Engineered Science
DOI: https://dx.doi.org/10.30919/es1192

Automatic Traffic Sign, Animal Detection, and Recognition Using You Only Look Once to Avoid Human-animal Road Conflicts

R. Rajesh* and P. V. Manivannan*

Abstract

This study presents a deep-learning-based solution to mitigate human-animal conflicts on the road, utilizing You Only Look Once (YOLOv8n) for automatic traffic signs, animal detection, and recognition (ATSADR). In addition, this study introduces a custom dataset that includes Indian traffic signs and animal images (*i.e.*, monkey and deer) collected from the Indian Institute of Technology Madras (IITM) Campus. Furthermore, the collected dataset was used to train the YOLOv8n model (with and without pretrained weights). A detailed comparative analysis of performance metrics, such as recall, mean Average Precision (mAP), accuracy, precision, F1-Score, and computational efficiency, was performed to select the most suitable model for the specified task. Real-time testing was conducted to demonstrate the practical performance of the proposed model. These findings signify the efficacy of the trained YOLOv8n model in real-time scenarios, showing promising solutions for addressing human-animal road conflicts. The trained model achieved the mAP of 73.7% and an average accuracy of 99.3%. The present work holds great potential to help prevent road accidents involving animals (deer and monkeys), and if trained with different animal datasets, the YOLOv8n model will offer a proactive solution to mitigate human-animal conflicts, ultimately enhancing the overall safety of both drivers and wildlife.

Keywords: YOLOv8n; Traffic sign detection, Animal detection, Deep learning, Data augmentation, Convolutional neural network.
Received: 13 June 2024; Revised: 22 July 2024; Accepted: 07 August 2024.
Article type: Research article.

1. Introduction

Automatic traffic sign detection and recognition (ATSDR) is a critical integral of modern intelligent transportation systems. This technology involves the use of computer vision and deep learning algorithms to detect and recognize traffic signs in real time from images or video streams captured by cameras installed on vehicles. The development of ATSDR systems has garnered notable interest in recent years due to their potential to enhance road safety, traffic management, and the overall efficiency of transportation systems. The work is discussed the comprehensive studies that have been carried out on ATSDR.[1] From this study, it can be observed that irrespective of the ATSDR algorithm, the current approaches exceed a detection accuracy of 95.6%. The evaluation of ATSDR algorithms has been predominantly conducted using datasets such as the German Traffic Sign Recognition Benchmark (GTSRB) and other self-collected datasets. However, these datasets may not fully represent the unique challenges posed by Indian road conditions, particularly in areas with specific traffic signs, such as those found in forested regions where human and animal habitats intersect (i.e., educational institutions such as IIT Madras). In these environmental conditions, Traffic Sign Recognition (TSR) and animal detection and recognition play pivotal roles in safeguarding drivers and wildlife. By accurately detecting and recognizing signs related to wildlife crossings, animal habitats, and cautionary alerts, TSR systems can significantly contribute to minimizing the risk of human-animal conflicts on roads.

Various approaches, including deep learning models, convolutional neural networks (CNN), and support vector machines (SVM), have been employed for precise and

Department of Mechanical Engineering. Indian Institute of Technology Madras, Chennai, Tamil Nadu, 600036, India.
**Email:* me19d755@smail.iitm.ac.in (R. Rajesh), pvm@iitm.ac.in (P. V. Manivannan)

