# Information extraction from operator interface images using computer vision and machine learning

Eirik Illing [a,*], Nils-Olav Skeie [b], Ole Magnus Brastein [c]

[a] *Emerson Automation Solutions,* [b c] *University of South-Eastern Norway (USN)*
eirik.illing@emerson.com

**Abstract**
In the process of system upgrades or migrations, the utilization of existing layouts and object structures for designing new Human Machine Interfaces (HMI) can significantly save time and effort. Operator interface images, commonly referred to as HMI´s, contain valuable information crucial to industrial operations, but access to source code or design files can be limited. Modern frameworks for object detection and text recognition offer a solution by extracting information directly from images. However, these methods require time-consuming data acquisition and manual effort to initiate. This paper proposes a novel approach utilizing traditional Computer Vision (CV) and Machine Learning (ML) techniques to extract objects from images. The extracted objects are used as training data to transfer learn a ResNet model for multi-label image classification. The combination of this model with techniques such as sliding window, pyramid scaling, and non-maximum suppression forms the basis for a semi-automated annotation tool. This tool generates training data for more optimized object detection methods, specifically the YOLO (You Only Look Once) one-stage object detector. The semi-automated annotation tool allows engineers to manually refine the training data and export state-of-the-art training images for YOLO. The YOLO model achieves an impressive mean Average Precision at IoU 50% ($mAP^{50}$) score of 95.5% when transfer learned on the annotated data. Additionally, an Optical Character Recognition (OCR) engine is utilized to extract text information from preprocessed images, followed by postprocessing to filter tag data. An algorithm is then employed to link objects and tags together. The final solution is implemented in software designed to optimize user interaction, resulting in an analysis document in Excel format, which can be easily exported for end-user access. With the novel use of this software to automate image analysis, the time required to analyze HMI images prior to migration or rebuild can be reduced by an estimate of 90%.

## 1. Introduction

The rapid advancement of technology has led to an increasing reliance on operator interface images, such as HMI [1] and Supervisory Control and Data Acquisition [2] (SCADA) graphics**,** in various industries. As the field advances the frameworks for these interface technologies evolves, new and improved design concepts are introduced, and migration from old systems to new become a necessity. These traditional operator interface images contain a wealth of valuable information related to production, process flows, and assembly lines. However, accessing the underlying source code or design files of these operator interface images can often be challenging or limited. To address this issue, modern frameworks for image classification and object detection have emerged as potential solutions, enabling the extraction of pertinent information directly from these operator interface images.

This paper explores the field of image classification [3] and object detection [4] for the purpose of extracting information from complex operator interface images. The primary objective is to develop a tool that can effectively analyze and interpret industrial applications depicted in these images. Specifically, the project will be conducted in two iterations, each with distinct goals and outcomes.

### *1.2. Previous work*

Several studies have investigated the recognition and extraction of information from industry related documentation, particularly in the context of Piping and Instrumentation Diagrams (P&IDs). Paliwal et al. [5] proposed a method in 2021 that utilized a Dynamic Graph Convolutional Neural Network (DGCNN) to recognize line-drawn symbols in P&IDs. Their approach involved constructing a graph based on sampled pixels along contour boundaries and incorporating ResNet-34 embeddings to improve classification accuracy. They employed an Arcface loss function to address misclassification issues caused by similar-looking objects [6]. The presented research utilizes a single image for each object, setting it apart from conventional approaches that typically require multiple images for training.

In an earlier paper, Paliwal et al. [7] developed an end-to-end data extraction system for P&IDs using fully convolutional networks. This involved annotating multiple training images with segmented pixels to identify different symbol classes. The authors employed a pipeline approach, separating text extraction and graphic object

detection, and used minimum Euclidean distance to link text and objects. Their system achieved effective information extraction and demonstrated the potential of performing extraction in multiple steps or iterations.

Moon et al. [8], proposed a three-step method for recognizing line objects and flow arrows in image-format P&IDs. Their approach involved removing outer borders and title boxes (considered noise), then detecting continuous lines, line signs, and flow arrows, and adjusting and merging lines accordingly. They employed preprocessing techniques to remove noise, applied thinning and pixel processing for line detection, and utilized a RetinaNet model to train on the line signs and flow arrows.

These studies have made significant contributions to the field of analyzing documentation, specifically in addressing challenges related to object recognition, noise reduction, and information extraction. These studies primarily focused on analyzing documentation in grayscale, predominantly using standardized symbols and texts. In contrast, the current project aims to tackle similar challenges while dealing with operator interface images that exhibit a wide variety of complexities, such as color variations, scales, and limited features, requiring different approaches for object recognition and information extraction.

