Deep Learning Based Layout Recognition Approach for HMI Software Validation

Ashton Xue Qun Pang
Jiaxing Jansen Lin
Chin Hee Ong
Yongquan Chen
AI Singapore
Singapore
e0031568@u.nus.edu
e0269033@u.nus.edu
ongchinhee@u.nus.edu
s200043@e.ntu.edu.sg

Ga Xiang Chong
Continental Automotive Singapore Pte Ltd
Singapore
ga.xiang.chong@continentalcorporation.com

Siti Nuruljannah Baharudin
AI Singapore
Singapore
jannah@aisingapore.org

Yon Shin Teo
Continental Automotive Singapore Pte Ltd
Singapore
yon.shin.teo@continentalcorporation.com

Kevin Seng Loong Oh
AI Singapore
Singapore
kevinoh@aisingapore.org

***Abstract*—Human machine interface (HMI) software testing in the automotive industry is a laborious and time consuming process, as testers are required to compare every screen downloaded from the dashboard display cluster against the reference design specified in the requirements. The current industry standard allows for efficient validation of the content of display texts and graphics. However, validating layout information such as the relative positions, alignments, sizes and types of each visual asset on the screen remains a difficult task, as the differences between layouts are not homogeneous. At first glance, some user interfaces may appear to be rather similar but in fact fall under different layouts, due to slight variances in the region or activation conditions. In this work, we propose a two-stage deep learning based framework for efficient layout recognition, which can be applied in general HMI user interface validation tasks. First, we train an object detection model to produce bounding boxes around visual assets within the specified region of interest in the pre-processed input images, along with the object class label. Next, we match the extracted structural information of the test image against the ground truth reference layout designs via two different strategies: a self-supervised learning method (BYOL) and a custom IoU-based layout matching algorithm. While the IoU-based method slightly outperforms BYOL in terms of accuracy, BYOL boasts faster inference speed and can be generalized easily to unseen input images and layouts. Our approach is language agnostic and allows thorough validation of HMI screens in a holistic way.**

***Index Terms*—Automotive HMI Software, Software Testing, Layout Validation, Self-supervised Learning**

This research/project is supported by National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-100E-2021-085). *(Ashton Xue Qun Pang, Jiaxing Jansen Lin, Chin Hee Ong and Yongquan Chen contributed equally to this work.)*

I. Introduction

Human-machine interface (HMI) software, which offers a graphical user interface (GUI) for users to monitor and control underlying equipment and machinery, has seen widespread applications in different industrial domains. In addition to displaying the consolidated hardware information at the system level, users can also interact with the system via touch panels, switches, voice control and even hand gestures in a real time manner. As vehicles become increasingly more software driven than hardware driven, automotive HMI software plays a vital role in providing a seamless passenger experience. Drivers on the road rely on automotive HMI software for sophisticated driving applications including advanced driver assistance system (ADAS), smart navigation, connectivity and security. Moreover, the automotive HMI market is projected to grow globally from $10.71 billion in 2021 to $18.64 billion in 2028 at a CAGR of 8.2% in the forecast period of 2021 to 2028. In light of its growing impact on vehicle control and maintenance, revenue stream and most importantly passengers' safety and welfare, any defects in HMI software can spell disastrous consequences, for example vehicle call backs, drop in consumer confidence, and even the cost of lives.

Effective and efficient software testing is crucial to eliminate software defects and ensure that all designs and features were developed according to requirements defined by the original equipment manufacturer (OEM). Design defects can arise from systematic issues such as pixel defects, uneven screen uniformity, errors in signal conversion, etc. It can also originate from human errors by the designer or developers, as sometimes typographical and rendering errors can be found

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00085

in the texts, or incorrect icons might be displayed on the screen. Although there exist commercial solutions like optical character recognition (OCR) and pattern matching to validate the correctness of the content of individual texts and icons, there is a gap in identifying whether the overall design has the correct layout, which is determined by the types, sizes and relative positions of all graphical assets on the screen, instead of the content. Current industry standards rely on the naked eyes of test engineers to visually inspect the layout of the HMI software. This is challenging as a typical automotive HMI software can have hundreds of different layouts, and there are many layouts that appear to be largely similar to each other as they might correspond to either a same function group, or belong to product lines of different market or language variants. Coupled with higher volumes of vehicular state information and interactive components on the UI screen, this poses a higher strain for the test engineers to keep up with the increment in software complexity to guarantee test accuracy and coverage.

