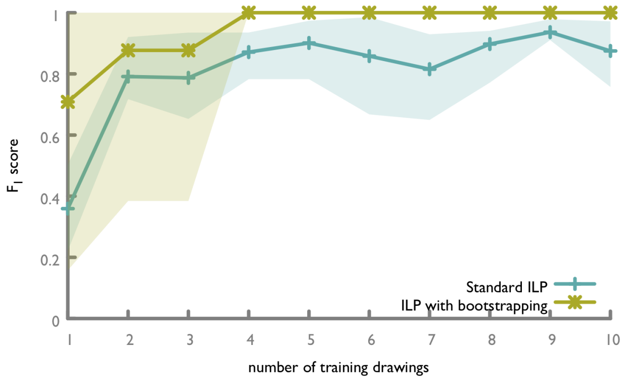


Figure 4: F_1 score of programs learning materials/2. Min/-max shading indicates the range of performance between the best and worst-performing program over 5 repetitions.

Boost against the contents and location of a cell and its neighbours) achieves high performance, yet – unlike our proposed method – fails at fully capturing the target concepts. When taking all the relevant column labels in consideration, GBM only interprets 19 out of 30 technical drawings entirely correctly. The errors GBM introduces carry a high cost, as any error drastically alters our model of a design. Allowing such errors to persist would undermine the trustworthiness of any patterns or conclusions drawn in the subsequent steps. Because of this, immense manual effort would be required to double-check the correctness of the classified cells. The poorest results for GBM are observed on the `description` column. This column is positioned in the middle of the bill of materials, and contains diverse contents across technical drawings. Here in particular, the ability to learn a recursive definition proves essential.

We do find that GBM has an easier time learning some columns over others. The `materialspec` column contains a unique vocabulary compared to other cells, and is positioned on the side of the table, making it particularly recognisable. Similarly, the `index` and `quantity` column are unique since theirs are the only cells that contain single integers. Furthermore, these two columns can easily be distinguished from one another, since contrary to `quantity`, there are no cells to the left of `index`. Nevertheless we find that even for highly recognisable cells such as these, classification errors can slip in on models trained using GBM.

Learning Using Bootstrapping

Learning perfect parsers for simple labels such as author or approval date, and even more complex ones like the columns of the bill of materials, can be achieved by both standard ILP and the bootstrap method with only a few training examples. More interesting is to look at the most complicated label, the indexed components (`materials/2` in Figure 3a). Figure 4 visualizes the performance between the standard ILP setting and our proposed bootstrapping extension on this label. Here, the training set consists of up to 10 drawings (randomly selected over 5 repetitions), while performance is evaluated against a test set of 20 drawings. The results show

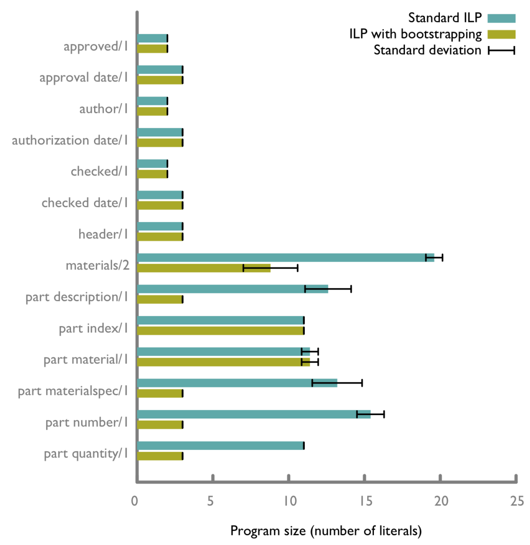


Figure 5: Difference in average program size (number of literals) between standard ILP and ILP with bootstrapping across programs induced on 14 labels. 10 repetitions on a training set of 20 labeled drawings.

that it only takes a few annotated designs for the bootstrapping method to learn a perfect parser, while the standard ILP approach fails to do so. Furthermore, it highlights how ILP with bootstrapping compared to Standard ILP is less sensitive to overfitting when presented with additional training data. This robustness of ILP with bootstrapping lends itself well to incremental learning.

The best performing program constructed using standard ILP in Figure 4 consists of 14 clauses and yields 17 false negatives. Bootstrap learning, however, succeeds at learning a completely accurate, concise program whenever more than three technical drawings are provided in the training set. The poor performance observed in some experiments when using only a few drawings is due to poor generalization potential. More specific, in these drawings the materials tables provided for training each consisted of only a single row in addition to the header, causing there to be no pressure on the inductive learner to add the recursive rule necessary to capture the rows of larger tables.

