An Interactive Explanatory AI System for Industrial Quality Control

Dennis Müller, Michael März, Stephan Scheele, Ute Schmid

Fraunhofer Institute for Integrated Circuits IIS
Sensory Perception & Analytics | Comprehensible AI
{dennis.mueller, michael.maerz, stephan.scheele, ute.schmid}@iis.fraunhofer.de

Abstract

Machine learning based image classification algorithms, such as deep neural network approaches, will be increasingly employed in critical settings such as quality control in industry, where transparency and comprehensibility of decisions are crucial. Therefore, we aim to extend the defect detection task towards an interactive human-in-the-loop approach that allows us to integrate rich background knowledge and the inference of complex relationships going beyond traditional purely data-driven approaches. We propose an approach for an interactive support system for classifications in an industrial quality control setting that combines the advantages of both (explainable) knowledge-driven and data-driven machine learning methods, in particular inductive logic programming and convolutional neural networks, with human expertise and control. The resulting system can assist domain experts with decisions, provide transparent explanations for results, and integrate feedback from users; thus reducing workload for humans while both respecting their expertise and without removing their agency or accountability.

Introduction

Knowledge-driven machine learning methods – in the context of this paper in particular *Inductive Logic Programming* (*ILP*; for an overview, see (Cropper and Dumančić 2020)) – and statistical data-driven methods, particularly deep learning, have complementary advantages and disadvantages: The scope of the former is naturally limited to specific representations of the training data. ILP in particular requires the data to be represented as Horn clauses, the translation to which is non-trivial, often prohibitively difficult, and informed by prior knowledge of a user. On the other hand, ILP needs relatively little data, and the result is itself a set of Horn clauses that are easily explainable and can be verbalized in a manner comprehensible to non-experts. Furthermore, retraining an ILP system is usually relatively fast and cost-effective, making interactive updates to the system feasible to include direct user feedback. We demonstrated the applicability of an interactive ILP system which offers rich verbal explanations in the context of an assistive system for data management (Siebers and Schmid 2019).

In contrast, deep learning has a virtually unlimited scope and can act on nearly any kind of data (text, images, sound), and find and exploit highly complex patterns in an input, without a user needing to extract the assumed-to-be-important information beforehand. As a trade-off, deep learning models are effectively black boxes, offering no justifications or explanations. Moreover, they require huge amounts of data and are costly to train in the first place, making adaptation and incremental refinements to a once trained system much more difficult.

In this paper, we combine the advantages of both methods into a support system for domain experts, exemplary in the context of quality control of industrial components. A first proof-of-concept for the feasibility of this novel type of neural-symbolic integration has been given for classification of blocks world towers combining visual explanations generated by the LIME approach and verbal explanations generated by the ILP system Aleph (Rabold et al. 2019). As a guiding use case for our experiments, we envision a setting where expensive to produce industrial components may contain various kinds of visually recognizable production flaws. Human domain experts are assumed to be able to decide – based on difficult to formalize experiential prior knowledge – whether the flaws in any given component are either acceptable (“ok”) or sufficiently severe for it to be discarded (“defective”).

An intended support system should then 1. present images of such components to a user, 2. detect and highlight the flaws in the image and 3. provide a suggested *classification* and a *textual justification* thereof. The user is then prompted to either accept the output as accurate (“right for the right reasons”), accept the classification but reject the justification (“right for the wrong reasons”) or reject both (“wrong”). The system thus suggests a comprehensible action, but importantly keeps the human in the loop and takes their feedback into account for the next image presented.

Overview

We implement a demonstrator instance of such a system using a data set of 7348 images of cast metal industrial components (Dabhi 2020). Figure 2 shows three example instances from the dataset, (i) without defects, and two with different types of defects: (ii) has a blowhole at the inner rim and (iii) shows abrasions on the outer rim.