### *1.3. Outline of paper*

System Description chapter explains how the project was executed, including the system overview and associated advantages and challenges.

Methods chapter covers data collection, training models, and tool development in detail.

Results and Discussion chapter present project outcomes, including model performance, effectiveness of data annotation tools, and discussions regarding the aforementioned topics.

Future Work chapter explores improvement areas, use cases, and opportunities for further development.

Conclusion chapter summarizes key findings and highlights the project's significance and potential impact.

This paper is based on a Master's Thesis project [9] conducted at USN and supported by Emerson Automation Solutions.

## 2. System description

### *2.1. Project execution*

In the first iteration, the project will focus on training a ResNet50 [10] image classification model. The model will be trained on custom data extracted from existing operator interface images using CV [11-12] techniques, manually sorted into class folders for data labeling [13]. This phase aims to explore different model configurations and assess their performance in accurately classifying industrial objects. An annotation tool is developed to aid in the labeling of training data. This tool will streamline the annotation process and lay the foundation for subsequent iterations.

The second iteration will leverage the annotated data generated from the previous phase to train a one-stage [14] YOLOv8 [15-16] model for object detection. The goal is to accurately identify and locate objects within the operator interface images. Furthermore, text extraction techniques such as Pytesseract OCR [17] will be employed to retrieve textual information from these images, which will then be linked to the respective objects detected using Minimum Euclidean Distance calculations. The text extraction will require extensive image preprocessing to ensure high acquisition accuracy, and a custom text-format library is embedded in the software to filter relevant information. To ensure usability and accessibility, the solutions developed in this project will be combined into one software solution tool that is accessible through a web interface. All development is done using Python and the Flask web framework.

Users will be able to upload operator interface images to the tool, which will then process the images and generate an analysis document as output. This document will provide a representation of the information extracted from the images, facilitating informed decision-making and analysis for industrial applications. To illustrate the benefits and advancements offered, the flowchart provided in Figure 1 compares the proposed analysis process with conventional approaches involving source-code tools or manual analysis.

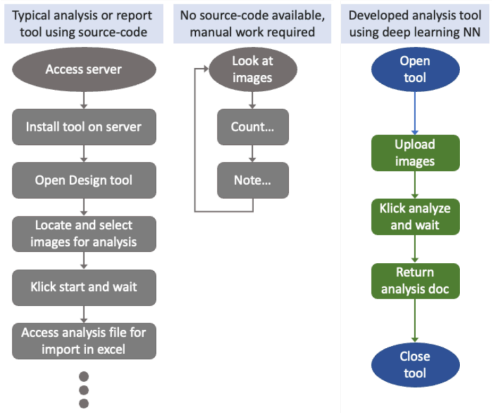


Figure 1: Comparing conventional approaches using source-code tools or manual analysis with the object detection tool suggested in this research.

### *2.2. Advantages*

Performing object detection on computer drawn images, such as drawings and documentations, is in

some ways easier than real-world images. The traditional challenges associated with factors [18] like lighting conditions, object angles, line of sight, dirt, and other real-life variables are non-existent in these 2D images. However, these types of images also present unique challenges that need to be addressed.

### *2.3. Challenges*

Operator interface images contain a large number of objects, lines, and text that represent various types of information. This abundance of visual elements introduces challenges related to noisiness and limited features. Due to the similarities between objects, there is a higher probability of misclassification. In general, there is a lack of true standardization in both object and image design, as well as tag structure within these images.

## 3. Methods

### *3.1. Data collection*

If source-code or design files is unavailable, extracting training data objects directly from raw image files may be necessary. To streamline this process, a Python script is developed for object extraction, eliminating the need for manual snipping tool usage. OpenCV [19] offers various methods simplifying the extraction of training, validation, and test set objects from raw operator interface images.

The script will convert input images to grayscale, apply thresholding and dilation, and identify contours using the "find contours" method in OpenCV**.** These contours are then enclosed in bounding boxes using a different method from the OpenCV library. Each bounding box represented an object and is snipped from the full-scale image into a separate folder. While some of these objects are suitable for training, validation, and test sets, others are incomplete or contain noise. Although this method is not perfect, it provides a foundation for the subsequent manual sorting of objects into class folders for labeling purposes.