In this paper, we propose a novel, deep learning based two-stage framework for HMI software validation, consisting of object detection of UI assets for the first stage, followed by layout matching based on the detected objects in the second stage. For the second stage, we experiment with two different approaches - firstly a deep learning based approach using image embeddings generated from a convolutional neural network, and secondly a custom similarity matching algorithm where we propose several scoring and penalty schemes. Our novel two-stage approach for HMI software validation results in 500 to 600 work-hour savings per test project for typical automotive HMI software, with comparable accuracy as compared to current industry standards. In a typical test loop, around 20 to 30K test images may be downloaded from the clusters, but without the use of our solution, only a fraction can be covered due to time and human resource constraints. Instead of relying on testers to identify the correct layout visually, which limits the test coverage and introduces human errors, our approach can achieve 100% coverage and allows the layout recognition and HMI design validation workflow to be automated in an end-to-end manner.

II. Background

At the system test level, testers conduct black box testing on dashboard instrument clusters to ensure the developed software conforms to various automotive standards and displays normal behavior in various driving scenarios. As HMI software are graphical and interactive in nature, visual design validation is one of the most important test objective, which covers both structural and content validation. In the testers' workflow, these two types of validation are separated. Structural (layout) validation consists of checking whether the right types of UI elements are displayed at the right positions with the correct size, as well as being aligned properly for text elements. Content validation focuses on whether the text elements are typographically correct with no rendering errors, and no wrong choice of icons were displayed on the screen. Separating these two components allows for errors to be isolated and attributed to either structural errors or content errors.

As content validation can be readily addressed by OCR and pattern matching based commercial solutions, our framework in this paper focuses only on structural validation (specifically, layout recognition). Having said that, a well-developed structural validation framework also has the added benefit of greatly facilitating downstream testing tasks for content validation, because intermediate outputs from the layout recognition module, such as the objects' positional coordinates, will be passed to other processes like OCR to check for content and textual correctness.

III. Related Work

Previous work on GUI element detection in mobile applications compared the detection performance between traditional techniques, like edge detectors and template matching, with deep learning models like Faster R-CNN [1]. In the context of mobile applications where fonts and UI assets are plentiful and varied, the authors found that traditional techniques, generic object detectors (e.g. Faster R-CNN) and off-the-shelf OCR tools (e.g. Tesseract [2]) are not up to task for detecting GUI texts reliably. Based on their findings, they opted to detect text and non-text GUI elements separately in their GUI element detector [3], relying on the specialized EAST scene text detector [4] for text detection. For non-text GUI elements, they utilized traditional techniques for region detection, before classifying the regions into GUI elements using a fine-tuned ResNet-50 [5].

More recently, Khaliq et al. found more success in using deep learning object detectors EfficientDet [6] and DEtection-TRansformer [7] for detecting GUI elements to be used for functional user interface testing [8].

Meanwhile specific to HMI testing, Rosenbauer et al. proposed using a genetic algorithm for finetuning traditional models like template matching and feature detectors automatically, without which test engineers would have had to manually calibrate the parameters such as edge detection thresholds [9].

Object detection is only one component of our layout matching framework. The final step requires us to be able to match the arbitrary number of detections to a specific reference design layout. In their work for the RICO mobile application dataset [10], Deka et al. annotated text and images within UI screens as two separate classes. They also provide the annotations encoded as color coded images. An autoencoder is trained on the color coded images, which thus learns to recognize layouts of texts and images. They were then able to use the autoencoder's outputs to retrieve similar UI images from a library based on a given UI query.

For our deep learning based layout matching approach, we encode UI components into three main classes, namely "icon", "text" and "mixed", in order to train a layout embedding and matching model using self-supervised learning [11]. Self-supervised learning was used due to the scarcity of data inherent to our use case. Specifically, to train a convolutional neural network to recognize unique layouts, each layout has only one

Fig. 1. Annotated HMI Image

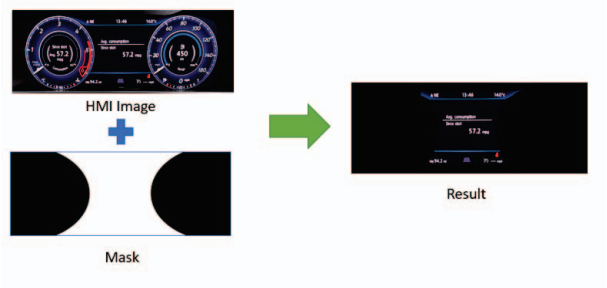

Fig. 2. Masking Process Visualisation

ground truth data sample (which is its reference specification). We found that using predictions from the upstream object detector as training data does not definitively improve the model, and may at times worsen it. In contrast with RICO, our layout embedding model will not be used to self-query the training UI dataset. Instead, they will be used to match frame-grabbed device screens, which are being tested, back to their corresponding reference layout design (or give a warning signal if there is no close match). For the matching process, we opted for Euclidean distance to compute the pairwise metric distance between the input screen and a particular reference layout, as its characteristics fit our use case. However, we note that there are also other popular distance metrics commonly used for image similarity matching and content retrieval [12].