A comparison of program size between Standard ILP and ILP with bootstrapping across all investigated labels is shown in Figure 5.

Extract Properties from CAD 2D Visualisation

Some aspects of a design can only be communicated by sharing a visual depiction. This concerns for example the subtleties of component shapes, and the exact manner of their assembly. These features can prove essential in distinguishing designs that have identical tabular information (e.g., material choices), and as such can be vital to construct a comprehensive representation of the technical drawing. A technical drawing tends to contain both a 2D and 3D depiction. We consider the 2-dimensional CAD drawing as the most suitable target for visual analysis as the profile view offers a clear, uncluttered view of the design.

Learn Key Identifiers from Unlabeled Data

Many of the visual features of interest are only present in the CAD drawing and cannot be linked to features in the table or in the meta-data. This means we cannot apply standard supervised learning because there are no labels available. However, this is not a problem since our interest lays in the identifying features of designs rather than their type. The goal will thus be to learn a limited set of features that are expressive enough to uniquely identify designs, can generalize over different designs, and remain unaffected by translations or rotations. We can thus use self-supervised learning to train our models using proven supervised methods (Misra and van der Maaten 2020)

We propose to transform the problem to a binary classification task that captures these requirements. Given a pair of 2-dimensional CAD drawings, a classifier with a limitation on the numbers of features is trained to predict whether the pair represents the same design or not. If the classifier achieves high accuracy on this task we consider the set of learned features to fit the requirements and we will use this set of features in a next step to summarize each design.

The data set for these input pairs is constructed as follows. For each of the 2-dimensional CAD drawings, we generate 10 variations by applying arbitrary flips, rotations and translations. We consider this image set consisting of 11 images to be representative for each design. The data for the 'same' class is then formed by considering every pairing within each image set, while the 'different' class is constructed by sampling an equal number of pairs across image sets. This ensures the resulting data set is balanced.

CNN Architecture

Neural networks are widely known for their capability of capturing complex and non-obvious properties of the data they are trained with. Convolutional Neural Networks (CNNs) are a category of neural networks of particular interest, as they have seen wide adoption in image recognition and classification. They are classifiers whose key characteristic is their usage of convolutional layers. An input image is passed through a series of such layers. Each layer consists of convolved features (i.e., the neurons) by applying a kernel on parts of its input. While the features captured in early layers are limited to angled edges and simple blobs, they become increasingly more complex as the layers deepen, until they are capable of describing domain-specific elements.

Convolutional neural networks have a proven capacity to learn complex features (i.e., representation learning). The challenge is to identify those that match our requirements. Notably, we find that the classification task outlined in the previous section can directly be integrated in a CNN architecture, allowing it to learn features that are optimized to score well on the classification task.

Figure 6 shows our siamese CNN architecture. The ResNet-50 (He et al. 2016) architecture is used to perform the various convolutions. Since the features identified in the

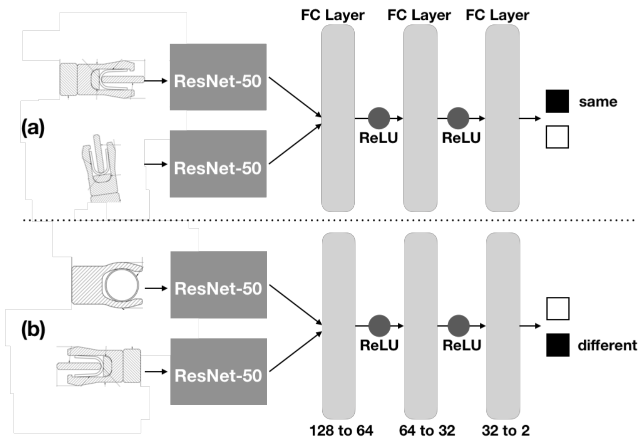


Figure 6: The ResNet-50 block maps the 2048 inputs of its final fully connected (FC) layer to a layer of size 64. This layer represents the derived visual features of interest.

pre-trained model are tailored to a wide set of common settings, while ours is very domain-specific, we re-train only the last layer to tailor the derived features to our data. While the output layer of ResNet-50 has size 1000, we map it to a layer of size 64. This layer corresponds to the features that will be used to visually identify a design. The rest of the architecture is constructed in order to effectively classify the input image pairs into one of the two possible classes: 'same', or 'different'. Note that the same ResNet-50 network is used to encode both images.