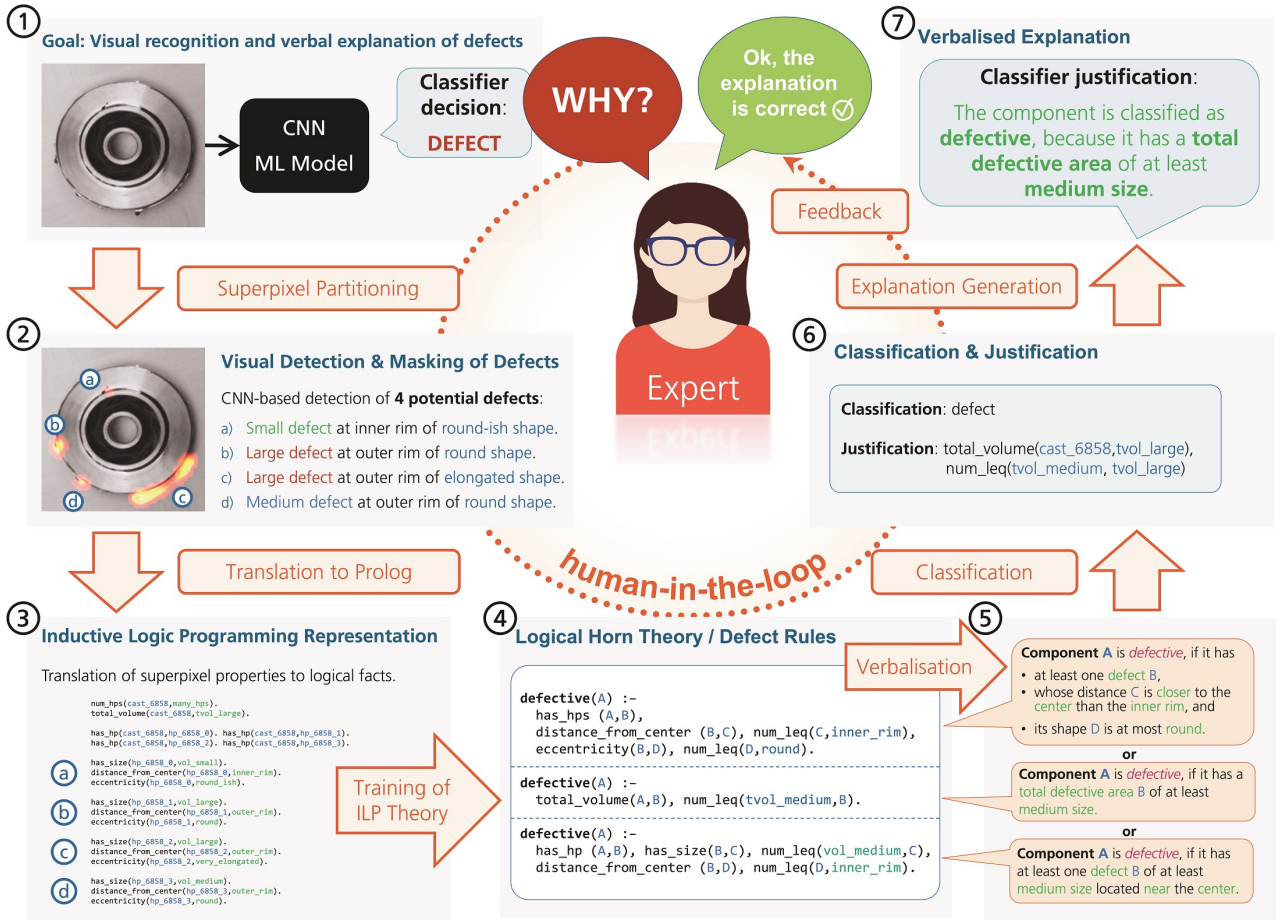

Figure 1: Overview of our approach and use case.

Figure 2: Examples from (Dabhi 2020) from left to right: (i) no defect, (ii) defect at inner rim, (iii) defect at outer rim.

