Real-Time Braille Letter Detection System Using YOLOv8

Reyshano Adhyarta Himawan [1], Eko Hari Rachmawanto [2*], Christy Atika Sari [3]
*Department Informatic Engineering, Dian Nuswantoro University, Semarang, Indonesia
111202213999@mhs.dinus.ac.id [1], eko.hari.rachmawanto@dsn.dinus.ac.id [2], atika.sari@dsn.dinus.ac.id [3]

Article Info

Article history:

Received 2025-07-03
Revised 2025-07-21
Accepted 2025-07-30

Keyword:

Braille,
Datasets,
Roboflow,
YOLOv8.

ABSTRACT

The purpose of this research is to create a system capable of detecting and recognizing Braille letters in real-time using the YOLOv8 algorithm for object detection, integrated with image processing technology and a user interface based on Tkinter. This system is developed to support visually impaired individuals in reading Braille text through the use of a webcam that captures and identifies Braille letters from images. The identification process is carried out by comparing the obtained images with a precompiled database of Braille letters. This research utilizes a dataset consisting of images of Braille code from letters A to Z, collected through public and private methods, with a total of 6013 images that comprehensively represent Braille letters. The model training is done using YOLOv8 to recognize Braille letter objects, with model performance evaluation using the Mean Average Precision (mAP) metric.The results of the tests show a very satisfactory model performance, with a mAP50 score of 0.98 and a mAP50-95 score of 0.789, as well as a high accuracy rate for almost all Braille letters tested. In addition, the system is equipped with a Tkinter-based graphical user interface (GUI) that allows users to operate the Braille letter detection process interactively and easily. This research proves that the YOLOv8-based object detection approach has significant potential for Braille letter recognition applications, which is expected to enhance accessibility and the independence of visually impaired individuals in reading text effectively.

I. Introduction

Braille is a tactile writing system that uses a combination of dots to represent letters, numbers, punctuation, and symbols [1]. The basic structure of Braille consists of six dots arranged in three rows and two columns, which are identified by visually impaired individuals using their fingers [2]. People with visual impairments rely on their auditory perception and somatosensory cues, primarily sound and Braille letters, to obtain information from their surroundings [3]. The struggles of individuals with visual disabilities to communicate with the world have inspired us to assist them in any possible way to make their lives easier [4]. The existence of Braille provides an essential solution for blind individuals; however, knowledge about it is still

limited to certain groups, which poses a barrier for the general public that does not understand the basics of Braille [5]. Created by Louis Braille in 1829, this system allows blind or visually impaired individuals to read and write through a combination of six raised dots arranged in a rectangular cell, consisting of two columns with three dots each. Approximately thirty-seven million out of six billion people worldwide suffer from blindness [6],[7].

Therefore, technology is needed that can detect and interpret Braille letters and translate them into text in an alphabet that is easy for the general public to understand [8],[9],[10]. Digital image processing technology has made rapid progress, providing great benefits for human life [11]. Image processing in this context involves processing images through computers to recognize objects in real-time using webcams. However, image-based Braille recognition remains vulnerable to external environmental factors, such as variations in light intensity, distance, and perspective, due to the small size of Braille characters [12].

In recent decades, artificial intelligence (AI) has shown extraordinary potential in addressing these challenges [13]. One prominent AI approach is deep learning, particularly for pattern recognition and object detection tasks [14].

Algorithms like YOLO (You Only Look Once) have proven to be effective in detecting objects in real-time with high accuracy, making it an ideal solution for practical applications [15],[16]. YOLOv8, as the latest version, offers significant improvements in efficiency, accuracy, and adaptability, making it very suitable for Braille character recognition applications [17],[18].

Previous research has attempted to address the challenge of recognizing and translating Braille letters through automated systems. One notable example is the study by I Made Agus Dwi Suarjaya, Bayu Adhya Wiratama, and Ayu Wirdiani [19], which developed a deep learning-based system to convert Braille to the Latin alphabet. This research employed deep learning architectures including Base Convolutional Neural Network (CNN), VGG-16, ResNet50, and Inception-v3, utilizing a dataset from the AEyeAlliance repository comprising images of Braille characters across 37 classes. The CNN-based model achieved a training accuracy of 98%, validation accuracy of 99%, and testing accuracy of 99.1%, demonstrating the significant potential of deep learning for Braille recognition and transliteration. The system was implemented as a website where users could upload Braille images for processing, indicating that detection and classification were performed offline rather than in real-time.

While these efforts have laid a strong foundation, they have not explicitly addressed the need for real-time Braille detection, a critical requirement for applications such as assistive devices for the visually impaired in dynamic environments. The offline processing approach in [19], while achieving high accuracy, introduces latency due to manual image uploads and batch processing, limiting its applicability for immediate translation during live interactions. Additionally, the reliance on a pre-defined dataset (AEyeAlliance) may not fully account for variations in real-world conditions such as lighting, noise, or diverse Braille patterns. Our research addresses these gaps by developing a real-time Braille detection system using YOLOv8s, achieving an mAP50 of 0.98, an average F1-score of 0.961, and a frame rate of 24.23 FPS, enabling seamless integration into real-time assistive technologies.

This research involves several stages, including the collection and processing of the Braille dataset, training the YOLOv8 model, and developing a real-time detection system with a graphical interface designed for the general user [20]. The resulting model is evaluated based on the detection accuracy, processing speed, and adaptability to variations in lighting conditions and Braille letter orientation. Thus, this research aims not only to advance accessibility technology but also to expand the application of YOLOv8 in various fields.

II. METHOD

A. *Previous Research*

Table 1 concluded that while other research has achieved extremely high classification accuracy for Braille characters, their primary focus has generally been on individual character recognition. In contrast, this research, employing the YOLOv8 model, successfully addresses a more complex challenge: Braille object detection, encompassing both the localization and classification of multiple characters within a single image. The exceptionally high mAP50 performance (0.98) and robust mAP50-95 (0.789) underscore the superiority of the YOLOv8 approach in real-time detection scenarios that demand not only accurate classification but also precise object localization, making it a significant and relevant contribution to this field.