Experimental Results

In order to prove useful, the CNN has to be attentive of features of varying detail, as the visual differences can vary from striking to intensely subtle. Our classifier achieves a 96.8% accuracy (training set: 68,507 pairs, validation set: 34,253 pairs, test set: 68,507 pairs).

t-Distributed Stochastic Neighbor Embedding (t-SNE) We can now apply the CNN network to all drawings in the database and construct a feature vector from the 64 relevant features, thus the neurons in the last layer of our ResNet-50 model. Figure 8 shows a t-SNE visualisation of the features vectors for each of the drawings in our dataset. The data points are colored according to an expert labeling which groups designs according to properties deemed to have a high visual impact. This labeling has two values with high representation, and the visualisation clearly separates them in distinct, non-overlapping clusters. Since our interest lies in the detection of novel features, being able to identify a pre-existing one is not actually our goal. However, since a failure to visually distinguish between these types would falsify our hypothesis that we are extracting meaningful visual data, this result does inspire confidence that informative features are found.

Gradient-weighted Class Activation Mapping (Grad-CAM) This belief is strengthened further when analyzing the behaviour of the CNN using Grad-CAM (Selvaraju et al.

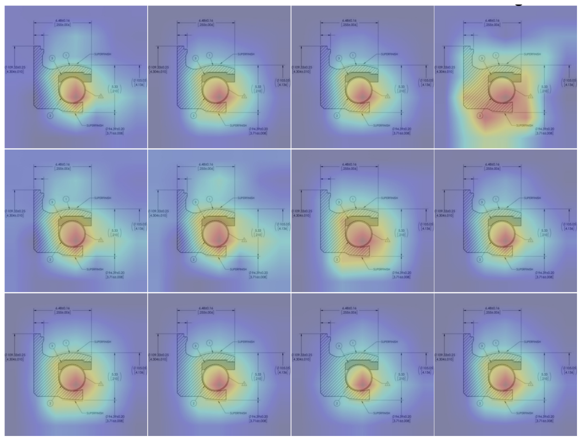
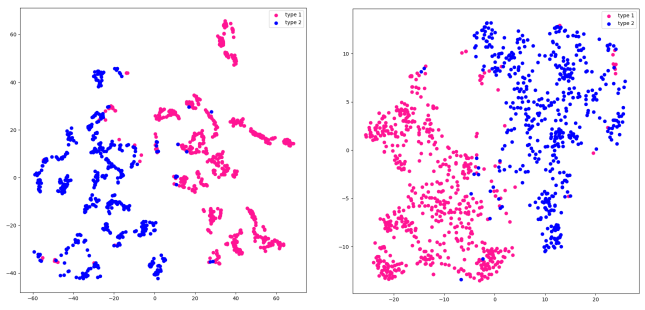


Figure 7: A Grad-CAM visualization. Warmer areas correspond to regions that play a more significant role in the activation of a derived feature. Here, each image depicts a feature whose attention is concentrated on a particular key component of the design.



(a) perplexity 20 (b) perplexity 100

Figure 8: t-SNE visualisation over 2 components for varying values of perplexity.

2017). Grad-CAM provides a coarse localization map of the important regions in the image. We find that our derived features focus their attention on image regions that correspond to meaningful properties in our application domain. Figure 7 visualizes a selection of the derived features activating in the presence of a particular key component.

Identifying Relevant Designs

We now have access to the design properties contained in both the tabular data and the visual depiction. In order to leverage these results when suggesting relevant designs, we first impose a similarity measure on them separately, and then combine the resulting similarity measures using a weighted geometric mean.

Tabular Data Similarity

Due to its highly relational nature, it is not feasible to directly apply a conventional similarity measure to the tabular data. Instead, we first propositionalize the data using Frequent pattern (FP) mining (Kramer, Lavrač, and Flach 2001).

```
% A design with a part with 'double' and 'nickel'
 materials(PartRow,A), cell_contains(A,double),
 materials(PartRow,C), cell_contains(C,nickel).

% A design with a 'spring' made out of 'cobalt'
 materials(PartRow,A), cell_contains(A,cobalt),
 materials(PartRow,C), part_description(C),
 cell_contains(C,spring).

% A design with 'jacket','spacer', and 'Spring'.
 part_description(A), cell_contains(A,jacket),
 part_description(B), cell_contains(B,spacer),
 part_description(C), cell_contains(C,spring)
```