Following the extraction of individual objects from the full-scale images using the Python script, a manual process is undertaken to organize and label the extracted objects. This involves moving the extracted objects into separate folders, with each folder representing a different object class. By separating the objects into distinct folders, it becomes simpler to manage and track the defined classes. Additionally, labeling is achieved by associating each object within a class folder with its corresponding label. While going through the extracted data it is important to check that the objects put into class folders represents the data that the model needs to learn. An example of a clear represented object with the label “valve_p” for valve pneumatic is shown in Figure 2.

Figure 2: Clean pneumatic valve object.

The objects are separated into single-label and multi-label folders. In the single-label folders, each folder is named according to a single class label. In the multi-label folders, the folder names consist of all class labels representing the labels of the objects contained within, separated by spaces.

The multi-label classifier requires the data converted from this folder structure into a single folder with a specification file defining name, label and validation set. This is achieved through a Python script which also randomly choose 20% of objects from each class as validation data. An example of the comma-separated values (CSV) specification file is shown in Table 1. The entire data collection process is summarized in Figure 3.

Table 1: CSV specification file format for multi-label classification. 20% validation (validation column “yes”).

| fname | label | validation |
|---|---|---|
| object1.png | pump | no |
| object2.png | valve | yes |
| object3.png | valve | no |

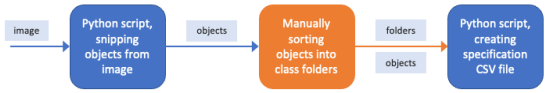


Figure 3: Data collection and preparation for training image classification models.

Data for object detection is attained through image annotation and will be a resulting part of the developed semi-automated annotation tool.

### *3.2. Image classification*

In this project, the primary focus for the application of single-label classification is to assess the performance improvement of a pretrained ResNet50 model. The objective is to evaluate the effectiveness of transfer learning with custom data in comparison to training a fresh model. For more information on this topic, please refer to the Master’s thesis [9]. The single-label classifiers used in this study were trained on approximately 1000 object images.

In the future task of classifying objects within a region of interest (RoI), the utilization of multi-label classification with the ResNet50 model proves to be advantageous. This approach enables the classifier to detect multiple objects within the RoI, while also taking into account situations where no objects are present, thus minimizing the occurrence of misclassifications. The multi-label classifier is trained on approximately 1400 object images, where 1000 of these are single-label object images. It is important to ensure that for all multi-label object classes, there exist good representative

single-label object classes. For instance, if a multi-label object image contains both a valve and a line, it is necessary to have separate single-label object images for both the valve and the line classes. This approach enables the model to effectively distinguish between different objects and enhance its classification capabilities.

*3.3. Semi-automated annotation*

Combining three computer vision techniques: sliding window, image pyramid scaling, and Non-Maximum Suppression (NMS), to extract objects from an image. The sliding window divides the image into overlapping windows, generating potential RoI's. The image pyramid scaling ensures object detection at different scales. NMS eliminates redundant detections, selecting the most accurate bounding boxes. Extracted image snippets are classified using a multi-label classification model derived in the previous step. This approach serves as the foundation for the pre-analysis stage in the development of a semi-automated annotation tool.

The requirements and functionality desired for this tool include the ability to upload images, perform preprocessing, conduct pre-analysis using the above-mentioned method, manually adjust the pre-analysis annotations, and export the annotated data in a format suitable for one-stage or two-stage object detectors. Specifically, the tool should provide the option to export the annotations in a text file format compatible with the YOLO detector.

*3.4. YOLOv8 object detector*

The annotation process for generating training and validation data for modern object detectors has now been optimized. It is well-established that one-stage detectors are faster but often yield lower accuracy compared to two-stage detectors [20]. At the time of undertaking this project, a new one-stage detector for YOLO was introduced, promising improved mAP scores and better results on tiny objects. Considering the goal of analyzing a large volume of images in a single run, speed is crucial. Consequently, the latest YOLOv8 architecture provided by Ultralytics [21] have been chosen.

Ultralytics provides different sizes of their model. As a general guideline, larger models are capable of capturing more features than smaller ones [22-23]. After checking this guideline by evaluating the medium, large, and extra-large models using the same dataset, the largest model was selected. This testing process is listed in Table 2.