TABLE I
PREVIOUS RESEARCH

Criteria	Our Proposed Method (YOLOv8)	AlSalman et al. [1]	Kausar et al. [2]	Florestiyanto & Prapcoyo [3]
Main Method	YOLOv8 (Deep Learning - Object Detection)	Deep Convolutional Neural Network (DCNN) (Deep Learning - Classification)	Lightweight Convolutional Neural Network (CNN) with Inverted Residual Block (Deep Learning - Classification)	Gabor Wavelet (Feature Extraction) + Support Vector Machine (SVM) (Traditional Machine Learning)
Main Goal	Real-time Braille character detection (localization & classification)	Conversion of Braille images to multilingual texts (Braille cell recognition)	Automatic Braille character recognition (character classification)	Identification of Braille letters from images (character classification)
Problem Type	Object Detection (Multi-object in one image)	Classification (Individual Braille cell recognition)	Classification (Individual Braille character recognition)	Classification (Individual Braille character recognition)
Number of Classes	26 (Letters A-Z)	Dataset 1: 27 symbols (alphabet); Dataset 2: 37 symbols (alphabet, numbers, & punctuation)	English Braille & Chinese double-sided Braille	Lowercase, uppercase, punctuation, & numbers (72-77)
Dataset Size	6,013 images	Dataset 1: 1,404 images; Dataset 2: 5,420 images	Two publicly available benchmark Braille datasets	758 Braille data points
Key Metrics	mAP50, mAP50-95, Average Precision, Average Recall	Classification Accuracy	Prediction Accuracy	Accuracy, Precision, Recall

Performance Result	mAP50: 0.98; Precision: 95.6%; Recall: 96.3%; mAP50-95: 0.789	Dataset 1: Test accuracy: 99.28%; Dataset 2: Test accuracy: 98.99%	English Braille: Accuracy: 95.2%; Chinese DSBI: Accuracy: 98.3%	Accuracy: 98.15%; Precision: 97.66%; Recall: 98.28%
Comments	High performance for a more complex object detection task (localization & classification). Shows robustness across various IoU thresholds.	Very high accuracy for multilingual Braille classification. Method focuses on cell recognition.	High accuracy with an efficient lightweight CNN model. Focus on character classification.	High accuracy using traditional feature extraction methods. Smallerdataset

B. *Research Stages*

The approach to identifying Braille letters in this research is conducted in real-time by utilizing image processing technology [21]. The methods applied include the use of Jupyter Notebook software integrated with Tkinter and the YOLOv8 object detection algorithm. The main goal of this method is to compare captured images with stored object data through an efficient and straightforward object tracking process. The Braille code identification process is carried out through several stages. First, images are captured using the laptop's webcam and stored in a database. The stored images are then compared with images identified in real-time through the webcam. Tkinter is utilized for preprocessing, such as color conversion or image segmentation. YOLOv8, as a deep learning-based object detection algorithm, is capable of detecting quickly and accurately [22].

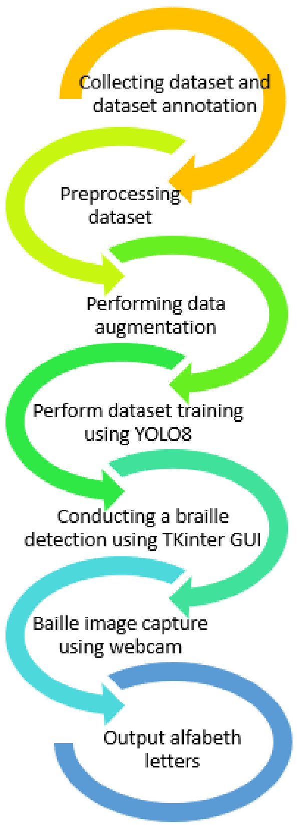

Collecting dataset and dataset annotation
Preprocessing dataset
Performing data augmentation
Perform dataset training using YOLO8
Conducting a braille detection using TKinter GUI
Baille image capture using webcam
Output alfabeth letters

Figure 1. Research stages

The application of YOLOv8 in this context is used to detect and recognize Braille letters in images [23]. This process involves training the model using a dataset of Braille images for the alphabet A-Z, which has been labeled and augmented to increase the quantity and variety of data. After training is complete, YOLOv8 is applied to images taken directly by a webcam to recognize Braille patterns and classify them as letters of the alphabet in real-time [24],[25]. If the detection results match the data in the database, the system will display the corresponding letter.

C. *Braille Datasets*

The data used in this research was obtained through two approaches, namely public and private. Public data collection was conducted by taking a Braille dataset from online sources such as Kaggle and Roboflow, which provides images of Braille codes for letters A-Z with a total of 6013 images. This dataset includes variations in lighting conditions, backgrounds, and positions, thus increasing the diversity of data for training. The private data search was collected by searching for existing Braille images on the internet such as Google and adding them to the dataset manually. The total dataset collected using this method reached about 213 images. After collection, the dataset was divided into three groups: 80% for training data (4810 images), 10% for test data (601 images), and 10% for validation data (601 images). This division aims to ensure that the model is trained and tested evenly to achieve optimal training results as visualized in Figure 2.

To expand and diversify the dataset, an augmentation process was carried out on the collected data. Augmentation includes rotation, brightness adjustment, and noise addition, to produce more varied data. In this study, the applied augmentation includes, rotating images by 6° and -6° as in Figure 3, adjusting brightness by 7% to make images brighter or darker as in Figure 4, and adjusting noise 1% pixels.

Figure 2. Braille Dataset

This augmentation helps the model to be more accurate in recognizing diverse data. The selection of rotation, brightness, and noise augmentation is based on the fact that during real-time detection, environmental factors such as varying lighting or unstable image positions due to hand movement can affect the results. With this augmentation, the potential for detection errors can be minimized.

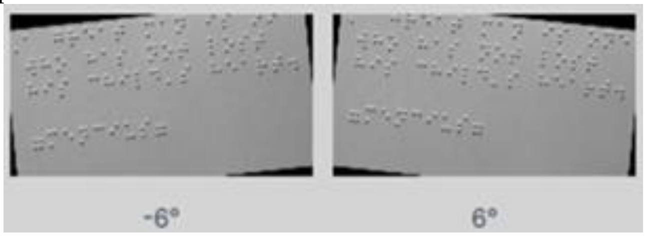


Figure 3. Braille Image that is Augmented with a rotation of 6°

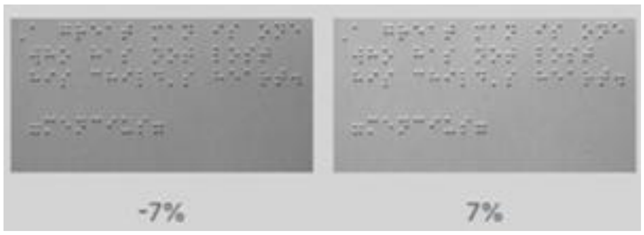


Figure 4. Braille Image that is Augmented with a brightness by 7%

D. Data Annotations and Label

The labeling process is a crucial step in the compilation of a dataset for training deep learning models, particularly in the development of a Braille character detection system. In this research, the researcher utilized Roboflow, an intuitive web-based platform designed to simplify image annotation, dataset management, and preprocessing stages [26]. With Roboflow, the researcher can effectively organize the dataset, ensure optimal annotation quality, and prepare the dataset to meet the needs of the deep learning model used. Here is a detailed explanation of the labeling process with Roboflow in this research.