IV. Dataset

The dataset used in this paper consists of annotated images from a commercial automotive HMI software. The dataset contains approximately 3,000 high-definition (1920 by 720) screen captures of modern vehicle digital dashboards, including settings, telematics, entertainment and navigation maps in various languages. An example HMI image can be found in Figure 1. The annotations are in the form of "HMI style guides", also known as the reference layouts. The style guides contain all the HMI design requirements from the OEM in the form of various metadata for each UI element, such as their exact bounding box location coordinates and object types. Object types are categorized into 3 classes:

- Text - rendered Unicode characters in multiple languages such as English, Chinese, and Japanese
- Icon - a graphical element that represents a specific action or feature
- Mixed - any combination of two or more text and icon objects

The final dataset of 1,619 annotated images across 328 unique layouts was curated after multiple iterations of exploratory data analysis (EDA) and data cleaning to eliminate duplicates, inconsistencies, and missing layout definitions.

As shown in Table I, EDA results revealed a highly skewed distribution across object types. Mixed annotations only constituted 3.2% of the total sample size. As a result, additional stratification steps were implemented during data splitting to ensure the distribution of object types was similar across the training, validation and test subsets.

TABLE I
Data split ratios and distributions

Dataset	**Train (70%)**	**Validation (15%)**	**Test (15%)**	**Total (100%)**
Images	1,133	243	243	1,619
Annotations	12,189	2,673	2,652	17,514
Text	6,568 (53.9%)	1,455 (54.4%)	1,461 (55.1%)	9,484 (54.2%)
Icon	5,217 (42.8%)	1,135 (42.5%)	1,109 (41.8%)	7,461 (42.6%)
Mixed	404 (3.3%)	83 (3.1%)	82 (3.1%)	569 (3.2%)

The following data transformation activities are performed on the data prior to model training:

- Masking was performed on the images to extract the region of interest (ROI). As seen in the visualisation in Figure 2, the speedometer and fuel guage are masked as these are not part of the layout.
- Conversion to COCO format [13] for training the object detection model in the first stage of the solution.

V. Methodology

In this section we describe in detail the architecture of our proposed layout recognition framework. To provide context, we start by first giving a brief overview of how it fits into the overall HMI testing workflow that includes both layout recognition (structural validation) and content validation, to achieve the test objectives in the automotive HMI test setting. As shown in Fig.3, bounding boxes of visual assets including textual and graphical objects are firstly detected from the input test image downloaded from the dashboard cluster. The salient features of the detected objects (in particular, their bounding box coordinates along with the object's class) are matched to the reference HMI style guides to recognize the correct reference layout. This completes the layout recognition component.

The next step in the tester's workflow is to apply OCR and pattern matching techniques to recognize the content of the texts and icons within each predicted bounding box, and validate them against the ground truth content described in

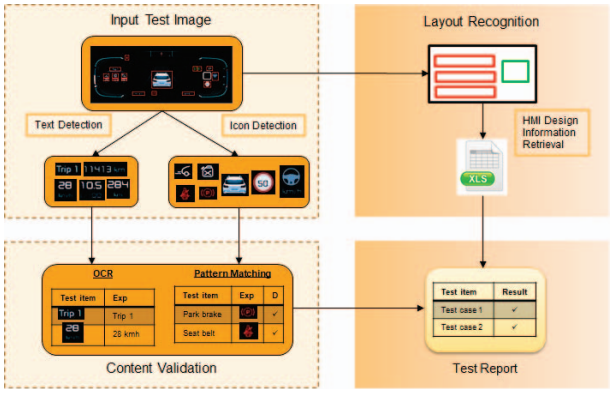

Fig. 3. Overall HMI Testing Workflow (including both Layout Recognition and Downstream Testing Tasks for Content Validation)