Figure 1 gives an overview of the approach specifically for our exemplary use case implemented by the following seven process steps:

1. First, we train a convolutional neural net (CNN) on the data set using the provided ground truth labels and 86 manually created image masks for defects, using the model from (Božič, Tabernik, and Skočaj 2021). The trained model in particular allows for extracting image masks for the remaining entries of the data set.

   After this step, we can discard the images labelled "ok" (no defects at all) from the data set and relabel the same 86 entries as either tolerable ("ok") or "defective" as training data for the ILP component.
2. We partition the image masks obtained from the ML model into its connected components (*superpixels*, representing the individual located defects) and compute various of their properties (size, location, etc.).
3. We represent each image as a list of Prolog clauses, by associating each type of property from the previous step with a corresponding predicate (see also Listing 1).
4. We train an ILP system (specifically *Aleph* (Srinivasan 2001)) on those Prolog representations of the previously relabelled images (see step 1) to obtain an initial classification hypothesis.
5. We generate a human-readable verbalization as explanation for users.
6. We evalute any given image on the learned clauses from the previous step, keeping track of any contained atom's validity. We thus obtain both a classification as well as (one or multiple) Prolog-style justifications for a clause's applicability (or lack thereof).
7. Finally, we verbalize these justifications for users.

⇒ We can now iteratively present an image with the corresponding mask, classification, and verbalizations to a

user, who is prompted to accept or reject the classification as well as each justification thereof individually. Based on this user feedback, we augment our training data to update the current classification hypothesis.

The resulting system combines the advantages of both knowledge-driven ILP and data-driven deep learning methods, and human expertise: The neural net component allows for extracting relevant information for a subsequent classification from image data; a format that is hardly amenable to alternative approaches. While training this component requires a large amount of data, this data only needs to be classified according to the presence of *any* defects. Such data can usually be obtained relatively easily. More detailed labelling regarding the precise locations and dimensions of individual defects is only necessary for a much lower number of data entries. Additionally, since the network is not used for the final classification, it only needs to be trained exactly once for any new domain of application.

Subsequently, the actual classification is provided by a logic program learned by the ILP component. Logic programs are intrinsically explainable, and a justification for a classification can be generated and presented in human readable form. The non-trivial data representation required for ILP can be generated automatically from the output of the neural component.

Finally, an ILP component can be retrained relatively inexpensively, allowing for integrating human expertise into the system by iteratively updating the ILP data set based on user feedback. In the following, we will describe these components in detail.

The CNN-based Component for Defect Detection

In the first component of the system, we aim to identify potential defects in an image of some given industrial component. The nature of the format of such data (i.e. image files) suggests training a convolutional neural network. We can safely assume that in any industrial setting, large amounts of image data of individual components can be easily obtained and classified as flawed or flawless, so that obtaining a sufficient amount of labelled training data for supervised learning is feasible.

Furthermore, a plurality of methods exist for localizing relevant parts of an image. Such methods fall broadly into one of two classes:

- *Explainable AI (XAI) methods* include e.g. *Lime* (Ribeiro, Singh, and Guestrin 2016), *Layer-Wise Relevance Propagation* (Montavon et al. 2019) and *Grad-CAM* (Selvaraju et al. 2019). These methods attempt to identify the parts of an image that are most relevant for the specific output of a (usually, but not exclusively) neural net classifier.

  Unfortunately, experiments with these methods on our data set have shown that the resulting image maps do not sufficiently reliably highlight the locations of actual defects.
- *Object Detection* models are instead trained directly to recognize, localize and/or classify specific objects in a given image input. These models are primarily used for e.g. identifying people in security camera footage or road signs in dashboard cameras, and have correspondingly received a lot of attention in recent years. Examples include the family of *YOLO*-models (Redmon et al. 2016), variants of *Mask RCNN* (He et al. 2018) and the models included in the *Tensorflow Object Detection API* (Huang et al. 2017).