The initial stage involves collecting datasets from two main sources. The first source comes from a public library dataset of Braille letters available on Roboflow, which provides images of Braille letters with annotations tailored for object detection purposes [27]. The second source is a self-created dataset by the researcher, collecting images of Braille codes from search results on Google, aimed at increasing data variation and improving the generalization ability of the Braille letter detection model.

The annotation process was carried out through the built-in annotation editor of Roboflow. Each image was opened manually, and areas containing Braille letters were marked with bounding boxes. The researchers utilized Roboflow's interactive feature, which involves dragging the cursor to create bounding boxes around each Braille letter. Once the bounding boxes were completed, the annotations were labeled according to the alphabet letters represented by the Braille codes in the image as in Figure 5. This research focuses only on alphabet letters, resulting in 26 labels used, namely letters A to Z.

After all images have been given bounding boxes and labels, the verification stage is carried out to ensure the accuracy of the annotations. This process involves a careful review of the dataset to check the positioning of the bounding boxes to ensure they align with the Braille characters, as well as to make sure there are no mistakes in the provided labels. This verification is important because annotation errors can decrease the model's performance during training and testing. Roboflow also offers flexibility with features such as keyboard shortcuts to speed up frequently used labeling, as well as tools to correct annotations if mistakes are found. This labeling process results in a well-organized and high-quality dataset, ready to be used for training the YOLOv8 model in a Braille letter detection system.

As a next step, after annotation and verification are complete, the dataset is further processed using Roboflow's preprocessing features, such as resizing images to 640x640 pixels and data augmentation to enrich variation and enhance the model's generalization capabilities. The dataset generated from this labeling process becomes a strong foundation for training the Braille character detection model with a high accuracy level.

The Braille letter detection dataset consists of 6,013 images, divided into 4,810 training images, 601 validation images, and 601 testing images, as defined in the data.yaml configuration file. The dataset was annotated using the YOLOv8s format, with each image paired with a corresponding .txt file in the labels subfolders (e.g., train/labels, val/labels, test/labels). Each .txt file contains one line per Braille character, formatted as <class_id> <x_center> <y_center> <width> <height>, where coordinates and dimensions are normalized to [0, 1] based on an image resolution of 640x640 pixels. For example, an annotation of 0 0.315625 0.14375 0.1015625 0.1625 for class 0 ('A') corresponds to a bounding box centered at approximately (202, 92) pixels.

The data.yaml file specifies 26 classes representing the Braille alphabet (A-Z), with nc: 26 and a names list mapping class IDs to characters. Annotations were performed manually using Roboflow, resulting in bounding box widths ranging from 0.089 to 0.120 (approximately 57-77 pixels) and heights from 0.147 to 0.167 (approximately 94-107 pixels), reflecting the 6x6 dot pattern of Braille characters. This variation in bounding box sizes is attributed to the manual annotation process, which may differ based on visual interpretation or image conditions.

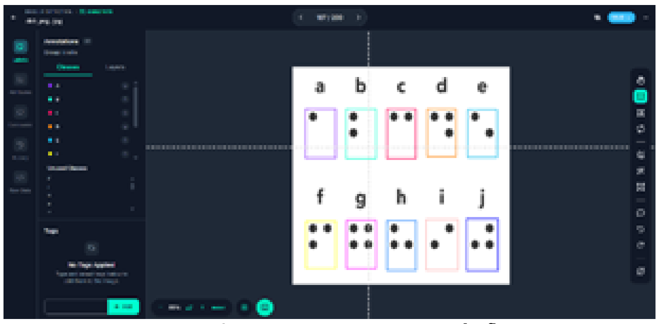


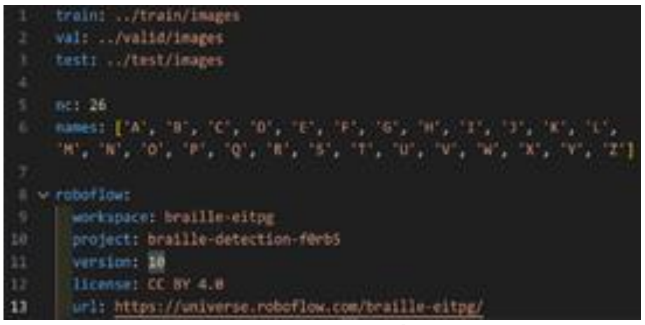
Figure 5. Annotation Process using Roboflow

E. *Training the Dataset with YOLO*

The dataset training process is the stage where the model is trained with the prepared dataset to produce a model capable of detecting objects with a high level of accuracy. At this stage, the pre-allocated test data is used to evaluate the performance of the trained model. The evaluation results are then compared with the validation data to ensure that the model can work well in real-world conditions. Here, the YOLO (You Only Look Once) algorithm is used to train the Braille code dataset. YOLO was first developed by Joseph Redmon in 2015. Unlike previous approaches such as R-CNN or Fast R-CNN that perform detection in multiple steps, YOLO combines all detection processes into a single integrated step.

The working method of YOLO involves dividing the image into a grid (for example, 13x13 or 19x19), where each grid cell is responsible for detecting objects in its area. Each cell predicts a bounding box, which includes the center coordinates (x, y), width (w), height (h), and a confidence score that indicates the probability of the object's presence and the type of object detected, such as animals, plants, cars, and others. The detection and classification process carried out in a single step using a convolutional neural network (CNN), making YOLO faster compared to other methods. YOLO also removes duplicate predictions for the same object and retains only the bounding box with the highest accuracy score.