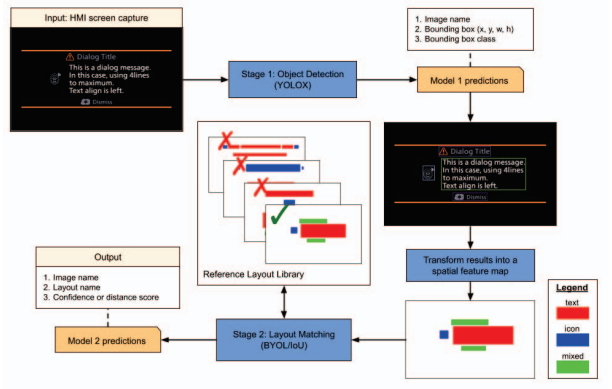

Fig. 4. Proposed Two-stage Framework for Layout Recognition

the style guide corresponding to the matched layout. This completes the content validation component. The results of the executed test scripts are consolidated inside the test reports, in which failed test cases are highlighted so that further actions can be taken by the test engineers to identify and resolve the issues.

A. Two-stage Framework for Layout Recognition

Our proposed framework for layout recognition is broken down into two stages: an object detection stage followed by a layout matching stage. In stage 1, a deep learning based object detection model is trained to detect and locate three classes of objects (UI elements): ”icon”, ”text” and ”mixed” [which is a group of icon(s) and text(s)]. The model outputs the X and Y coordinates of the top-left and bottom-right of each bounding box enclosing the detected object, as well as a predicted label for the class of the object. In stage 2, the boxes’ positions and the classes of the detected objects are used as features, to match the input image to each of the reference layouts and the UI elements within them. We tested two different methodology options within stage 2: a self-supervised deep learning model, and a custom similarity matching algorithm based on Intersection over Union (IoU). The overview of our framework is illustrated in Fig.4.

B. UI Object Detection

For stage 1, we chose the YOLOX [14] object detection model. This is an anchor-free version of the well-known YOLO series of object detectors. The primary consideration was its fast inference speed, followed by its object detection performance.

C. Layout Matching with Deep Learning

The first approach in stage 2 is based on a deep learning model. We generated a processed version of the input image by: i) converting the background to white, ii) replacing the UI objects by their bounding boxes, and iii) filling the boxes with one of three colors to encode the class of the object (red for ”text”, blue for ”icon”, green for ”mixed”). The resulting processed image has thus encoded the following information: positions of the objects and the type of content within. This creates the features for the deep learning model to generate embeddings to represent layouts.

The actual textual and graphical characteristics of the content itself are not utilized in the creation of the embeddings in stage 2. This is because the objective is to perform matching based on layout only, and it is important not to allow the matching to be biased by differences in the specific content within each of the objects. This is also the reason why the framework is split into the above two stages, instead of a single unified stage.

As a baseline, we adopted an unsupervised approach with no model training. At inference time, the processed version of the input image is generated based on the detected objects from the object detection model. The processed images are passed into the pre-trained ImageNet convolutional neural network to be embedded [11]. The generated embeddings are then compared against a dictionary of stored embeddings for the reference layout images, to find the nearest matching reference layout based on the Euclidean distance between the embedding vectors. This approach achieved above 70% accuracy for layout matching, but it was not sufficient for our use case.

To achieve better matching performance, our goal was to train a model to learn to discriminate between layouts. However, we were not able to adopt a full supervised learning approach, as we had a very limited number of training samples - on average, there were only a few ground truth sample images per layout. This practical limitation warranted a self-supervised learning approach, as such methods are designed to work with as little as one to two samples per layout. We chose the BYOL (Bootstrap Your Own Latent) [15] algorithm as the framework, due to its state of the art performance and high inference speed.

BYOL uses two neural networks, referred to as online and target networks (Fig. 5). The online network is defined by a set of weights θ and consists of three stages: an encoder $f\theta$, a projector $g\theta$ and a predictor $q\theta$. The target network has the

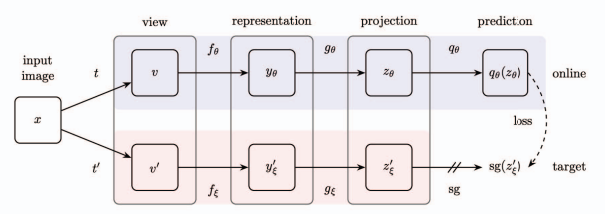

Fig. 5. Architecture of BYOL model

same architecture as the online network, but uses a different set of weights ξ, which are an exponential moving average of θ. The model aims to minimize similarity loss between the prediction $q\theta(z\theta)$ and the target network's projection $sg(z'_\xi)$. Like other Siamese architectures, BYOL uses two augmented views of the input image to train each network; however, it does not require the use of negative pairs of images.