  Unfortunately, these models often require large amounts of pixelwise-annotated training data, which is difficult and expensive to obtain. While few-shot approaches for object detection exist, these (even more than object detection models in general) suffer from the additional flaw, that they are pretrained on large amounts of highly *heterogeneous* images, with the objects intended to be recognized being largely *homogeneous*. In contrast, data sets such as ours are highly *homogeneous*, whereas the features to be detected (i.e. flaws) are relatively *heterogeneous*, making it substantially more difficult for object detection models to effectively learn to recognize them reliably from little training data.

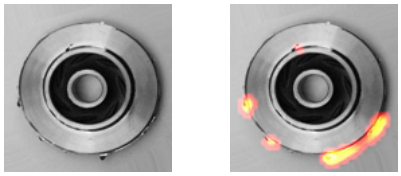

Figure 3: Example input image with its inferred mask.

This problem is also identified by (Božič, Tabernik, and Skočaj 2021) regarding the same domain of application (i.e. defect detection in industrial components), and approached with a novel mixed-supervision model architecture, which trains on large amounts of binary classified training data (has defects / has no defects), combined with a small subset of pixelwise-annotated data. The resulting model outputs such a binary classification as well as an image mask highlighting the locations of defects in the image input.

We consequently use their approach to train a new model on our data set for 20 epochs, augmented by manually annotated image masks for 86 of the training images. The resulting model is able to predict image masks on the 4211 training images with detectable flaws, allowing us to localize their present defects, see Fig. 3. Since this model is ultimately used for generating masks only, training statistics (which primarily relate to classification accuracy) are hardly informative, and hence omitted.

Discussion Which model architecture works best for a given use case depends on the makeup of the data set used. In (Rabold et al. 2019), it consists of simple geometric shapes, for which a vanilla convolutional network and Lime suffice. (Božič, Tabernik, and Skočaj 2021) was specifically introduced for detecting defects in industrial components, so lends itself nicely to our use case, but might fail when the data set is less homogeneous, in which case object detection

models will likely be more suitable.

Additionally, our model only highlights defects, but does not further classify them. Modifications of the model architecture used, or using object detection models instead, could boost the performance of the subsequent ILP component if in addition, the *kinds* of defects can be identified by the statistical model already (e.g. holes, scratches, abrasions, etc.).

The ILP-Based Component for Classification and Justification

Our next goal is to extract relevant information about a given image datum from the image masks obtained from the neural net component, and represent this as a set of Prolog clauses for inductive logic programming.

To that end, we heuristically determine a suitable cut-off value (in our use case 0.3) below which we consider pixel values in an image mask (representing the model's certainty) to be equal to 0. We then determine the fully connected components (i.e. superpixels) of the resulting masks and compute various properties of both the image itself as well as the individual superpixels (each corresponding to a single defect), which we formalize in terms of Prolog predicates.

For our use case, we decide on the following specifics:

- `has_hp`: a binary predicate associating a (name for an) image datum with its respective (names for) superpixel components.
- `has_size`: a binary predicate associating a superpixel with the sum of its mask pixel values, representing its "pixel·certainty mass".
- `distance_from_center`: a binary predicate representing the distance of a superpixel from the center of an image. This approximates the relevant aspects of the location (e.g. inner rim, outer rim) of a superpixel, assuming rotational symmetry and sufficient uniformity across the images (zoom, angle/skew etc.).
- `eccentricity`: a binary predicate associating a superpixel with $\sqrt{1 - b^2/a^2}$, where 1. a is the maximum distance between two pixels a_1, a_2 in the superpixel and 2. b is the shortest distance between two pixels b_1, b_2 in the superpixel such that the line segment between b_1 and b_2 is orthogonal to the line segment between a_1 and a_2.

  This approximates the "roundness" of a superpixel by treating it as an ellipsis, ideally allowing the ILP to distinguish between (round) holes and (less round) scratches or abrasions.
- `num_hps`: a binary predicate associating an image datum with the total number of its superpixels.
- `total_volume`: a binary predicate associating an image with the sum of the volumes of all of its superpixels.