Before the training begins, the folder structure and paths for the dataset must be well organized so that the 'data.yaml' file can be read. This file contains the paths to the training, testing, and validation dataset folders, as well as information about the number of classes ('nc'), which in this study totals 26 classes, and a list of class names in the 'names' section that includes letters A-Z. During training with YOLOv8 using the model yolov8s.pt weights, chosen for its stability and efficiency. The process was conducted for 25 epochs with an input image resolution of 640x640 pixels, as specified by the imgsz=640 parameter, the default hyperparameters of YOLOv8s were applied, including a batch size of 16 and an automatically adjusted learning rate. The choice of YOLOv8 was made because this model is considered more stable and efficient compared to newer versions. After the training is completed, the resulting model, 'best.pt', is used to predict Braille codes in real-time through a Tkinter interface with the laptop's webcam.



```
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 26
names: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

roboflow:
  workspace: braille-eitpg
  project: braille-detection-f0rb5
  version: 10
  license: CC BY 4.0
  url: https://universe.roboflow.com/braille-eitpg/
```

Figure 6. Data.yaml

F. *Braille Detection System GUI using Tkinter*

Figure 7 shows the Braille letter detection system development project, where the graphical user interface (GUI) was created using Tkinter, a built-in Python library designed for building desktop applications with graphical displays. Tkinter provides a variety of GUI elements, such as buttons, labels, and images, which enable the creation of an interactive interface to operate the application directly. Tkinter operates on an event-driven approach, where the application waits for user actions, such as pressing a button or moving the mouse. When an event occurs, Tkinter executes functions or commands set by the developer. In this project, the GUI developed with Tkinter is equipped with three main buttons: "Start Detection" and "Stop Detection" to control the Braille letter detection process through the camera, and a "Read Aloud" button to audibly announce the detected text, enhancing accessibility for visually impaired users. The start_detection() and stop_detection() functions, associated with their respective buttons, manage the workflow of the application by utilizing the trained Braille detection model. The "Read Aloud" functionality, implemented using the pyttsx3 library at 150 words per minute, converts the detected text into speech in real-time, providing immediate auditory feedback. This interface not only offers ease of control for users but also displays detection results visually and audibly, with the detection output shown immediately after the process begins, making it a comprehensive tool for real-time interaction.

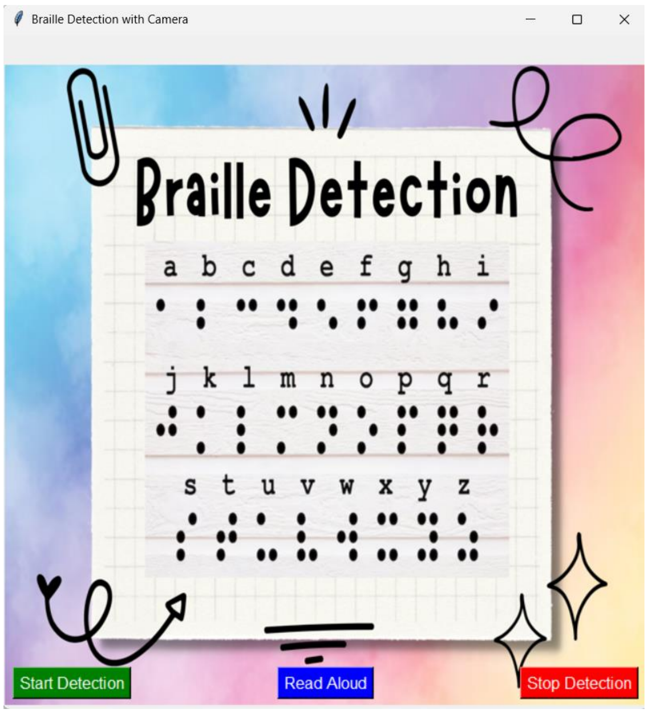


Figure 7. The User Interface using Tkinter

III. Results and Discussion

This research is aim to develop a Braille character detection system using the YOLOv8 model with input from cameras in real-time. This discussion focuses on evaluating the model's performance, the stability of detection under real-time conditions, and the effectiveness of the graphical interface designed using Tkinter. The results of this system are evaluated based on performance metrics such as accuracy, precision, recall, and usage in real-world situations.

A. Analysis of Model Evaluation Metrics

The Braille letter detection system was evaluated using a dataset of 6,013 Braille images, divided into 4,810 training images, 601 validation images, and 601 testing images. The YOLOv8s algorithm was employed to train the dataset for 25 epochs. The evaluation metrics used include mean Average Precision at IoU 50% (mAP50) to assess the accuracy of bounding box detection at an Intersection over Union (IoU) level of 50%, where a high value indicates precise object localization. Additionally, mAP50-95 calculates the average accuracy over an IoU range from 50% to 95%, providing a more comprehensive measure of the model's performance. Precision, recall, and F1-score were also computed to evaluate the model's ability to correctly identify and classify Braille letters, with F1-score balancing precision and recall to provide a single metric of detection effectiveness.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} APi \quad (1)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

Based on the evaluation results presented in Table II, the YOLOv8s model trained to recognize Braille letters demonstrates excellent performance across the 26 character classes (A-Z). Overall, the model achieved an mAP50 of 0.98 (98%) and an mAP50-95 of 0.789 (78.9%), indicating high accuracy in detecting Braille letters across various IoU levels. Letters such as M, Q, S, and T performed exceptionally well, with mAP50 values ranging from 0.991 to 0.993 and mAP50-95 above 0.8, reflecting precise and consistent detection. In contrast, letters such as I, J, R, W, X, and Z exhibited slightly lower performance, particularly in mAP50-95, with values of 0.785, 0.747, and 0.797, respectively. This is likely due to the smaller number of dataset instances for these letters, such as J (75 instances) and Z (99 instances), compared to letters with more instances, such as E (1,357 instances) and A (809 instances), which showed more stable performance with mAP50 values above 0.97.

To further evaluate per-class performance, the F1-score was computed for each Braille character. The model achieved an average F1-score of 0.961 across all classes, reflecting a strong balance between precision and recall. For example, class 'Q' achieved a precision of 0.981, recall of 0.981, and F1-score of 0.981, while class 'O' achieved an F1-score of 0.979, demonstrating robust detection for these classes. Most classes exhibited F1-scores above 0.95, confirming the model's ability to accurately detect and classify Braille characters. However, classes 'Z' (F1-score: 0.925, recall: 0.899) and 'M' (F1-score: 0.946, precision: 0.919) showed slightly lower performance, likely due to visual similarities with other Braille patterns or minor inconsistencies in dataset annotations. These findings align with the mAP50-95 results, where classes with fewer instances tend to have lower performance, suggesting the need for greater data variation for these letters.