D. Layout Matching with IoU-based Algorithm

The second approach in stage 2 is a similarity matching algorithm we designed from scratch. It is based on Intersection Over Union (IoU), which is a fundamental metric in object detection that quantifies the amount of overlap between two bounding boxes. The algorithm has the following defining elements:

- IoU score calculation for two single boxes. (This is the foundation of this algorithm.)
- Algorithm to search for the analytical best confidence score threshold for stage 1 predictions.
- Penalty for object class mismatch between a single ground truth bounding box and a single predicted bounding box.
- Penalty for different number of bounding boxes between ground truth layout and predicted layout.
- Similarity scoring schemes that incorporates the above mentioned elements to generate layout similarity scores from the single box-to-single box IoU score/s for every inference image.

1) Key hyperparameters:

Stage 1 confidence score threshold, $C.S.T$. This is the confidence score threshold for the stage 1 object detection model's predictions. It is a float value in the range of [0, 1). We also provide a custom algorithm to search for the analytical best value without the need to run the entire pipeline end to end.

Penalty for object class mismatch, P_1. This is the penalty for object class mismatch between the ground truth bounding box and IoU-matched predicted bounding box. It is a configurable arbitrary float value in the range of [0, 1]. Specifically, the IoU score of the IoU-based best match would be multiplied by this value if a class mismatch is found. A maximum value of 1 represents no penalty. A minimum value of 0 represents a full penalty.

Penalty for different number of bounding boxes, P_2. This is the penalty for the absolute difference in the total number of bounding boxes between ground truth and predicted layouts. It is a float value in the range of [0, 1]. It is used to adjust the layout similarity score. We provide four different formulae for calculating this penalty, outlined in supplementary material, section VII-A).

Scheme for layout similarity scoring. This is the scheme used for similarity score calculation between two layouts. We provide the following two schemes:

- Equal-weighted IoU
- Area-weighted IoU

2) Search algorithm for stage 1 object detection model's confidence score threshold:

This is a custom algorithm we created to search for the analytical best value for the stage 1 object detection model's confidence score threshold. This method makes use of all masked images that have matching reference layout coordinates (called "matchable images"). For each of these matchable images, we will perform a stage 1 object detection inference to get a predicted layout. The absolute difference in the total number of bounding boxes in the predicted layout versus the ground truth layout is then calculated. This difference is summed across all matchable images, and termed as the "sum of difference". We define the analytical best value as one that would minimise this sum of difference.

Based on this definition of the analytical best stage 1 confidence score threshold, we proceed to a 2-stage search algorithm:

Brute force search. For C.S.T, we try all values from 0.1 (inclusive) to 0.9 (inclusive) with a step size of 0.1 and calculate the sum of difference for each of them. The stage 1 confidence score threshold value that gives the lowest sum of difference shall be the best value.

Binary search. The best stage 1 confidence score threshold value determined from the brute force search is used as the pivot value for binary search.

3) Generate layout similarity scores:

The end goal of the IoU-based layout matching algorithm is to generate a matrix of similarity scores between all unique ground truth layouts and predicted layouts, to facilitate the comparison of each predicted layout to all unique ground truth layouts. Within a single layout-to-single layout comparison, each ground truth bounding box will have a single closest matching predicted bounding box, based on the highest IoU score found. The number of IoU scores to be expected here equates to the number of ground truth bounding boxes. If there is a mismatch in the object class between the ground truth box and the predicted box with the highest IoU score, the score would be adjusted by multiplying with the penalty ratio, P_1. An initial similarity score for each ground truth layout and predicted layout combination is calculated by aggregating the IoU scores of the matched bounding boxes in a way defined by the chosen layout similarity scoring scheme (i.e. Equal-weighted IoU or Area-weighted IoU). The final similarity score matrix is then derived by multiplying each initial similarity score with its corresponding penalty ratio, P_2.

VI. Experiments

A. Evaluation Metrics

In the stage 1 object detection model, mean average precision (mAP) was used to assess model performance.

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N}\text{AP}_i$$

where:

- N is the total number of queries or instances.
- APi is the average precision for the *i*-th query or instance.

Average precision is calculated as:

$$\text{AP} = \sum_{n}(R_n - R_{n-1})P_n$$

where R_n and P_n are the precision and recall at the nth threshold.