Ideally, we would use the exact numerical values in all of the above predicates for ILP purposes. In practice, it is difficult for ILP systems to learn theories with numerical constants, even if (as in our use case) we can restrict their occurences to upper or lower bounds on variables, and to values occuring in the training data only. We therefore discretize the range of likely values for each of the above predicates using heuristically determined cut-off values and constants representing numerical intervals. For example, the values of the `has_size`-predicate are let to be `vol_small` (for values < 200), `vol_medium` (for values $200 \leq n < 900$) or `vol_large` (otherwise). Additionally, we introduce a binary predicate `num_leq` representing the $\leq$-relation between the thus introduced interval constants.

```
1  total_volume(cast_6858, tvol_large).
2  has_hp(cast_6858, hp_6858_2).
3
4  has_size(hp_6858_2, vol_large).
5  distance_from_center(hp_6858_2, outer_rim).
6  eccentricity(hp_6858_2, very_elongated).
```

Listing 1: Prolog representation of the image from Figure 3.

Listing 1 shows an excerpt of the resulting Prolog representation of the example image used in Figure 3, corresponding to steps 2 and 3 from Fig. 1, using the above predicates and interval constants for numerical values. It states that the image contains a total defective volume of size `tvol_large`, and has a superpixel (i.e. defect) `hp_6858_2` (representing the abrasion at the outer rim) of volume `vol_large`, being located at the `outer_rim` and having an eccentricity of `very_elongated`.

Finally, we classify a subset of our training data (in our case, the same 86 entries we manually annotated with image masks) more-or-less arbitrarily as "ok" or "defective" and use the ILP system *Aleph* (Srinivasan 2001) to induce an initial hypothesis (shown in Figure 1 at step 4). In our experiments, we use a noise value of 10 and a search depth of 50 to obtain a theory consisting of three clauses and an accuracy of 98% in less than ten seconds; a time frame well-suitable for iterative online retraining. Each returned clause represents a candidate for a sufficient condition for the targeted class (in our case: "defective").

Discussion Naturally, which predicates we use strongly depends on the specifics of the use case. While we use Aleph, alternative systems might plausibly be better suited in situations where numerical values should occur as bounds in a proposed hypothesis. Alternatively, clustering algorithms can be used to determine the number and thresholds of interval classes, obviating the need to decide on them manually. We use `distance_from_center` and `eccentricity` as approximations of the relevant locations within the component in the image, and shapes of defects, respectively. In a real-life use case with less homogeneous data, classical image recognition methods are more appropriate for determining such properties, or, as mentioned in Section 1, can possibly be inferred by the preceding ML component already.

Notably, the inferred image masks and the manually annotated image masks differ significantly with respect to the properties used for Prolog predicates; most notably the pixel mass of superpixels: The manually annotated superpixels are precisely localized with sharp boundaries and values either 0 or 1, whereas the inferred superpixels are less precise, more

spread out and have smoother boundaries. If we want to use both types of masks for training an ILP system, some scaling of the associated pixel values is correspondingly necessary to make the computed attribute values adequately consistent across both types.

Finally, there is a choice involved with respect to which classification is used as the *positive* one explained by the learned theory: ILP systems attempt to learn a theory which entails as many *positive* (C^+) examples and as few *negative* (C^-) examples in the data set as possible. This matters insofar as the positive class is then explained as a set of Horn clauses, which together correspond to a *disjunction of existentially quantified conjunctions of non-negated atoms*. The positive class should correspondingly be chosen as the one more likely explainable by such statements. We choose C^+="defective" as positive, since this classification is more likely the result of the *existence* of a defect with certain properties.

The Interactive Component

Having obtained relevant properties of an input image and a first hypothesis for classification, our next goal is to provide a classification and a justification to a user in human-readable form, obtain feedback and integrate the latter back into the training process.