Listing 1: Three frequent patterns mined using WARMR

We perform FP mining using WARMR (Dehaspe and Toivonen 1999) on the extracted tabular data. We focus on retrieving patterns that occur in $\geq$ 10% of the drawings. Listing 1 shows a sample of the 9120 patterns we mined. Each technical drawing is then represented as a binary vector indicating which patterns are applicable. Given such a vector representation, closely related designs can be identified by performing a ranking using the complement of its normalized Hamming distance to all other designs. Given two drawings represented as binary vectors X and Y, $\textit{sim}_{\textit{tabular}}(X,Y) = 1 - \frac{1}{n}\sum_{i=1}^{n}|X_i - Y_i|$.

Visual Similarity

Previously we discussed how the final fully connected layer of our ResNet-50 architecture captures key visual propertiesusing a layer of size 64. All the features in this layer are continuous. Since features of visually similar objects are expected to have similar values, cosine similarity against this set of derived features is used to determine similarity. Given two drawings represented as feature vectors X and Y, $\textit{sim}_{\textit{visual}}(X,Y) = \frac{X \cdot Y}{||X||_2 \cdot ||Y||_2}$.

Ranking Designs by Similarity to a Given Design

Tabular data similarity and visual similarity are combined using the weighted geometric mean $\prod_i \left(x_i^{w_i}\right)^{1/\sum_i w_i}$. The use of a geometric mean ensures that the resulting similarity is not biased towards a particular one of its constituent similarity measures due to the distribution of their values, but rather accounts only for the relative changes to each score across designs. The use of a weighted mean makes the inherent trade-off between potentially conflicting measures explicit.

Here, we use this convex combination solely to combine tabular data similarity and visual similarity. This trade-off can be captured in a single parameter α such that $\textit{similarity} = (\textit{sim}_{\textit{tabular}})^{\alpha} \cdot (\textit{sim}_{\textit{visual}})^{1-\alpha}$.

In doing so, users can easily impose their own biases and preferences to influence the ranking. In our implementation we use $\alpha = 0.5$. Figure 9 shows some of the ranked designs given a particular technical drawing.

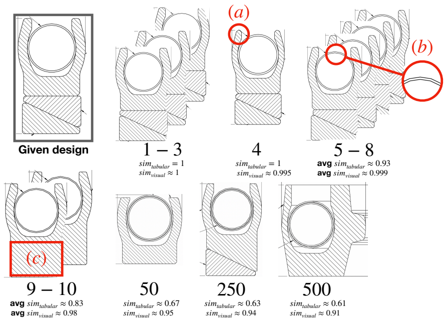

(a)
(b)
(c)
Given design
1 – 3
$sim_{tabular} = 1$
$sim_{visual} \approx 1$
4
$sim_{tabular} = 1$
$sim_{visual} \approx 0.995$
5 – 8
avg $sim_{tabular} \approx 0.93$
avg $sim_{visual} \approx 0.999$
9 – 10
avg $sim_{tabular} \approx 0.83$
avg $sim_{visual} \approx 0.98$
50
$sim_{tabular} \approx 0.67$
$sim_{visual} \approx 0.95$
250
$sim_{tabular} \approx 0.63$
$sim_{visual} \approx 0.94$
500
$sim_{tabular} \approx 0.61$
$sim_{visual} \approx 0.91$

Figure 9: Selected designs annotated with a rank indicating their similarity to the highlighted design ($\alpha = 0.5$). Subtle though significant differences are seemingly picked up, while insignificant though sizable differences in shading seem to have little to no effect. The feature highlighted in (a) contrasts with its less rounded depiction in the original, possibly explaining the difference in its sim_{visual} score. The ring in (b) appears as a polygon instead of a circle, and the body of (c) consists of a single contiguous part as opposed to multiple elements separated by a diagonal.

Ranking Designs by Similarity to a Given Partial Design

The ability to identify similar designs is particularly useful when applied to a design under construction. If such a design displays high similarity to existing ones, the most similar ones can provide the engineer with possible completions.

We consider a partial design to be represented as a technical drawing containing a number of empty cells. During property extraction, the cell text of empty cells is represented using logical variables. If a visual depiction of the design is available, sim_{visual} can be computed as usual. If it is omitted, the similarity measure is restricted to its $sim_{tabular}$ term.