Table 2: Model sizes tested using custom dataset of 21 train and 3 validation images. 1000 epochs of training.

| Model | Early stopping | $mAP^{50}$ |
|---|---|---|
| medium | 270 epochs | 80.8% |
| large | 388 epochs | 83.0% |
| xlarge | 326 epochs | 90.5% |

*3.5. Extracting tags*

OCR involves preprocessing the image, localizing text, character segmentation and recognition, and post-processing. The focus is on utilizing the recognition part of the Pytesseract OCR library, excluding post-processing dictionary translation. The tags in operator interface images consist of combinations of numbers, letters, and symbols. To filter out unwanted combinations, a custom dictionary is created. The recognized tags and their positions relative to the image are stored in a text file, with positions normalized between 0 and 1. Initially, OCR on the raw image did not provide valuable information due to small tag sizes, random placement, and low contrast. To address this, the image is divided/split into sections, a scale pyramid is applied, and preprocessing steps such as grayscale conversion, blurring, edge detection, and dilation are performed as shown in Figure 4. This significantly improves the OCR results as shown in Table 3.

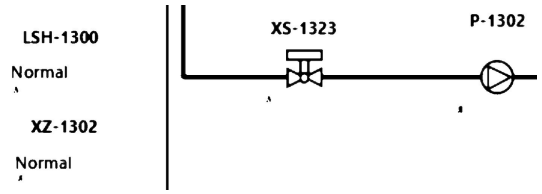


Figure 4: Preprocessing images to only identify tags and high contrast lines.

Table 3: Showing a small part of OCR result before and after preprocessing the image.

| No image preprocessing | Custom image preprocessing |
|---|---|
| ***2341*** *0.090365 0.093981 0.003385 0.002315* | ***LSH-1300*** *0.089461 0.689971 0.030979 0.020058* |
| ***0201*** *0.533594 0.083912 0.004687 0.002546* | ***XS-1323*** *0.183917 0.687645 0.025043 0.020058* |
| ***0201*** *0.611393 0.083796 0.003255 0.001852* | ***P-1302*** *0.258107 0.683430 0.019687 0.023256* |

*3.6. Linking objects and tags*

The concept behind tag extraction in conjunction with object detection is to establish a relationship between tags and objects based on their respective locations. It is reasonable to assume that tags and objects located close to each other are associated. However, there are certain arguments against this generalization. For instance, tags may be situated far from an object due to unobservable status variables associated with the object in the current image state. This distance may even exceed the distance between the tag and an unrelated object. Consequently, in this scenario, a single tag can be linked to multiple objects.

As demonstrated in the previous step, the OCR engine extracts texts and converts their positions to a normalized scale ranging from 0 to 1, matching the YOLOv8 object detection location scale. To determine the distance between the center of the tag location and the object location, the Minimum

Euclidean Distance calculation is employed. Visual representation of the distance calculation is shown in Figure 5.

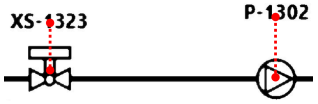


Figure 5: Visualized linking of object and tags.

*3.7. ICE - Industrial Component Extraction tool*

An analysis software that encompasses all the components, including the object detector, OCR and minimum Euclidean distance calculator, is created. Additionally, the software will provide a defined export format that allows users to easily view the final analysis. The final solution is structured as shown in the use case diagram in Figure 6.

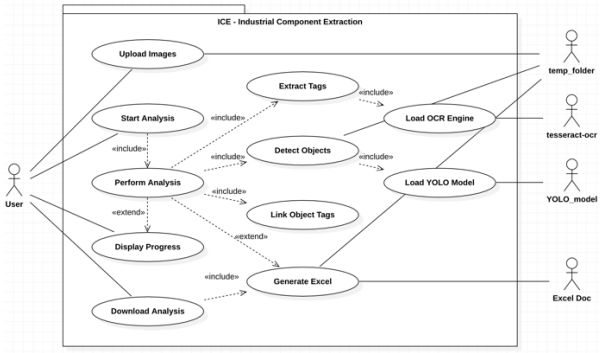


Figure 6: Use Case Diagram for final analysis software ICE – Industrial Component Extraction.

## 4. Results and Discussion

*4.1. Multi-label image classification*

A learner is defined, with minor data augmentation such as vertical and horizontal flipping, as well as zero-padding. Since the data never will be warped, and all information within the image is relevant, no further augmentation is needed. Figure 7 displays a sample from a training data minibatch.

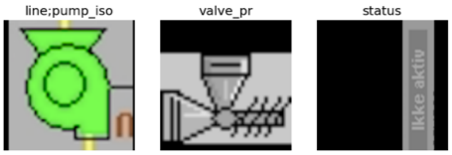


Figure 7: Sample of a minibatch.