TABLE II
Result of Metrics Evaluations

Class	Images	Instances	Precision	Recall	F1-Score	mAP50	mAP50-95
All	610	11386	0.956	0.963	0.961	0.98	0.789
A	378	809	0.944	0.959	0.951	0.978	0.776
B	149	211	0.927	0.919	0.923	0.973	0.729
C	202	291	0.966	0.964	0.965	0.987	0.801
D	339	628	0.965	0.986	0.975	0.989	0.786

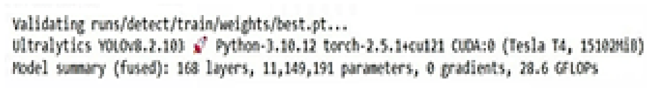
E	503	1357	0.971	0.971	0.971	0.989	0.807
F	167	218	0.949	0.946	0.948	0.962	0.765
G	177	206	0.974	0.961	0.967	0.983	0.807
H	392	555	0.951	0.971	0.961	0.985	0.815
I	299	551	0.936	0.96	0.948	0.985	0.778
J	67	75	0.96	0.962	0.961	0.985	0.726
K	158	201	0.96	0.948	0.954	0.97	0.782
L	301	680	0.962	0.971	0.967	0.983	0.804
M	199	279	0.919	0.974	0.946	0.97	0.756
N	337	558	0.969	0.973	0.968	0.983	0.797
O	434	798	0.979	0.98	0.979	0.991	0.802
P	155	236	0.951	0.962	0.957	0.99	0.804
Q	281	414	0.981	0.981	0.981	0.992	0.828
R	327	527	0.957	0.981	0.969	0.98	0.797
S	488	1038	0.963	0.983	0.973	0.987	0.801
T	270	577	0.962	0.978	0.970	0.987	0.794
U	181	228	0.937	0.982	0.959	0.99	0.81
V	268	330	0.96	0.976	0.968	0.989	0.792
W	111	161	0.98	0.925	0.952	0.976	0.785
X	111	121	0.942	0.967	0.954	0.954	0.789
Y	129	238	0.955	0.962	0.958	0.968	0.797
Z	97	99	0.952	0.899	0.925	0.943	0.747

TABLE III

BEST METRIC RESULT OF PROPOSED METHOD

Best Metric	Class	Value
Precision	O	0.979
Recall	D	0.986
F1-Score	Q	0.981
mAP50-95	Q	0.828

In terms of overall performance evaluation, as shown in Table III, the model achieved a precision of 0.979 (97.9%), a recall of 0.986 (98.6%), and an F1-score of 0.961 (96.1%) for the combined dataset. The high precision indicates that the model rarely produces incorrect detections (false positives), while the high recall demonstrates its capability to recognize nearly all Braille letters present in the images. The F1-score further confirms the model's robust performance by balancing these two metrics. These results, combined with the real-time performance (average FPS: 24.23, latency: 20.14 ms at 640x640 resolution), validate the model's suitability for practical Braille detection applications.



```
Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,149,191 parameters, 0 gradients, 28.6 GFLOPs
```

Figure 8. YOLO8 Architecture

Based on Figure 8, the YOLO model used in this research has an architecture with 168 layers and a total of 11,149,191 parameters, reflecting the complexity of the network in capturing features from the Braille letter dataset. In terms of computational efficiency, this model has 28.6 GFLOPs (Giga Floating Point Operations per Second), indicating that the number of operations performed to process a single image input is still quite efficient for the purpose of detecting Braille letters.

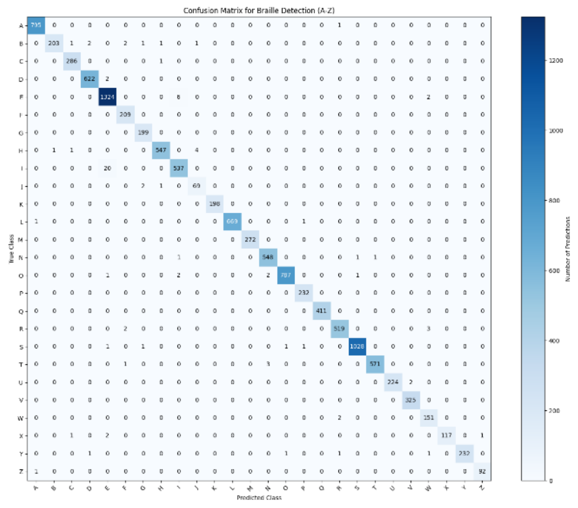


Figure 9. Confusion Matrix

This confusion matrix provides a comprehensive visualization of the Braille detection system's performance across the English alphabet (A-Z). The matrix is structured such that each row represents the actual, or true, class of a Braille character, while each column denotes the class predicted by the model. The numerical values within the matrix cells indicate the count of instances corresponding to the intersection of the true and predicted classes. A critical observation from the matrix is the high concentration of values along the main diagonal, stretching from the top-left to the bottom-right. These diagonal elements signify the number of correct classifications, or true positives, for each Braille character. For example, the system accurately identified 'A' 795 times, 'B' 203 times, and so forth, demonstrating a strong ability to correctly recognize a wide range of Braille characters. Conversely, the off-diagonal elements represent misclassifications. Numbers located in rows but not on the diagonal indicate false negatives, where a true character was incorrectly identified as something else. For instance, if there were a value in the 'A' row and 'B' column, it would mean that some 'A' characters were mistakenly predicted as 'B'. Similarly, values in columns but not on the diagonal represent false positives, where a character was incorrectly predicted as a specific class when it was actually a different one. The striking absence or very low counts in most off-diagonal cells throughout the matrix is a significant finding, indicating a remarkably low rate of misclassification. This suggests that the model is robust and highly effective in distinguishing between the various Braille characters, with minimal confusion between similar-looking patterns. Overall, the visual representation through the varying shades of blue, where darker shades on the diagonal denote higher correct prediction counts, reinforces the system's strong performance and its potential for reliable Braille detection in practical applications.

B. Analysis of Training Performance

Here, we had been yielded the performance and convergence of the YOLOv8 model during 25 epochs of training on a unique 26-class Braille character dataset. From the loss perspective, the train/box_loss, train/cls_loss, and train/dfl_loss curves show a rapid and consistent decline from high initial values towards stable and low values, specifically from approximately 1.8, 3.5, and 1.5 down to below 1.0, 0.5, and 1.0 respectively as in Figure 9.

This indicates that the model efficiently and stably learned the parameters to accurately predict bounding box locations, correctly classify 26 types of Braille characters, and precisely model spatial feature distributions on the training data. In parallel, the validation loss curves (val/box_loss, val/cls_loss, and val/dfl_loss) also exhibit similar and significant downward trends (from approximately 1.3, 2.0, and 1.3 down to below 0.9, 0.6, and 1.0, respectively), although with slight early fluctuations for val/cls_loss. This crucially confirms that the model did not overfit and was capable of generalizing the acquired knowledge to unseen Braille data.