In the stage 2 layout matching model, the following classification metrics were used.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (1)$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (2)$$

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Sample Size}} \quad (3)$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

B. Stage 1 Object Detection - YOLOX

For our experiments, we divided the full dataset of 1,619 HMI images into 70% training, 15% validation and 15% test sets using stratified splitting. The data split ratios and distributions are outlined in Section 4 of this paper. The training set of 1,133 HMI images consist of 12,189 object annotations in total, of which 6,568 belong to text, 5,217 to icon, and 404 to mixed object classes. The validation set of 243 HMI images consist of 2,673 object annotations in total, of which 1,455 belong to text, 1,135 to icon, and 83 to mixed object classes. The test set of 243 HMI images consist of 2,652 object annotations in total, of which 1,461 belong to text, 1,135 to icon and 82 to mixed object classes. All of the models used in the experiments were pre-trained on the COCO object detection dataset by the MMDetection community. The MMDetection repository [16] was used for training the YOLOX model. In particular, we chose the YOLOX-s backbone as it is lightweight and offers the best tradeoff between performance and resource usage.

We trained the YOLOX model using the training set and evaluated its performance on the validation set. For training, we used a batch size of 16 and learning rate of 0.01 with the default YOLOX scheduler. In addition, We also turned off RandomFlip augmentations, together with the YOLOX augmentations from a default of last 15 epochs to last 50 epochs. The model was trained for a total of 100 + 100 epochs. This translates to 100 training epochs, with a continuation of 100 more training epochs on the already trained model weights, equating to 200 epochs in total. We used the AP50 score for evaluating all of the models, and the results are shown in Table II.

TABLE II
YOLOX Experiments

Model	Batch Size	Learning Rate	Epochs	Validation mAP
YOLOX-s	16	0.01	200	0.984
YOLOX-s	16	0.01	100 + 100	0.986
YOLOX-s	8	0.001	200	0.963
YOLOX-s	8	0.001	100 + 100	0.983

C. Stage 2 Layout Matching - BYOL

For the training dataset, we generated processed versions of the original reference layout images as described in Section 5.3. These consisted of white backgrounds with the ground truth UI objects' bounding boxes color coded into red for "text", blue for "icon", and green for "mixed" classes. A total of 347 reference layout images were generated for training, consisting of 328 unique layout images and 19 duplicated images. The Lightly open-source repository [17] for self-supervised learning was used for training the BYOL model, with a PyTorch Lightning implementation. All of the models in the Lightly repository were pre-trained on ImageNet by the Lightly community.

The models were trained with varying number of epochs - 300, 500 and 800 with a set of consistent model hyperparameters. All models were trained with 4 workers, batch size of 8, learning rate of 0.06, input size of 128, and a default collate function. Once the model was trained, we performed an evaluation on the test set using the learned weights. The test images were passed through our inference and post-processing pipeline to generate the image embeddings. These embeddings were then compared with the set of reference layout image embeddings using a K-nearest neighbors algorithm to match the predicted layout with the references, and retrieve the top-1 most similar layout.

For computing evaluation metrics, we used FiftyOne's [18] classification evaluation to generate a classification table of the results, based on comparing the top-1 matched layout with the ground truth reference layout. Key results are shown in Table III.

TABLE III
BYOL Key Experiments

Model (Stage 2)	Epochs	Precision	Recall	F1 Score	Accuracy
BYOL	300	0.946	0.946	0.946	0.946
BYOL	500	0.945	0.945	0.945	0.945
BYOL	800	0.957	0.957	0.957	0.957

The metrics were calculated based on matching the predicted layouts to 347 reference layouts, which include 19 duplicates.

D. Stage 2 Layout Matching - IoU-based Algorithm

This algorithm is used as an alternative layout matching solution in stage 2. In contrast to BYOL, it does not require any model training. We experimented with different sets of hyperparameters, which gave rise to different accuracy scores as shown in IV. C.S.T of 0.6 is the optimal solution derived by our custom search algorithm described in section V-D2.

TABLE IV
IOU-BASED LAYOUT MATCHING ALGORITHM KEY EXPERIMENTS

No.	C.S.T	P_1	P_2	Accuracy
1	0.6	0.5	["simple ratio"]	0.9839
2	0.6	0.5	["sigmoid", 1, 3]	0.9839
3	0.6	0.5	["ellipse", 4, 0.1]	0.9839
4	0.6	0.5	["ellipse", 6, 0.1]	0.9753
5	0.6	0.5	["linear", 6, 0.1]	0.9790

The accuracy metric was calculated by matching the predicted layouts to 328 unique reference layouts. Note that as the 19 duplicates are excluded here, these results may not be fully comparable to those calculated in Table III. However, we expect that any discrepancy will be negligible, and the results may still be compared in a practical sense. Refer to supplementary material section VII-A to understand the notations used for P_2. All experiments here were done using the "Equal-weighted IoU" scheme for layout similarity scoring.