effective traffic sign detection and recognition. These advancements have paved the way for the improvement of autonomous vehicles and advanced driver assistance systems, significantly contributing to the future of smart and safe transportation. Early work began with a technique for identifying traffic signs using self-organizing maps (SOM). This approach entails the extraction of a feature vector that characterizes the Region of Interest (ROI) using a preprocessor, subsequently passing it to the SOM for recognition. The accuracy achieved with this method is reported to be very high, which indicates the effectiveness of using self-organizing maps to detect and recognize road signs.[2] The authors presented an effective TSR using the Local Energy based Shape Histogram (LESH).[3] Initially, color-based segmentation was employed to segment the traffic signs, and contourlet transform was used to identify the shape of the segmented signs. Subsequently, the features from the segmented traffic sign were extracted using the local shape information of the sign within an image by considering the distribution of the local energy responses in different directions. Essentially, it characterizes the shape and structure of signs within an image by quantifying the distribution of edges and contours. Similar to LESH, the authors proposed using a normalized signature of traffic signs to detect the shape of traffic signs.[4] Using the cross-correlation matrix, traffic signs were recognized with an average accuracy of 91.07%. The major drawback of this method is that the computational cost of the correlation process and normalized signature of the traffic sign are sensitive to illumination variations, scaling, and orientation. To address this problem, an improved method was developed,[5] where the VIAPIX acquisition module (which contains three cameras and can produce 6-megapixel panoramic images) was used for ATSDR. The authors used geometrical and color-based segmentation to detect traffic signs. Furthermore, Inverse Fast Fourier Transform (IFFT)-based correlation matching was performed to recognize traffic signs. In addition, this correlation estimation process was improved by estimating the correlation between the filtered reference image and detected traffic signs. The reference images were filtered based on the peak-to-correlation energy (PCE). The authors developed a shape-symmetry estimation method to detect traffic signs in segmented input images.[6] Since traffic signs have distinguishable color features (BLUE and RED), red-blue normalization was initially performed, and adaptive thresholding was applied to segment the images. Then, using the histogram of the different traffic sign shapes, the shape symmetry that detects the traffic sign is estimated. The validation results (using the GTSRB dataset) revealed that the developed method works well under low-contrast and low-brightness conditions; however, the false detection rate was high.

To avoid complexities in the thresholding process, a Lookup Table (LUT) based traffic sign segmentation method was proposed.[7] In this, the authors have created two LUTs (Hue and Saturation) for segmenting each color of the traffic sign (BLUE, RED, and YELLOW). Once the LUT was applied to the images, an output-segmented image was obtained. Subsequently, the normalized signature of the traffic sign is used to train the SVM, which classifies traffic signs. From the results, it is evident that the SVM-based classification achieved an 81.34% true-positive rate for the tested dataset. CNN-based traffic sign detection was presented.[8] In this study, the authors initially transformed the input RGB image into a grayscale image using SVM and then used CNN to detect and recognize traffic signs on the GTSRB dataset. Furthermore, from this study, it can be noted that increasing the number of learnable layers significantly increased the true-positive rate; however, the training time increased. An improved detection network using the Faster R-CNN was developed [9], and this network achieved an mAP of 0.3449. To enhance the mAP and detect smaller traffic signs in a larger image and under complex conditions using Faster R-CNN, the authors proposed an attention mechanism in their work.[10] In addition, a Fine Region Proposal Network (FRPN) was implemented to yield the final region proposals for smaller signs. The presented work shows two frames per second (FPS) improvement compared to the usual Faster R-CNN, and it also shows 0.4981 mAP for the smaller signs (i.e., 0.15 mAP improvement) on the GTSRB dataset. A similar Faster RCNN-based ATSDR was presented.[11] In this study, the authors tested the network using the GTSRB dataset and attained an mAP of 91.75%. A deep neural network framework was proposed to perform roadside analysis, including TSR and lane analysis (vehicle and lane detection).[12] In this, the authors have used the CNN to perform the defined tasks. The TSR achieved an average precision of 93.94% with training for 500 epochs, and the lane analysis module performed the detection task at 64 FPS. Traffic sign segmentation has been improved by utilizing Hue, Saturation, and Value color space conversion, and recognition using the LeNet-5 CNN model shows improved overall performance.[13] In general, processing time can be reduced by decreasing the input image resolution. However, this affected the accuracy of the detection process. A region proposal for traffic signs was implemented with parallelized preprocessing of the input image and CNN on a Graphical Processing Unit (GPU).[14] Subsequently, the YOLO model was implemented for ATSDR owing to its processing speed, accuracy, and unequivocal

architecture. The YOLO model achieved the mAP of 93.44% with a processing speed of 29 FPS. Moreover, it can be noted that YOLO takes 45 ms to process 1920 × 1054 resolution images.