Naturally, Aleph outputs a Prolog theory, so we can use Prolog to obtain classifications based on the learned theory. In that case, however, we only obtain a boolean value. While we can still present a user with a verbalization of the learned theory as justification, it would be preferable to present a case-specific justification instead, instantiated for the exact image currently under consideration.

For that reason, we implement a custom method for evaluating the learned theory on a given image input which, besides evaluating each atom in a clause and instantiating variables with adequate values (e.g. the superpixels in an image), keeps track of whether an atom in a clause evaluates to *true* or *false* under a given variable assignment. While this is naturally less efficient than a dedicated logic programming language, the number of possible values for any given variable and the corresponding search tree are small enough that efficiency is not considerably relevant. As a result however, we obtain a concrete explanation (in the form of a list of true or false atoms) for a given image's classification, as shown in Figure 1 (step 6).

In order to obtain human-readable representations of both the current Prolog theory and the justifications, we need to associate each predicate with a natural language verbalization. This is largely straight-forward, except that we need to consider variable instantiations in *functional* predicates jointly with subsequent atoms. More precisely: binary predicates that evaluate to *true* if and only if the second argument is a uniquely determined value, such as `total_volume` or `has_size`, are representations of *functions* in logic programming languages and (in our case) only occur in conjunction with subsequent predicate applications containing the same value, such as `total_volume(A,B)`, `num_leq(B,tvol_small)`. The corresponding verbalization therefore needs to adequately take into account, that these two atoms conjointly describe a property of *the* (unique) total volume *of* the element `A`. Similar considerations apply to the `has_hp`-predicate, which is used analogously to a non-deterministic function.

Finally, we need to integrate user feedback back into the system. This comes in the form of a boolean value for the classification itself as well as each justification. We distinguish three cases of user acceptance:

1. Both classifications and justifications are accepted as correct; in that case we can simply add the image datum with its inferred classification to the training data.
2. The classification is rejected. This necessarily entails that all justifications are false. We then add the image datum with the opposite label to the training data and retrain the ILP component to obtain an updated hypothesis.
3. The classification is accepted, but (one or several) justifications are rejected ("right for the wrong reasons").

The last case deserves some closer attention:

The "right for the wrong reasons" case means that an explanatory clause E should not entail the associated classification $C^{\pm}$. To ensure that, we can add a new dummy entry D, which should logically entail $\neg\forall x(E[x] \Rightarrow C^{\pm}[x])$ (or equivalently: $\exists x(E[x] \wedge \neg C^{\pm}[x])$) while remaining logically consistent with any alternative theory the ILP system might learn in the absence of D. Notably, ILP systems based on Horn clauses do not use *negations* of atoms or whole clauses.

In the positive case $C^{\pm} = C^+$, the dummy entry should discourage Aleph from learning E as a clause. It then suffices to have D satisfy E and having ground truth label C^-, thus implying that Aleph incurs a penalty iff E entails a clause of the learned theory. By not supplying any additional properties for D, adding D does not otherwise impact Aleph's training procedure negatively.

In the negative case $C^{\pm} = C^-$ however, the only conclusion we can draw is that Aleph's learned theory is not *complete* - which is to be expected, given a limited data set and arbitrarily complex *true* theory. In that case, additional data is required to eventually increase the coverage of the learned theory.

Discussion Some ILP systems allow for *incremental learning* directly, improving a previously learned hypothesis based on new data alone without the need to reprocess the entire training set. While this clearly speeds up the retraining process, in our use case full batch training was sufficiently fast so that this aspect did not require further attention. This will likely become more relevant when scaling the system up to real-world applications.

For use cases where less computationally expensive (e.g. few-shot object detection) models are used, iterative updates to the statistical model can be included, for example via a mask editor. In the ideal case, the model would be updated immediately online, more realistically, it could be retrained periodically (e.g. nightly).