TABLE V
END-TO-END PERFORMANCE MEASUREMENTS

Measurements	YOLOX+BYOL	YOLOX+IoU
Accuracy of layout recognition	0.957	0.960
Frames per second	5.468	0.300

Inference speeds were benchmarked on a single Azure NC4as T4 v3 virtual machine. Note that the BYOL algorithm utilized GPU while the IoU-based algorithm utilized CPU for inference. All experiments were carried out on hardware with 4xCPU 32GB and 1xGPU V100 16GB.

E. Summary of Final Chosen Approach

Both the YOLOX+IoU-based and YOLOX+BYOL approaches achieved highly impressive matching accuracy. Table V summarizes the experiment with the best test results across the combined two-stage, end-to-end workflow. While the accuracy of the YOLOX+IoU-based approach is marginally higher, the YOLOX+BYOL approach had much faster inference speed that was within acceptable limits for production implementation. The methodology was also more generalizable to different test settings in future due to the ability to learn. For these reasons, the final solution chosen for productionization was YOLOX+BYOL.

VII. CONCLUSION

In this paper, we have presented a deep learning powered HMI software validation framework based on an innovative two-stage layout recognition process. Its feasibility and effectiveness in an industrial production environment was demonstrated using an actual automotive HMI software as an example. Our framework addresses the labor intensive nature of automotive software testing by achieving HMI design validation, from ingestion of raw HMI images to identifying the correct layout, and finally extracting design information on the style guide to achieve HMI content validation in an end to end manner. A business study by an industry partner shows this can result in thousands of work-hour savings per year, and eliminates human errors in the process, as there are hundreds of layouts in each HMI software, some with differences indiscernible to the naked eye.

It is also interesting to note that the graphic assets bounding box detection method in our framework has been validated to be language agnostic, as it is able to recognize text boxes regardless of language. This greatly enhances its usability and practicality as one HMI software can feature up to 26 different languages. The automation process enabled by our deep learning based framework is fast (with an average inference speed of 0.9 FPS), which reduces the time to delivery from several weeks to overnight, and allows 100% test coverage since all test images can be examined.

ACKNOWLEDGMENT

The authors would like to thank Mr. Phyo WaiAung from Continental Automotive Singapore Pte Ltd and Mr. Codrut Toader from Continental Automotive Romania for their domain knowledge support and critical contributions in the integration of the solution into Continental's own HMI software testing toolchain; Mr. Lin Yuxin and Ms. Teoh Soo Yee from Continental Automotive Singapore Pte Ltd for testing the deployment of the solution within Continental; Ms. Sudha Ravi from AI Singapore for providing project management support; and Mr. Syakyr Surani and Mr. Ryzal Kamis from AI Singapore for providing machine learning operations and IT support.

REFERENCES

[1] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu, and G. Li, "Object detection for graphical user interface: Old fashioned or deep learning or a combination?" *CoRR*, vol. abs/2008.05132, 2020. [Online]. Available: https://arxiv.org/abs/2008.05132

[2] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, 2007, pp. 629–633.

[3] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "Uied: A hybrid tool for gui element detection," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1655–1659. [Online]. Available: https://doi.org/10.1145/3368089.3417940

[4] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[6] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," 2020.

[7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.

[8] Z. Khaliq, S. U. Farooq, and D. A. Khan, "A deep learning-based automated framework for functional user interface testing," *Information and Software Technology*, vol. 150, p. 106969, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922001070

[9] L. Rosenbauer., A. Stein., and J. Hähner., "A genetic algorithm for hmi test infrastructure fine tuning," in *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics - ICINCO,*, INSTICC. SciTePress, 2021, pp. 367–374.

[10] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 845–854. [Online]. Available: https://doi.org/10.1145/3126594.3126651

[11] S. Moon, D. Buracas, S. Park, J. Kim, and J. Canny, "An embedding-dynamic approach to self-supervised learning," 2022.

[12] M. Tahoun, K. Nagaty, T. El-Arief, and M. A-Megeed, "A robust content-based image retrieval system using multiple features representations," in *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.*, 2005, pp. 116–122.

[13] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[14] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[15] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent: A new approach to self-supervised learning," 2020.

[16] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[17] I. Susmelj, M. Heller, P. Wirth, J. Prescott, and M. E. et al., "Lightly," *GitHub. Note: https://github.com/lightly-ai/lightly*, 2020.

[18] B. E. Moore and J. J. Corso, "Fiftyone," *GitHub. Note: https://github.com/voxel51/fiftyone*, 2020.