Since extracting image features from the frequency domain is robust, a Neural Network that operates in the frequency domain was presented.[15] The test results on the GTSRB dataset show that this network achieves higher accuracy with a smaller number of neurons in the frequency domain than in the spatial domain. A YOLO network-based TSR for the GTSRB dataset was presented for extreme imaging conditions, such as cloudy, snowy, night, and shadow occlusion. The results show that the TS-YOLO model achieves an mAP of 83.73% at 83 FPS when trained for 600 epochs.[16] The YOLOv3 model was used for the TSR in a previous study.[17] A TSR using YOLOv5 was presented a custom-collected dataset comprising eight classes.[18] YOLOv5 achieved an mAP of 97.7% for the presented custom dataset. A recent study presented Indian traffic sign detection using a modified Mask-RCNN.[19] In this study, the authors used histogram images to train the Mask-RCNN for 10 epochs and achieved an average precision of 96.7%, which shows an improvement of nearly 3% over the Fast RCNN.

From the above discussion, it can be seen that ATSDR is available for the GTSRB dataset, and environmental conditions such as human and animal habitat intersections (*i.e.*, educational institutions such as IIT Madras) are not available. Moreover, TSR accuracy can be improved using a recent network, such as YOLOv8, owing to its speed and accuracy for object detection tasks on the MS-COCO dataset. Therefore, this study presents a new approach to resolving human-animal conflicts on the road by employing the YOLOv8 model-based ATSADR, which is trained using a customized ATSADR dataset collected from the IITM campus. The key contributions of this study are as follows.

1) A custom dataset comprising 21 Indian traffic sign classes and animals, including monkeys and deer, is collected, annotated using the Computer Vision Annotation Tool (CVAT), and utilized for training our ATSADR deep-learning model.
2) Investigations are conducted to assess the effectiveness of the YOLOv8n model when trained with and without pretrained weights using the collected ATSADR dataset.
3) Real-time experiments show that the proposed ATSADR offers a proactive solution to mitigate Human-Animal Road conflicts and promote safer driving.

2. Materials and methods

This section discusses the materials and methods used in this study. The methodology of the developed IITM-ATSADR algorithm is depicted in Table 1, which comprises three stages: dataset generation, training pipeline, and test pipeline.

During the dataset generation stage, we collected 21

Table.1 Methodology of the proposed IITM-ATSADR algorithm.

Algorithm: IITM-ATSADR algorithm

Require: Train YOLOv8n model on custom ATSADR dataset

Input: 21 classes of traffic signs and 2 classes of animals, N = number of images, T = test images, E = number of epochs

Output: Trained YOLOv8n model (model), output

Dataset generation

capture images of various classes of traffic signs and animals like monkey and deer
train = 0.9*N, validation = N-train, & T
for i ← 1 to train **do**
    data augmentation ← scaling, rotation, shear, brightness change
end
CVAT data annotation
for i ← 1 to train **do**
    store bounding box, label
end

Train pipeline

load images and labels
for i ← 1 to N **do**
    mosaic augmentation
end
for i ← 1 to E **do**
    training on Nvidia 4GB 3050Ti GPU
    store weights
end
model ← Trained YOLOv8n model

Test pipeline

threshold ← 0.5
for i ← 1 to T **do**
    result ← X_1, X_2, Y_1, Y_2, score, label
    if score > threshold **do**
        draw bounding box and impose label
    end
    draw bounding box and impose label
end

classes of Indian traffic signs and two classes of animals, monkeys and deer (called the ATSADR dataset). Additionally, to expand our dataset, we utilized different data augmentation techniques and labeled our ATSADR dataset using the CVAT open-source annotation tool. During the training process, we employed the YOLOv8n model (both with and without pretrained weights) on a GPU. Once trained, we used the model to conduct the experiments during the testing stage.