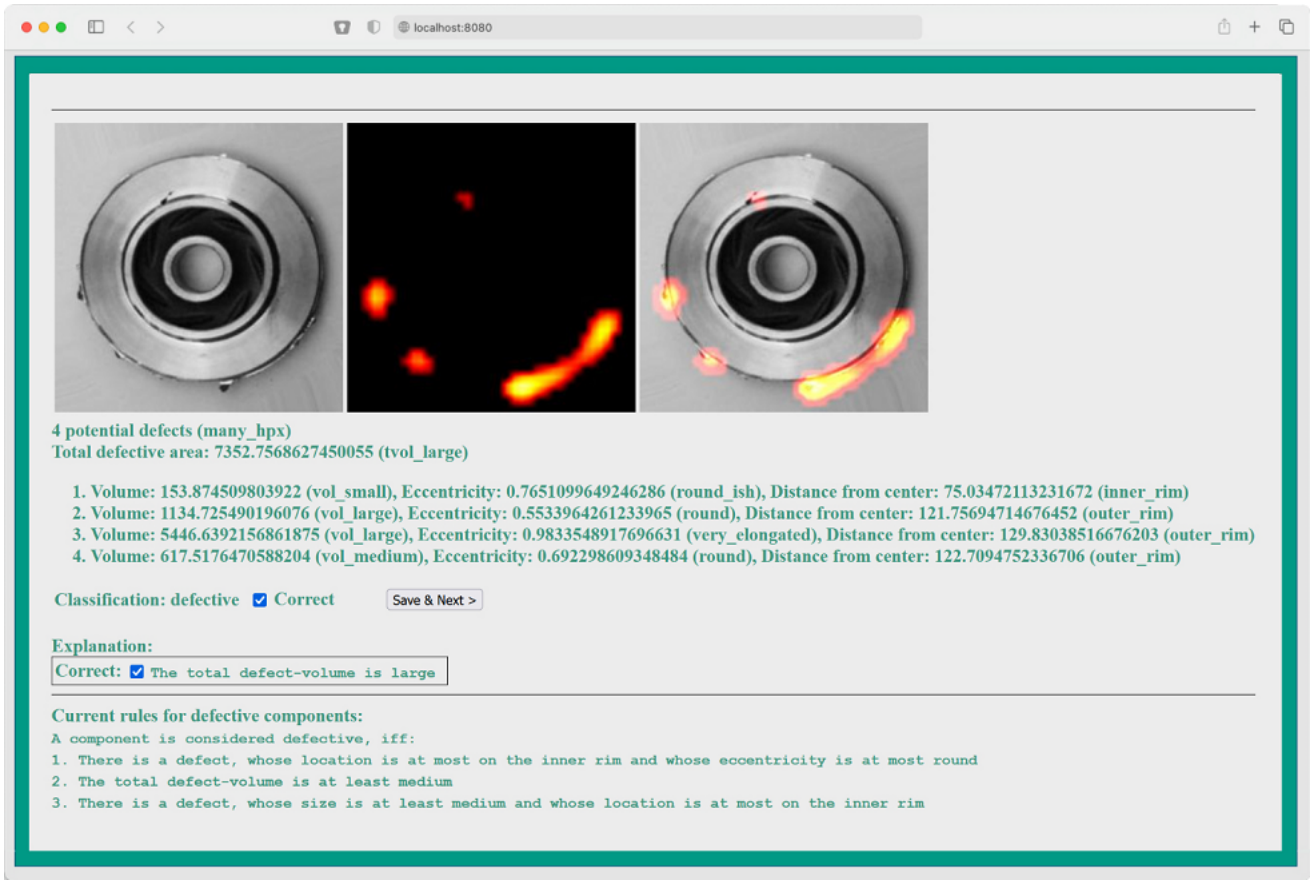

Figure 4: A screenshot of the demonstrator user interface.