When dealing with multi-label image classification, it is crucial to establish an appropriate multi-accuracy threshold. A threshold value of 0.8 is selected which is heigh and within a smoothness of FastAI's [24] threshold finders' curve for this function thus ensuring no outliers are selected [25].

After finetuning the ResNet50 model for 11 epochs (4 freeze and 7 un-freeze), an accuracy of 99.33% is achieved. By analyzing the loss plot shown in Figure 8 both the validation- and training-loss flattens. There is no indication of overfitting, so a score of 99.33% is acceptable and training is stopped after the 11th epoch.

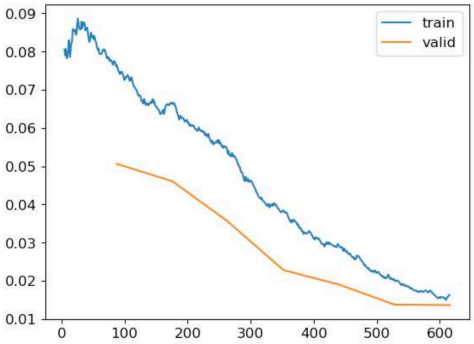


Figure 8: Multi-label classification loss plot.

*4.2. Semi-automated annotation tool*

The multi-label classifier in combination with pyramid scaling, sliding window and soft NMS results in a multi-class object detector. A customized version of the soft NMS is required to only allow same type labels to suppress each other, shown in Equation (1).

$$J_{label}(A_{label}, B_{label}) = \frac{|A_{label} \cap B_{label}|}{|A_{label} \cup B_{label}|} \quad (1)$$

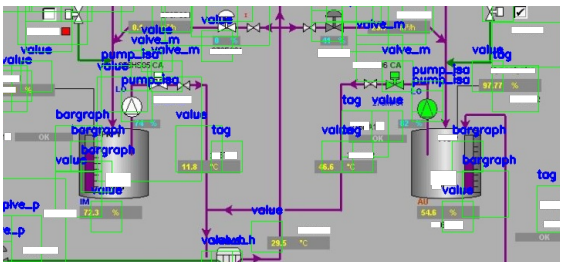

Figure 9: Small extract (snippet) of the multi-class object detector result.

From the image displayed in Figure 9, this method appears sub-optimal as it scores poorly on both position and classification. The resulting custom $mAP^{50}$ calculation score of 5 individual images was 11.36%. More on how this custom accuracy calculation was performed in Master's thesis [9]. This method provides a reference to the object and its location, serving the purpose as a pre-analysis step for the semi-automated annotation tool. The goal is to enhance efficiency in annotating data for modern object detectors.

The annotation tool is developed as specified by the requirements, where a user can upload an image, pre-analyze it using the multi-class object detector with the multi-label classifier model trained in chapter 4.1, and further make modifications and improve the annotation. The user can now export a state-of-the-art annotation file for the object detector. It is estimated an 75% increased efficiency using this tool compared to traditional third-party tool due to the pre-analysis which

provides classes and a starting point for the user to annotate.

The tool was later modified to also take pre-annotated images, if existing. Which would give a good starting point of annotation (no pre-analysis needed by multi-class object detector).

*4.3. YOLOv8 object detector*

To prevent aliasing due to downscaling when loading the training and validation datasets to the object detector, the image and annotation data was split with a custom script. Thus, increasing the training data. This resulted in a better model.

At the final iteration, the YOLOv8 model was transfer learned on 59 training and 11 validation images from three different sites to improve generalization. An overview of the training iterations can be seen in Table 4.

Table 4: Iterations of training and validating the YOLOv8 transfer learned model. All runs are performed with parameters: patience=150, batch=8, model=xlarge.

| Runs | Dataset | N Sites | mAP$^{50}$ | Note |
|---|---|---|---|---|
| 1 | 21 train, 3 val | 1 | 90.5% | Tag classes included |
| 2 | 21 train, 3 val | 1 | 87.1% | Removed tag classes, fixed some errors |
| 3 | 40 train, 8 val | 1 | 97.2% | Realized non-generalized model |
| 4 | 59 train, 11 val | 3 | 95.5% | Added more data from different sites |

Since the object detector returns an annotation file during testing on new data, sub-optimal tests that detect only a small percentage of objects can be fed back to the semi-automated annotation tool for improvement. Consequently, new test images are transformed into training and validation images, necessitating the need for more data. This created the idea of modifying the annotation tool to include the YOLOv8 object detector as a pre-analysis step instead of the multi-class object detector.

In summary, the misclassifications of the YOLOv8 model shown to the left in Figure 10 is strongly related to number of representatives in the dataset shown to the right in Figure 10.