2.1 Evolution of YOLO

The YOLO algorithm was first introduced in 2016, and it proposes a single neural network that forecasts bounding boxes and the probability of the object class directly from an entire image to achieve real-time object detection. Following this, YOLOv2 was introduced as an improvement to address some of the limitations of the original version, such as the localization accuracy. YOLOv3, released in 2018, further enhances the performance of the algorithm by incorporating features such as multiscale prediction and feature pyramid networks. In 2020, an updated version called YOLOv4 was introduced, which focused on improving speed and accuracy. YOLOv4 incorporates technical advancements such as the CSPDarknet53 backbone and PANet path-aggregation neck. Later, YOLOv5 adopted a different approach using the CSPNet network and introduced extensive hyperparameter optimization to achieve high performance. The improved YOLOv5 is called YOLOv6 and uses anchor-free detectors. In 2023, the YOLOv7 architecture was introduced with three key features: Extented-Efficient Layer Aggression Network (E-ELAN) encompasses strategies for efficient learning through the regulation of gradient paths, model scaling for the optimized use of hardware, and the integration of the "bag-of-freebies" framework, employing parameter change to improve both model accuracy and efficiency.

2.2 YOLOv8 architecture

Recently, YOLOv8 was introduced with improvements over other YOLO variants.[20] Analogous to YOLOv5, YOLOv8 utilizes an anchor-free detector that directly detects the center of an object, rather than offsets from predefined anchor boxes. This anchor-free approach minimizes the number of box predictions, consequently accelerating the crucial Non-Maximum Suppression process for final object detection. In addition to the data augmentation techniques discussed (Section 2.4), the YOLOv8 model leverages mosaic augmentation during the training phase. Mosaic augmentation involves combining four images to introduce variations in object location and occlusion, contributing to the enhanced accuracy achieved by YOLOv8.

The mosaic augmented image is represented by equation (1) as follows,

$$I_{mosaic}(x,y) = f(x) = \begin{cases} I_1'(x,y), & if\ x < W\ and\ y < H \\ I_2'(x-W,y), & if\ x \geq W\ and\ y < H \\ I_3'(x,y-H), & if\ x < W\ and\ y \geq H \\ I_4'(x-W,y-H), & if\ x \geq W\ and\ y \geq H \end{cases} \quad (1)$$

In the output mosaic image (I_{mosaic}) equation (1), the top-left, top-right, bottom-left, and bottom-right images are referred to as I_1', I_2', I_3', and I_4' respectively. Furthermore, the width and height of the original image are denoted by W and H respectively, while the pixel coordinate of the mosaic image is denoted by (x, y).

The YOLOv8 presents five scaled variants, namely YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large), each with varying computational requirements and mAP. The computational intensity and mAP of each YOLOv8 version depend on the floating-point operations per second (FLOPs) and the number of parameters utilized by the model. For instance, YOLOv8n features the lowest FLOPs and parameters (8.7 billion and 3.2 million, respectively), whereas YOLOv8x leverages 257.8 billion FLOPs and 68.2 million parameters, resulting in a notably enhanced mAP at a higher computational cost, which exponentially increases the training time. Considering the computational complexity, this study used the YOLOv8n model for the ATSADR. In general, the loss function plays a pivotal role in model training. It is typically composed of three primary components: Distribution Focal Loss (DFL), class loss, and bounding box (bbox) loss. Each of these components makes a unique contribution to the overall performance of the model.