We mentioned in Section that other candidate models for the CNN component can potentially be used to also classify the *kinds* of defects (scratch, abrasion, etc.) occurring in some input image. Alternatively, we can have the ILP component learn predicates for these kinds of defects, which can be subsequently used in the ILP training data. While the final theory learned should be logically equivalent, using the new predicates in the training data would lead to more informative descriptions and justifications given to a user. Additionally, it would allow for more detailed user-feedback, in that the classifications of the individual defects (and the justifications thereof) could be accepted/rejected in the exact same manner as the final classification and their justifications.

Demonstrator User Interface

Figure 4 shows a screenshot of the user interface for our implementation. It uses a python-based web framework, serving a web page accessible via a browser. The original image and the image mask are shown both separately and overlayed, with verbalizations of the defects, the current theory (bottom of the screenshot), the classification and explanations for the classification shown. The classification and the explanations have check boxes that a user can uncheck to indicate their invalidity. Clicking on the "Save & Next"-button registers the user feedback, (optionally) updates the knowledge base and displays the data for the next image.

The code for our demonstrator and a link to a publicly running instance will be made available at https://gitlab.cc-asp.fraunhofer.de/sees/iaai-cai-2021.

Conclusion and Future Work

We have presented a proof-of-concept of an interactive AI-driven support system for industrial quality control, that extends visual defect recognition with verbal explanations by combining deep learning with inductive logic programming. Besides the possible improvements and design choices discussed in the sections on the individual components, we plan to extend the approach to other use cases. Both performance and general usefulness of the system can be improved by including more background knowledge, possibly on the basis of more detailed ontologies. Additionally, the explanatory component can be extended by and combined with various other approaches, for example *contrastive* and other example based explanations (Rabold, Siebers, and Schmid 2021). Finally, we plan to evaluate our approach in terms of a user study with respect to applicability and usefulness of explanations in the manufacturing domain.

References

Božič, J.; Tabernik, D.; and Skočaj, D. 2021. Mixed supervision for surface-defect detection: From weakly to fully supervised learning. *Computers in Industry*, 129: 103459.

Cropper, A.; and Dumančić, S. 2020. Inductive logic programming at 30: a new introduction. arXiv:2008.07912.

Dabhi, R. 2020. casting product image data for quality inspection. https://www.kaggle.com/ravirajsinh45/real-life-industrial-dataset-of-casting-product. Accessed: 2021-09-07.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2018. Mask R-CNN. arXiv:1703.06870.

Huang, J.; Rathod, V.; Sun, C.; Zhu, M.; Korattikara, A.; Fathi, A.; Fischer, I.; Wojna, Z.; Song, Y.; Guadarrama, S.; and Murphy, K. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. arXiv:1611.10012.

Montavon, G.; Binder, A.; Lapuschkin, S.; Samek, W.; and Müller, K.-R. 2019. *Layer-Wise Relevance Propagation: An Overview*, 193–209. Cham: Springer International Publishing. ISBN 978-3-030-28954-6.

Rabold, J.; Deininger, H.; Siebers, M.; and Schmid, U. 2019. Enriching Visual with Verbal Explanations for Relational Concepts - Combining LIME with Aleph. In Cellier, P.; and Driessens, K., eds., *Machine Learning and Knowledge Discovery in Databases - International Workshops of ECML PKDD 2019, Würzburg, Germany, Proceedings, Part I*, volume 1167 of *Communications in Computer and Information Science*, 180–192. Springer.

Rabold, J.; Siebers, M.; and Schmid, U. 2021. Generating contrastive explanations for inductive logic programming based on a near miss approach. *Machine Learning*, https://doi.org/10.1007/s10994-021-06048-w.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 1135–1144.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2019. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, 128(2): 336–359.

Siebers, M.; and Schmid, U. 2019. Please delete that! Why should I? *KI - Künstliche Intelligenz (German Journal of Artificial Intelligence)*, 33(1): 35–44.

Srinivasan, A. 2001. The Aleph Manual. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html. Accessed: 2021-09-08.