*4.4. ICE - Industrial Component Extraction tool*

Combining the custom YOLOv8 object detector model with OCR and linking objects and tags leads to the development of effective analysis software. The software is designed based on the requirements and use case diagram depicted in Figure 6. The resulting software exhibits an estimated improvement in efficiency, approximately 10 times greater than manual analysis.

The Pytesseract OCR engine only extracts approximately 50% of tags due to non-customized model (depending on image to text size ratio). OCR image preprocessing is also the most time-consuming part of the analysis. Analyzing 12 images takes 1 minute and 20 seconds, where the object detection only uses approximately 25ms (on average) for each image. Image preprocessing and OCR account for most of the remaining time required.

The analysis results are exported in an Excel document format, which includes a summary sheet as the first page, providing an overview of the analyzed images and the detected objects in each image. Additionally, separate sheets are generated for each image, displaying the bounding box objects with labels within the images, shown in Figure 11. Moreover, a data sheet for each image is

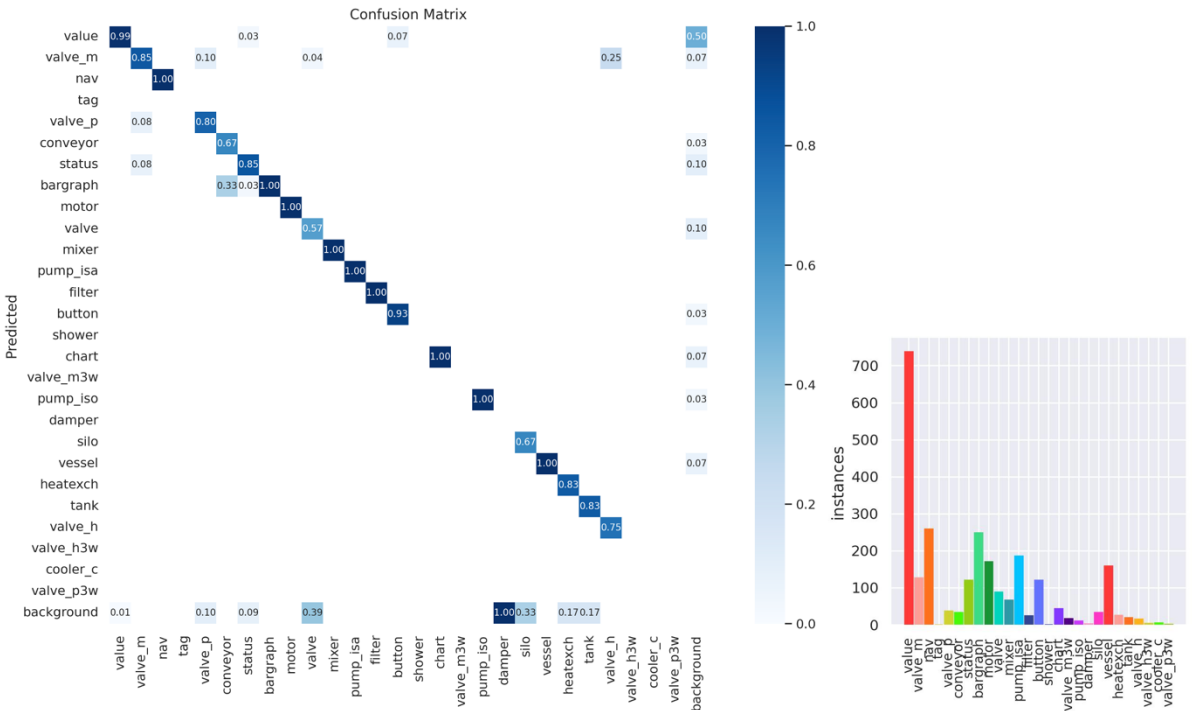


Figure 10: YOLOv8 confusion matrix (left) and number of instances (right), run 4.

included, showing each detected object snipping along with its associated tag and information in a table format, shown in Figure 12.

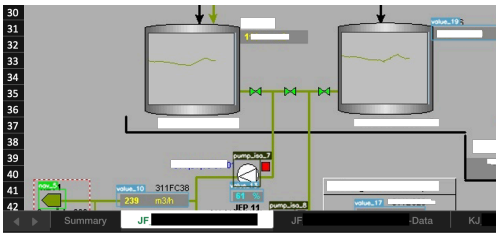


Figure 11: Individual image analysis sheet with bounding boxes.