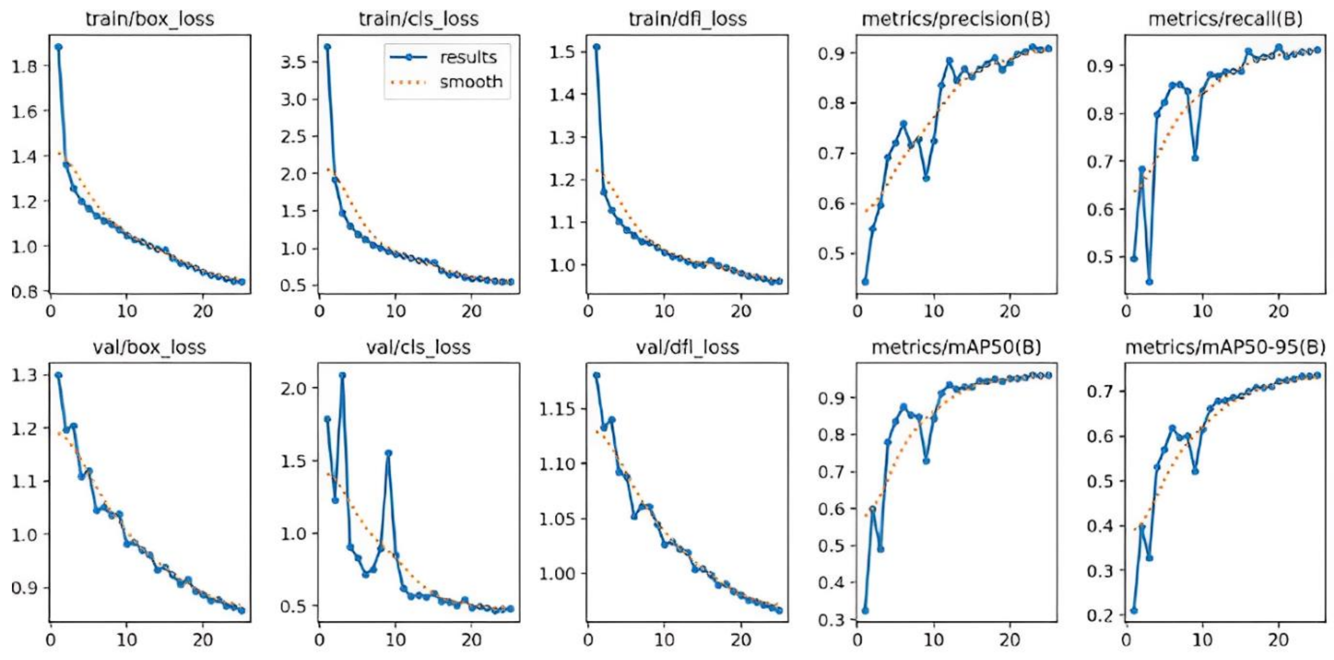


Figure 10. Training Graph Performance

Furthermore, the analysis of object detection performance metrics reveals substantial and impressive improvements; both metrics/precision(B) and metrics/recall(B) sharply increased from around 0.5 to above 0.9 after approximately 15 epochs, indicating the model's exceptional ability to minimize false detections while finding nearly all existing Braille characters.

This peak performance is reinforced by metrics/mAP50(B), which drastically improved from about 0.3 to above 0.9, affirming overall detection accuracy at a common IoU threshold.

Finally, metrics/mAP50-95(B), a stricter metric, also showed a solid increase from approximately 0.2 to above 0.7, proving the model's capability to generate highly precise bounding boxes and classify Braille characters with high accuracy even under more stringent evaluation criteria. Collectively, these results conclusively demonstrate that YOLOv8 is a highly effective and robust solution for complex multi-class Braille character object detection tasks.

C. Analysis of Bounding Box

Each Braille letter is marked by a rectangle that indicates the location of the letter. The colors, and label on the box reflect the identity of each class. In the image, all codes can be detected, although the appearance is somewhat unclear due to the proximity of the codes being too close together. Some errors were found, such as the sixth letter which should be 'F' but was recognized as 'D', and the twenty-third letter which should be 'W' but was detected as 'R'.

The numbers above the boxes indicate the model's confidence level in its predictions, for example, the character 'B' has a value of 0.8, which means the model is 80% confident that the character is 'B'. This indicates that the model is quite effective in detecting letters, although prediction accuracy can be influenced by factors such as distance and size of the image, noise in the image, and lighting conditions when the code image is captured as in Figure 10.

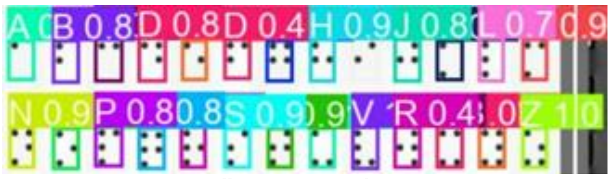


Figure 11. Braille Detection with Accuracy Score

Based on Table IV, there are 6 images of the Braille code prediction test. There are 4 images that show correct prediction results and 2 images that show examples of incorrect predictions. There several letters that are sometimes misread, such as the letter 'I' which is sometimes read as 'E', the letter 'R' which is read as 'W', and the letter 'H' which is read as 'J'. This is influenced by various factors such as distance, camera position, inadequate lighting, and hand instability while holding the image of the code.

D. Real-Time Performance Analysis

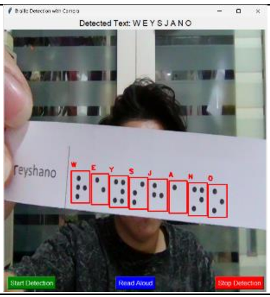
To evaluate the real-time performance of the Braille detection system, measured the Frame Rate per Second (FPS) and latency using a Tkinter-based graphical user interface (GUI) with a 720p webcam (30 FPS) on a system equipped with an Intel i7-12650H CPU, NVIDIA RTX 4060 8GB GPU, 16 GB RAM, and Windows 11. FPS was calculated as the total number of frames processed during a test divided by the duration of the test, while latency was measured as the average time from webcam frame acquisition to the display of detection results in the GUI, computed for each frame and averaged over the test duration.

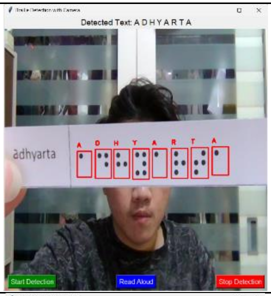
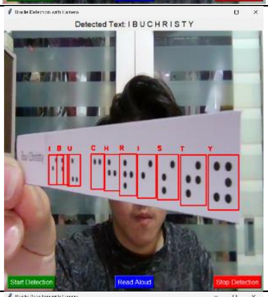
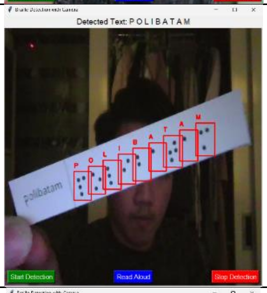
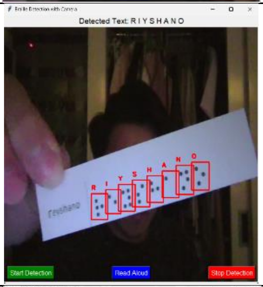
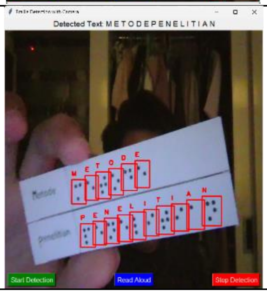
The system was tested with two input resolutions: 640x640 pixels (matching the training dataset) and 416x416 pixels (to reduce computational load). With an input resolution of 640x640, the system achieved an average FPS of 24.23 and an average latency of 20.14 milliseconds, indicating excellent responsiveness for real-time Braille detection. When tested with a reduced resolution of 416x416, the system achieved a slightly higher FPS of 25.16 but a marginally increased latency of 22.35 milliseconds, likely due to additional preprocessing overhead from resizing the 720p webcam input.