As a consequence of allowing the inclusion of empty cells, some of the patterns employed by the feature vector used to construct $sim_{tabular}$ might not have a proper instantiation on the partial design. Given T the set of all properties of the partial technical drawing, we distinguish three situations:

- *True.* A feature is true on the partial design when each of the literals in its conjunction have a corresponding match in T that does not require the instantiation of any logical variable in T.
- *False.* A feature is false on the partial design if at least one of the literals in its conjunction fails to find a corresponding match in T.
- *Unknown.* Any feature for which it is not known whether it is *True* or *False*.

$sim_{tabular}$ against a given partial design can then be computed by utilizing feature vectors that are filtered to contain only those features that were *True* or *False* on the partial design.

Related Work

Digitisation of Technical Drawings

Two recent papers cover the domain of document digitisation. First, Staar et al. (2018) shows significant progress digitising general PDF and bitmap documents. Second, Moreno-García, Elyan, and Jayne (2018) provides an overview of all recent trends on digitising engineering drawings in particular. The work of (Staar et al. 2018) focuses on detecting elements in text documents in general and parses tables but cannot take easily into account expert knowledge to achieve near perfect extraction for specific cases or extract information embedded in figures.

With regards to shape detection, Moreno-García, Elyan, and Jayne (2018) makes a distinction between *specific* approaches focused on identifying shapes that are known in advance, and *holistic* approaches where the underlying rules of the drawing are exploited to split it into parts. Our approaches for identifying technical drawing elements utilized a mixture of both. The use of image segmentation and text detection in "Identify Elements in a Technical Drawing" falls under the holistic view, while the contour detection "Recognize Image Segments" is an illustration of a specific approach. While Moreno-García, Elyan, and Jayne (2018) notes that some frameworks do perform contextualisation, like recognizing symbols in a technical drawing, but none parse the full table. Our approach is to our knowledge the first that enables the construction of a comprehensive, formal representation of a technical drawing by learning an interpretable parser capable of taking into account expert knowledge and can identify unique properties in drawings.

Feature Extraction

While established feature extraction methods such as SIFT (Lowe 2004) and SURF (Bay, Tuytelaars, and Van Gool 2006) are still viable alternatives, CNNs are increasingly the go-to method when extracting features. Industrial applications of CNNs are however strongly limited by the cost of collecting a suitable set of labeled training data (Moreno-García, Elyan, and Jayne 2018). Our approach side-steps this issue by learning from unlabeled data through the introduction of a discriminative setting. This approach is comparable to Exemplar-CNN (Dosovitskiy et al. 2014).

In this work, autoencoders were observed to perform poorly as a means to capture features. When encoding an input image to a lower dimensional feature space, an autoencoder seeks to capture as much of the input data as possible with the aim of later on reconstructing the image as faithfully as possible. Here however, most of the input data proves to be irrelevant. The exact position and rotation of the visual depiction of a design is completely irrelevant, and even though the shading represents a large amount of data, it is not something that merits encoding. We found that our proposed, discriminative approach is far more suitable for identifying notable features.

Inductive Logic Programming Systems

The ILP system Aleph was used for the parser learning. Related are all the ILP systems that currently define the state-of-the-art. This includes Tilde (Blockeel and De Raedt 1997), Aleph (Srinivasan 2001), Metagol (Cropper and Muggleton 2016), Progol (Muggleton 1995), and FOIL(Quinlan and Cameron-Jones 1993).

While Aleph learns from entailment, Tilde learns a relational decision tree from interpretations. Both systems were considered, but only Aleph was capable of constructing recursive programs. This allows it to construct concise programs, making Aleph our system of choice. FOIL is expected to be similarly suitable. Metagol is also highly effective at this task, as its metarules allow for a more targeted search for recursive programs. A downside of this system is that meta-rules are currently user-defined, imposing an additional burden on the user, who in this setting is a domain expert with no background in ILP. Automatic identification of metarules is ongoing work (Cropper and Tourret 2018).

Deployment

After deployment - having processed 5000 archived drawings - the tool was used to automatically tag around 1% of the drawings for removal and clean the data set because they were unfinished, had overlapping objects, or were scrambled.

A workshop was organised during which engineers evaluated the effectiveness of the tool and its suggested design completions. Overall, it was considered to see daily use, saving at least 15 and possibly more than 30 minutes of time per use when compared to their previous workflow. Particular areas of note were its usefulness in exploring the viable design space, and its role in improving consistency when converging onto a final design.