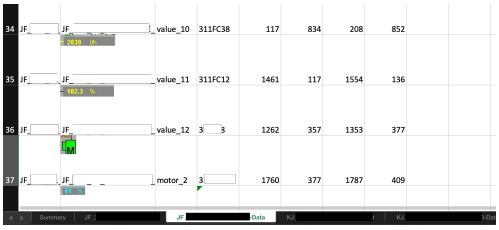


Figure 12: Adjacent image data sheet with information.

## 5. Future Work

### *5.1. Improvements to current solutions*

The OCR tag extraction method used in this project can be improved by exploring alternative extraction methods or adding tags as a separate label class in the object detector. Treating tags as a separate class would require more training data but could improve text extraction by individually feeding tag objects to the OCR engine. The poor performance in the current solution is due to the small size and random placement of text compared to other objects. Multiple image scales and preprocessing were used to improve detection, but this increases computing power and is time consuming.

To enhance the final ICE software, incorporating features from the annotation tool would be beneficial. Users could perform pre-analysis on a portion of customer images using the YOLOv8 model, make manual adjustments to the detection, and retrain the model. Then upload the rest of the images and get an improved analysis with the improved model. This iterative process improves model generalization and user experience. It would be valuable to have these features available for all users of the ICE software for daily image analysis.

Including more annotated images from various sites would enhance the model's performance and generalization. Alternatively, modifying the YOLOv8 network architecture by substituting the classification network with the multi-label image classification model could eliminate the need for annotating more images.

### *5.2. Opportunities for future development*

Combining the developed product with pixel processing for pipeline detections, as discussed by Moon et al. [8], could provide a solution for documenting structured image flows. Further training a large language model (LLM) on the source-code libraries for operator interface image designs would enable features for mapping of detected objects to generate prompts that results in automatically generating code for new images. Extending this concept to configuration and documentation such as system control diagrams (SCDs), and P&IDs, by training the LLM on relevant data, and improving models for detection and text extraction, could automate the process of redesign/migrating operator interface images entirely.

A solid object detection model for process graphics can also monitor real-time system images and extract information without direct system logic interaction. For instance, it can be useful in situations where integration with communication protocols is not feasible. As an example, placing a web camera in front of old HMI panels and extracting data to a cloud solution offers a solution when integration or modernization is not an option.

These advancements hold potential for diverse applications, improving efficiency in challenging environments.

## 6. Conclusion

In this project, an object detection system was developed for operator interface images, with a focus on optimizing data acquisition, annotation, model training, and software integration. The approach involved a semi-automated annotation tool that utilized multi-label classification and traditional computer vision techniques, resulting in an estimated 75% efficiency improvement compared to traditional tools. The tool supports both two-stage and one-stage detectors, allowing manual adjustments to analysis and exporting annotations in popular formats.

Next, the utilization of the YOLOv8 model from Ultralytics was explored, and it was trained with custom data generated using the semi-automated annotation tool. After multiple iterations and preprocessing techniques on 70 images, a $mAP^{50}$ score of 95.5% was achieved. The final model was then integrated into a user-friendly web application that enables users to upload images, perform analyses, and obtain downloadable results in an Excel format. This tool streamlines project planning, improves efficiency, and facilitates cost estimation for migration projects. Estimating a reduction of time spent analyzing HMI by 90% compared to the manual approach.

The project successfully established a novel foundation for object detection in operator interface images, providing an efficient semi-automated annotation tool and a high-performing YOLOv8 model. The developed software application has the

potential to enhance project planning efficiency and accuracy, benefiting various industries. However, it is essential to emphasize the need for thorough data collection and testing to ensure the accuracy and generalizability of the model.