Distribution Focal Loss (DFL) - The DFL is employed to refine the predicted bounding box coordinates, aiming to improve localization precision by focusing on the distribution of the predicted bounding boxes. The DFL can be expressed as equation (2) as shown below.

$$DFL = \sum_{i=1}^{N} -\log(p_i) \quad (2)$$

where, p_i is the probability of the true bounding box location for each of the N predicted boxes.

Class loss - Typically, YOLOv8 employs Binary Cross-Entropy (BCE) to address class imbalance by concentrating on difficult-to-classify instances. The $Class\ loss$ can be expressed as equation (3) as shown below.

$$Class\ loss = -\sum_{c=1}^{C}[y_c \log(p_c) + (1-y_c)\log(1-p_c)] \quad (3)$$

where C is the number of classes, y_c is the ground truth label for class c, and p_c is the predicted probability for class c.

Bounding Box ($BBox$) *Loss* - The bounding box loss calculates the discrepancy between the predicted bounding

box and the actual bounding box. YOLOv8 typically employs the Complete Intersection over Union ($CIoU$) loss for bounding box regression, taking into account the area of intersection, the distance between the centers of the boxes, and the ratio of the width to the height. The $CIoU$ loss can be expressed as equation (4) as shown below.

$$CIoU\ loss = 1 - IoU + \frac{\rho^2(b,b^g)}{l^2} + \gamma\upsilon \quad (4)$$

where IoU is the Intersection over Union between the predicted box and the ground truth box, $\rho(b, b^g)$ is the Euclidean distance between the centers of the predicted box b and the ground truth box b^g, l is the diagonal length of the smallest enclosing box covering the two boxes, υ measures the similarity of the aspect ratios, and γ is a positive trade-off parameter.

Total loss function - The YOLOv8 model's training process utilized a weighted combination of three distinct elements to determine its overall loss function which can be expressed as equation (5).

$$Total\ loss = \lambda_{DFL}.DFL + \lambda_{class}.Class\ loss + \lambda_{bbox}.CIoU\ loss \quad (5)$$

where, $\lambda_{DFL}, \lambda_{class}$, and λ_{bbox} are the weights that balance the contribution of each loss component.

2.3 ATSADR Dataset generation

This study introduces a custom dataset collected within the IITM campus to facilitate the training of the YOLOv8 model for ATSADR. The 21 classes of traffic signs collected and the two animal classes are shown in Fig. S1. Notably, the dataset encompasses unique traffic signs, including animal crossing, no smoking, walk and jog, and no urination, as depicted in Figs. S1(d), (n), (t), and (u). Unlike urban environments, the IITM campus hosts a diverse range of animals, including over 250 spotted deer, 20 blackbucks, and a substantial population of monkeys. Given the potential for human-animal conflicts on campus roads, mitigating such risks is paramount to ensuring the safety of both humans and animals. The presence of these animals necessitates the development of a vision-based animal detection and recognition network, in conjunction with traffic signs. Traffic signs, monkeys, and deer images were collected across campus to form the basis for training the YOLOv8 model. This effort aims to address the mischievous behavior of animals on roads, which requires vigilance from other road users. Since the ATSADR dataset was collected solely from the IITM campus, the number of occurrences was limited; hence, there may be a lack of diversity in terms of traffic signs and animal images. This limited diversity could potentially constrain the generalization capability of the model. Therefore, in the future, geofencing technology will be utilized to enhance the model's generalization capability, particularly concerning its application in diverse geographic locations where the dataset may not fully represent the varied traffic signs and animal species encountered on roads. Integrating geographically diverse datasets, potentially facilitated by geofencing, can mitigate this issue and significantly improve the robustness and real-world applicability of the model. Additionally, the animal dataset can include other classes such as dogs and cattle that roam on roads.

2.4 Data augmentation

In general, the accuracy of a deep-learning model relies heavily on the amount of data used for training. Since the number of traffic signs is low, it is imperative to augment the data to increase the size of the dataset by providing artificial constraints such as scaling, rotation, shear, horizontal translation, vertical translation, and brightness change.