Conclusions and Future Work

We introduced an approach to assist an engineer by automatically interpreting technical drawings and allowing for a flexible search method.

To achieve this we introduced five contributions. First, we introduced the use of ILP to learn parsers from data and expert knowledge to interpret a technical drawing and produce a formal representation. Second, we introduced a novel bootstrapping learning strategy for ILP. Third, we introduced a deep learning architecture that learns a meaningful summarization of CAD drawings by identifying unique properties in drawings. Fourth, we introduced a similarity measure to find related technical drawings in a large database. Finally, the efficacy of this method was demonstrated in a number of experiments on a real-world data set.

Based on this work, additional tasks are now within reach that would be useful in an automated engineering assistant. For example, given the interpreted technical drawings, one can learn constraints or rules that apply to a given set of designs. Such rules can then later be used to automatically verify novel designs or find anomalous designs by identifying constraints that are violated.

Acknowledgements

This work was supported by Flanders Innovation & Entrepreneurship (VLAIO O&O project 'Digital Engineer').

The tools presented are built upon open-source software supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694980, SYNTH: Synthesising Inductive Data Models). The authors would like to thank Luc De Raedt and Sreeraj Rajendran for their valuable suggestions and Ashwin Srinivasan for creating and permitting the usage of Aleph.

References

Anghel, A.; Papandreou, N.; Parnell, T. P.; Palma, A. D.; and Pozidis, H. 2018. Benchmarking and Optimization of Gradient Boosted Decision Tree Algorithms. *CoRR* abs/1809.04559. URL http://arxiv.org/abs/1809.04559.

Bay, H.; Tuytelaars, T.; and Van Gool, L. 2006. Surf: Speeded up robust features. In *European conference on computer vision*, 404–417. Springer.

Blockeel, H.; and De Raedt, L. 1997. Experiments with top-down induction of logical decision trees. Technical report, KU Leuven, Dept. CS.

Cropper, A.; and Muggleton, S. H. 2016. Learning Higher-Order Logic Programs through Abstraction and Invention. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 1418–1424.

Cropper, A.; and Tourret, S. 2018. Derivation Reduction of Metarules in Meta-interpretive Learning. In *International Conference on Inductive Logic Programming*, 1–21.

De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S., eds. 2008. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911. Springer.

Dechter, E.; Malmaud, J.; Adams, R. P.; and Tenenbaum, J. B. 2013. Bootstrap Learning via Modular Concept Discovery. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI, 1302–1309.

Dehaspe, L.; and Toivonen, H. 1999. Discovery of frequent datalog patterns. *Data Mining and knowledge discovery* 3(1): 7–36.

Dosovitskiy, A.; Springenberg, J. T.; Riedmiller, M.; and Brox, T. 2014. Discriminative Unsupervised Feature Learning with Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 27*, 766–774.

Ester, M.; Kriegel, H.-P.; Sander, J.; and Xu, X. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on KDD*, 226–231.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Kramer, S.; Lavrač, N.; and Flach, P. 2001. Propositionalization approaches to relational data mining. In *Relational data mining*, 262–291. Springer.

Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60(2): 91–110.

Manhaeve, R.; Dumančić, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: Neural Probabilistic Logic Programming. In *NeurIPS*.

Mao, J.; Gan, C.; Kohli, P.; Tenenbaum, J. B.; and Wu, J. 2019. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*.

Misra, I.; and van der Maaten, L. 2020. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6707–6717.

Moreno-García, C. F.; Elyan, E.; and Jayne, C. 2018. New trends on digitisation of complex engineering drawings. *Neural Computing and Applications* ISSN 1433-3058.

Muggleton, S. 1995. Inverse entailment and Progol. *New generation computing* 13(3-4): 245–286.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS 2017 Workshop Autodiff*.

Quinlan, J. R.; and Cameron-Jones, R. M. 1993. FOIL: A midterm report. In *European conference on machine learning*, 1–20. Springer.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, 618–626.

Smith, R. 2007. An Overview of the Tesseract OCR Engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, 629–633. ISBN 0-7695-2822-8.

Srinivasan, A. 2001. *The Aleph Manual*. URL http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.

Staar, P. W. J.; Dolfi, M.; Auer, C.; and Bekas, C. 2018. Corpus Conversion Service. *Proceedings of the 24th ACM SIGKDD International Conference on KDD* .

Suzuki, S.; and Abe, K. 1985. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing* 30(1): 32 – 46. ISSN 0734-189X.