**Acknowledgment**
Thanks to industrial partner and employer Emerson Automation Solutions.

## References

[1] C. C. Editor, "Human-Machine Interface (HMI) - Glossary | CSRC." https://csrc.nist.gov/glossary/term/human_machine_interface (accessed Jun. 04, 2023).
[2] Inductive Automation, "What is SCADA?," *Inductive Automation*. http://www.inductiveautomation.com/resources/article/what-is-scada (accessed May 05, 2023).
[3] K. Balaji and K. Lavanya, "Image Classification - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/engineering/image-classification (accessed May 08, 2023).
[4] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey." arXiv, Jan. 18, 2023. Accessed: Mar. 23, 2023. [Online]. Available: http://arxiv.org/abs/1905.05055
[5] S. Paliwal, M. Sharma, and L. Vig, *OSSR-PID: One-Shot Symbol Recognition in P&ID Sheets using Path Sampling and GCN*. 2021.
[6] J. Deng, J. Guo, J. Yang, N. Xue, I. Kotsia, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 5962–5979, Oct. 2022, doi: 10.1109/TPAMI.2021.3087709.
[7] R. Rahul, S. Paliwal, M. Sharma, and L. Vig, "Automatic Information Extraction from Piping and Instrumentation Diagrams:," in *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2019, pp. 163–172. doi: 10.5220/0007376401630172.
[8] Y. Moon, J. Lee, D. Mun, and S. Lim, "Deep Learning-Based Method to Recognize Line Objects and Flow Arrows from Image-Format Piping and Instrumentation Diagrams for Digitization," *Applied Sciences*, vol. 11, no. 21, Art. no. 21, Jan. 2021, doi: 10.3390/app112110054.
[9] E. Illing, "Object detection, information extraction and analysis of operator interface images using computer vision and machine learning.," Masters Thesis, University of South-Eastern Norway, Porsgrunn, Norway, 2023.
[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." arXiv, Dec. 10, 2015. Accessed: Mar. 29, 2023. [Online]. Available: http://arxiv.org/abs/1512.03385
[11] V. Wiley and T. Lucas, "Computer Vision and Image Processing: A Paper Review," *International Journal of Artificial Intelligence Research*, vol. 2, p. 22, Feb. 2018, doi: 10.29099/ijair.v2i1.42.
[12] Nichole Peterson, "History of Computer Vision and Its Principles | alwaysAI Blog | alwaysAI." https://www.alwaysai.co/blog/history-computer-vision-principles (accessed Mar. 23, 2023).
[13] IBM, "What is Data Labeling? | IBM." https://www.ibm.com/topics/data-labeling (accessed Jun. 04, 2023).
[14] Jeremy Jordan, "An overview of object detection: one-stage methods.," *Jeremy Jordan*, Jul. 11, 2018. https://www.jeremyjordan.me/object-detection-one-stage/ (accessed Jun. 04, 2023).
[15] Sovit Rath, "YOLOv8 Ultralytics: State-of-the-Art YOLO Models," Jan. 10, 2023. https://learnopencv.com/ultralytics-yolov8/ (accessed Mar. 24, 2023).
[16] J. Solawetz, F. JAN 11, and 2023 10 Min Read, "What is YOLOv8? The Ultimate Guide.," *Roboflow Blog*, Jan. 11, 2023. https://blog.roboflow.com/whats-new-in-yolov8/ (accessed Mar. 24, 2023).
[17] Zelic Filip and Anuj Sable, "Tesseract OCR in Python with Pytesseract & OpenCV," *Nanonets AI & Machine Learning Blog*, Aug. 09, 2022. https://nanonets.com/blog/ocr-with-tesseract/ (accessed Feb. 23, 2023).
[18] exposit_marketing, "Computer Vision Object Detection: challenges faced," *Exposit*, Apr. 20, 2021. https://www.exposit.com/blog/computer-vision-object-detection-challenges-faced/ (accessed Jun. 04, 2023).
[19] T. OpenCV, "About," *OpenCV*. https://opencv.org/about/ (accessed Feb. 23, 2023).
[20] X. Lu, Q. Li, B. Li, and J. Yan, "MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., in Lecture Notes in Computer Science, vol. 12359. Cham: Springer International Publishing, 2020, pp. 541–557. doi: 10.1007/978-3-030-58568-6_32.
[21] Ultralytics, "Quickstart - YOLOv8 Docs." https://docs.ultralytics.com/quickstart/ (accessed Mar. 24, 2023).
[22] A. Kumar, "Model Complexity & Overfitting in Machine Learning," *Data Analytics*, May 29, 2022. https://vitalflux.com/model-complexity-overfitting-in-machine-learning/ (accessed Jun. 04, 2023).
[23] T. FastAI, "FastAI 05_pret_breeds." https://colab.research.google.com/github/fastai/fastbook/blob/master/05_pet_breeds.ipynb (accessed May 08, 2023).
[24] J. H. and S. Gugger, "fast.ai - fastai A Layered API for Deep Learning." https://www.fast.ai/posts/2020-02-13-fastai-A-Layered-API-for-Deep-Learning.html (accessed Feb. 23, 2023).
[25] T. FastAI, "FastAI 06_multicat." https://colab.research.google.com/github/fastai/fastbook/blob/master/06_multicat.ipynb#scrollTo=invs-Qyn8lSC (accessed Feb. 27, 2023).