These results were obtained after implementing optimizations, including multithreading to separate frame acquisition from inference and rendering, and YOLOv8 parameters: a confidence threshold of 0.5 to filter low-probability detections, an IoU (Intersection over Union) threshold of 0.4 for Non-Maximum Suppression to reduce overlapping bounding boxes, and FP16 (half-precision) inference to leverage the GPU's computational efficiency. These optimizations significantly improved performance compared to initial tests (FPS: 20.60, latency: 33.91 ms), as the confidence threshold reduced unnecessary detections, the IoU threshold minimized redundant bounding boxes, and FP16 inference accelerated computations on the NVIDIA RTX 4060 GPU.

The performance is influenced by the YOLOv8s model complexity (168 layers, 11.1M parameters, 28.6 GFLOPs), input resolution, and GUI rendering overhead. The results confirm the system's suitability for real-time applications, such as reading Braille signage or educational materials for visually impaired users. Future improvements could include adopting the lighter YOLOv8n model, further reducing input resolution, or implementing advanced quantization techniques to achieve FPS above 30 and latency below 20 ms.

TABLE IV
VARIATIONAL TESTING BASED ON POSITION TO PROOF METHOD PERFORMANCE

Criteria	Visualization	Description
Testing with sufficient light situation and a bit tilted angle	 Braille Detection with Camera Detected Text: W E Y S J A N O	Wrong, because the letter 'R' is detected as unstable like the letter 'W', and the letter 'H' with the letter 'J'.

Testing with sufficient light situation and normal angle	 Detected Text: A D H Y A R T A adhyarta A D H Y A R T A Start Detection Read Aloud Stop Detection	Correct, because the output result matches the input words.
Testing with sufficient light situation and more tilted angle	 Detected Text: I B U C H R I S T Y I B U C H R I S T Y Start Detection Read Aloud Stop Detection	Correct, because the output result matches the input words.
Testing with low light situation and tilted angle	 Detected Text: P O L I B A T A M polibatam P O L I B A T A M Start Detection Read Aloud Stop Detection	Correct, because the output result matches the input words.
Testing with low light situation and tilted angle	 Detected Text: R I Y S H A N O reyshano R I Y S H A N O Start Detection Read Aloud Stop Detection	Wrong, because the letter 'E' is detected as unstable like the letter 'I'.
Testing low light situation, tilted angle, and 2 words input	 Detected Text: M E T O D E P E N E L I T I A N metode penelitian M E T O D E P E N E L I T I A N Start Detection Read Aloud Stop Detection	Correct, because the output result matches the input words.

This research focuses on the development of a real-time Braille letter detection system utilizing the YOLOv8 model. The dataset used consists of 6,013 images of Braille letters (A-Z), divided into training, validation, and testing groups. The model was trained for 25 epochs with an image resolution of 640x640, incorporating rotation augmentation to enhance data variation. Evaluation results indicate strong technical performance, with an average accuracy (mAP50) of 0.98, an average precision of 95.6%, and an average recall of 96.3%. The mAP50-95 value of 0.789 reflects robust detection across various IoU thresholds. Class-specific analysis revealed top performances: class 'O' with a precision of 0.979, class 'D' with a recall of 0.986, and class 'Q' with an mAP50-95 of 0.828, demonstrating the model's high accuracy and consistency.

Real-time testing identified challenges, with letters such as R, W, J, H, and I being detected inaccurately or inconsistently, likely due to visual similarities. The system is currently limited to recognizing alphabet letters A-Z, as constrained by the available dataset. The technical foundation supports practical implementations, such as portable Braille readers, and interactive educational tools. For future research, it is recommended to expand the dataset with Braille symbols and numbers, refine post-processing to address detection inconsistencies, and develop hardware prototypes to realize these real-world applications.

IV. CONCLUSION

This research focuses on the development of a real-time Braille letter detection system utilizing the YOLOv8 model. The dataset used consists of 6,013 images of Braille letters (A-Z), which are divided into training, validation, and testing groups. The model training was conducted for 25 epochs with an image resolution of 640x640, supplemented with rotation augmentation to enrich data variation. Model evaluation demonstrated strong overall performance, achieving an average accuracy (mAP50) of 0.98, with an average precision of 95.6% and an average recall of 96.3%. More specifically, an mAP50-95 value of 0.789 reflects robust detection performance across various Intersection over Union (IoU) thresholds. Further analysis of class-specific metrics revealed superior performance for certain Braille letters: class 'O' achieved the highest precision (0.979), indicating excellent accuracy in its positive detections. Class 'D' recorded the highest recall (0.986), demonstrating the model's strong ability to identify nearly all instances of this letter. Furthermore, class 'Q' exhibited the best overall accuracy with an mAP50-95 of 0.828, highlighting its robust and consistent detection.

Nevertheless, real-time testing using cameras revealed that some letters, such as R, W, J, H, and I are often detected inaccurately or inconsistently. This indicates challenges for the model in distinguishing these letters, possibly due to visual similarities between them. Additionally, the system can only recognize letters of the alphabet A-Z and is not yet able to detect symbols or other characters in the Braille system, due to the limitations of the available public dataset.

For future research, it is recommended to expand the dataset by including Braille symbols and numbers, as well as improving post-processing techniques to enhance detection stability. The addition of this dataset will enable the model to encompass all characters in the Braille system, improve accuracy, and ensure a more effective Braille detection system for visually impaired individuals.

DAFTAR PUSTAKA

[1] A. AlSalman, A. Gumaei, A. AlSalman, and S. Al-Hadhrami, "A Deep Learning-Based Recognition Approach for the Conversion of Multilingual Braille Images," Computers, Materials and Continua, vol. 67, no. 3, pp. 3847–3864, Mar. 2021, doi: 10.32604/cmc.2021.015614.