SUPPLEMENTARY MATERIALS

A. Penalty for different number of bounding boxes, P_2

The calculated penalty ratio, P_2, will be used to adjust the similarity score, S, in the following manner:

$$S' = S \times P_2$$

We present 4 configurations to calculate the penalty ratio.

1) Simple ratio:

Let a = number of bounding boxes in ground truth layout
Let b = number of bounding boxes in predicted layout

Configuration for P_2: ["simple ratio"]

$$P_2 = \min[\frac{a}{b}, \frac{b}{a}]$$

2) Linear-based:

Let x = absolute difference in the total number of bounding boxes (Ground Truth vs Predicted)

Configuration for P_2: ["linear", scalar, m]

$$P_2 = \max[1 - \frac{x}{scalar}, m]$$

Refer to Figure 6 for an illustration of the formula above.

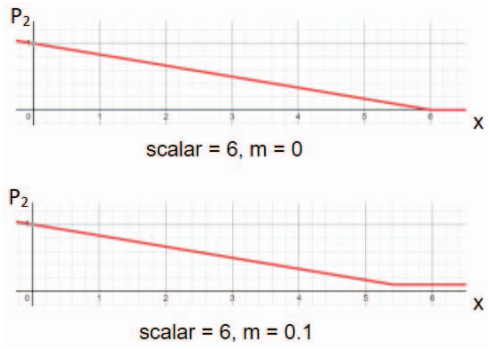

Fig. 6. Linear-based Penalty Ratio

3) Ellipse-based:

Let x = absolute difference in the total number of bounding boxes (Ground Truth vs Predicted)

Configuration for P_2: ["ellipse", scalar, m]

$$P_2 = \sqrt{\max[1 - (\frac{x}{scalar})^2, m^2]}$$

Refer to Figure 7 for an illustration of the formula above.

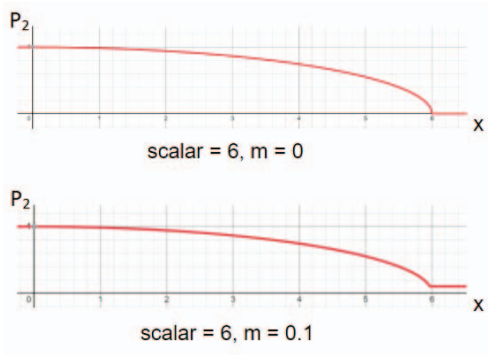

Fig. 7. Ellipse-based Penalty Ratio

4) Sigmoid-based:

Let x = absolute difference in the total number of bounding boxes (Ground Truth vs Predicted)

Configuration for P_2: ["sigmoid", scalar, shift]

$$P_2 = \frac{\sigma(\frac{-x}{scalar} + shift)}{\sigma(shift)}$$

Refer to Figure 8 for an illustration of the formula above.

B. Scheme for layout similarity scoring

Let the individual IoU score(s) between a ground truth layout and a predicted layout be $I_1 + I_n + ...$, where n is any

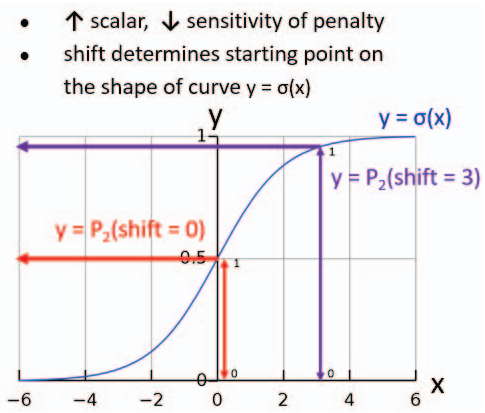

Fig. 8. Sigmoid-based Penalty Ratio. The red arrows and purple arrows represent the possible ranges of x and P_2 values when $shift = 0$ and $shift = 3$ respectively.

positive integer.

Let the corresponding area(s) of the ground truth bounding box(es) be $A_1 + A_n + ...$, where n is any positive integer.

If there is no matching bounding box between a ground truth layout and a predicted layout, the similarity score between the two layouts will be assigned 0.

As pointed out in subsubsection V-D3, n corresponds to the number of ground truth bounding boxes.

1) Equal-weighted IoU:

$$\text{Similarity score} = \frac{I_1 + ... + I_n}{n}$$

2) Area-weighted IoU:

$$\text{Similarity score} = \frac{I_1 \times A_1 + ... + I_n \times A_n}{A_1 + ... + A_n}$$