[2] T. Kausar, S. Manzoor, A. Kausar, Y. Lu, M. Wasif, and M. Adnan Ashraf, "Deep Learning Strategy for Braille Character Recognition," IEEE Access, vol. 9, pp. 169357–169371, 2021, doi: 10.1109/ACCESS.2021.3138240.

[3] D. Lee and J. Cho, "Automatic Object Detection Algorithm-Based Braille Image Generation System for the Recognition of Real-Life Obstacles for Visually Impaired People," Sensors, vol. 22, no. 4, Feb. 2022, doi: 10.3390/s22041601.

[4] R. M. Kumar G, S. Subhash, S. Guru, P. P. Jain, and R. M. Kumar, "Speech to Braille Convertor for Visually Impaired Using Python," International Journal of Advance Science and Technology, vol. 29, no. 10S, pp. 3916–3921, 2020, [Online]. Available: https://www.researchgate.net/publication/342169863

[5] H. Liu, R. Li, and X. Wang, "Optical Braille Recognition_Based on Semantic Segmentation Network With Auxiliary," 2020.

[6] S. Shokat, R. Riaz, S. S. Rizvi, K. Khan, F. Riaz, and S. J. Kwon, "Analysis and Evaluation of Braille to Text Conversion Methods," Mobile Information Systems, vol. 2020, 2020, doi: 10.1155/2020/3461651.

[7] M. A. Sayeedi, A. M. Muktadir Osmani, and D. M. Farid, "BrailleSense: Deep Learning for Braille Character Classification," in Proceedings - 6th International Conference on Electrical Engineering and Information and Communication Technology, ICEEICT 2024, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 681–686. doi: 10.1109/ICEEICT62016.2024.10534500.

[8] B. M. Hsu, "Braille recognition for reducing asymmetric communication between the blind and non-blind," Symmetry (Basel), vol. 12, no. 7, Jul. 2020, doi: 10.3390/SYM12071069.

[9] D. Gonçalves, G. G. Santos, M. B. Campos, A. M. Amory, and I. H. Manssour, "Braille character detection using deep neural networks for an educational robot for visually impaired people," 2021. [Online]. Available: https://github.com/lsa-pucrs/donnie-assistive-robot-sw

[10] I. G. Ovodov and R. Zelenograd, "Optical Braille Recognition Using Object Detection Neural Network," 2021.

[11] R. F. Rahmat et al., "Braille letter recognition in deep convolutional neural network with horizontal and vertical projection," Bulletin of Electrical Engineering and Informatics, vol. 13, no. 5, pp. 3380–3391, Oct. 2024, doi: 10.11591/eei.v13i5.7222.

[12] B. S. Park et al., "Visual and tactile perception techniques for braille recognition," Dec. 01, 2023, Society of Micro and Nano Systems. doi: 10.1186/s40486-023-00191-w.

[13] T. Mahin, N. Ava, and J. Zerin, "Real Time Braille & Sign Language Detection Using Artificial Neural Networks Declaration of uthorship," Jun. 2024.

[14] J. Lee and K. il Hwang, "YOLO with adaptive frame control for real-time object detection applications," Multimed Tools Appl, vol. 81, no. 25, pp. 36375–36396, Oct. 2022, doi: 10.1007/s11042-021-11480-0.

[15] A. Vijayakumar and S. Vairavasundaram, "YOLO-based Object Detection Models: A Review and its Applications," Multimed Tools Appl, Oct. 2024, doi: 10.1007/s11042-024-18872-y.

[16] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," Jul. 01, 2023, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/machines11070677.

[17] H. Lou et al., "DC-YOLOv8: Small-Size Object Detection Algorithm Based on Camera Sensor," Electronics (Switzerland), vol. 12, no. 10, May 2023, doi: 10.3390/electronics12102323.

[18] N. D. Hendrawan, R. Kolandaisamy, and A. History, "Jurnal Teknologi dan Manajemen Informatika A Comparative Study of YOLOv8 and YOLO-NAS Performance in Human Detection Image Article Info ABSTRACT," vol. 9, no. 2, pp. 191–201, 2023, [Online]. Available: http://http://jurnal.unmer.ac.id/index.php/jtmi

[19] I Made Agus Dwi Suarjaya, B. A. Wiratama, and Ayu Wirdiani, "Sistem Pengenalan Huruf Braille Menggunakan Metode Deep Learning Berbasis Website," JITSI : Jurnal Ilmiah Teknologi Sistem Informasi, vol. 5, no. 3, pp. 93–99, Jul. 2024, doi: 10.62527/jitsi.5.3.244.

[20] M. Sohan, T. Sai Ram, and Ch. V. Rami Reddy, "A Review on YOLOv8 and Its Advancements," 2024, pp. 529–545. doi: 10.1007/978-981-99-7962-2_39.

[21] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, “A Review of Yolo Algorithm Developments,” in Procedia Computer Science, Elsevier B.V., 2021, pp. 1066–1073. doi: 10.1016/j.procs.2022.01.135.

[22] J. R. Terven and D. M. Cordova-Esparaza, “A Comprehensive Review Of Yolo: From Yolov1 To Yolov8 And Beyond Under Review In Acm Computing Surveys,” 2023.

[23] W. Q. Yan and C. Li, “Braille Recognition Using Deep Learning,” 2021.

[24] M. Y. Florestiyanto and H. Prapcoyo, “Braille Detection Application Using Gabor Wavelet and Support Vector Machine,” RSF Conference Series: Engineering and Technology, vol. 1, no. 1, pp. 160–169, Dec. 2021, doi: 10.31098/cset.v1i1.390.

[25] S. Shokat, R. Riaz, S. S. Rizvi, I. Khan, and A. Paul, “Characterization of English Braille Patterns Using Automated Tools and RICA Based Feature Extraction Methods,” Sensors, vol. 22, no. 5, Mar. 2022, doi: 10.3390/s22051836.

[26] V. V. Kukartsev, R. A. Ageev, A. S. Borodulin, A. P. Gantimurov, and I. I. Kleshko, “Deep Learning for Object Detection in Images Development and Evaluation of the YOLOv8 Model Using Ultralytics and Roboflow Libraries,” in Lecture Notes in Networks and Systems, Springer Science and Business Media Deutschland GmbH, 2024, pp. 629–637. doi: 10.1007/978-3-031-70285-3_48.

[27] M. Kocaleva Vitanova Zoran Zlatev, B. Zlatanovska Lectoure Snezana Kirova, and B. Zlatanovska Mirjana Kocaleva Vitanova, “Deep Learning-Based Detection and Classification of Document Elements Using Roboflow.