Scaling: Scaling involves resizing images while maintaining the aspect ratio, helping the model learn to recognize objects of various sizes, and ultimately improving performance on unseen data. The images were randomly scaled between 0.8 and 1.2.

Rotation: Rotation involves rotating images by a certain angle, thus aiding in enhancing the model's capability to recognize objects from various perspectives and improving generalization and robustness. The images were randomly rotated between -45 °and +45 °.

Shear: Shear transforms images by shifting pixels along a defined axis horizontally and vertically. In both axes, the transformation is done between -0.2% and +0.2 % of the image width and height.

Brightness: Image brightness is typically reduced by subtracting a constant intensity value from all pixel values, thereby modifying overall luminance. A random constant value between 50 and 100 was subtracted from all the pixel values.

2.5 Data annotation

Data annotation involves labeling or tagging data to make them understandable to machines. This is a crucial process in deep learning and computer vision, as it helps deep learning models recognize and interpret patterns, objects, or features within the data. Accurate and comprehensive data annotation is vital for training effective deep-learning models. The data annotation process involved the utilization of the open-source Computer Vision Annotation Tool (CVAT) to annotate all 4800 images in the dataset, with particular emphasis on the precise delineation of bounding box coordinates around the signs of interest. Subsequently, these annotated bounding boxes are

used to crop the signs from the input images. In the present study, the quality of the annotation process was evaluated by template matching, a technique used to verify the correctness and precision of the annotations. The template-matching algorithm utilizes the normalized cross-correlation technique, which computes the correlation between the template and the actual image. The template-matching output is verified using a particular class of traffic signs/animals presented in the input image. This approach allowed for rigorous validation of the annotated data, ensuring the reliability and fidelity of the dataset for subsequent model training and evaluation in the context of research or application. The annotated label includes information such as the class name, x-coordinate, y-coordinate, width, and height. Once the dataset and its annotation were prepared, the dataset was split into training, validation, and testing datasets. In this study, testing was performed using test images captured separately, which were not included in the training dataset. The step-by-step procedure for data annotation using CVAT is illustrated in Fig. 1.

After downloading the annotation file from CVAT for the ATSADR dataset, it was necessary to create yet another markup language (YAML) file in Python. This YAML file contains the global path to the dataset folder, which includes the training, validation, training, and validation labels. Additionally, the names of the individual classes must be specified (this class name will appear in the output). Once the YAML file has been created, it can be used to train the YOLOv8n model.

3. Evaluation of developed ATSADR algorithm

3.1 Model training

The training setup utilized a PC featuring an AMD Ryzen 9 5900HS CPU operating at 3.3 GHz, 16 GB of installed RAM, and a dedicated 4 GB NVIDIA 3050Ti GPU. This configuration was employed to train the YOLOv8n model using a custom ATSADR dataset. The training environment was based on the PyCharm IDE with essential packages, such as PyTorch version 2.2.0 and CUDA 12.1. The learning rate (α) and batch size (b) play crucial roles in the training process and can significantly impact convergence, training time, and final performance of the model. The learning rate determines the number of steps required for optimization. A larger learning rate allows the model to learn faster, but it can also lead to instability and difficulty in converging to an optimal solution. The smallest learning rate leads to more stable learning but may require more time to converge (*i.e.*, $\alpha = 0.0001$). High learning rates (*i.e.*, $\alpha = 0.1$) can cause the model to overshoot the minimum loss, leading to divergence, while very low learning rates might result in slow convergence or getting stuck in local minima. In addition, a larger batch size ($b = 128$) enables the model to process more samples in parallel, leading to faster training but requiring more memory. A smaller batch size ($b = 1$) offers less accurate estimates of the gradient but concedes for more frequent updates on the model's weights. Larger batch sizes can result in faster convergence. However, they may also cause the model to become stuck in a global minimum. Smaller batch sizes can offer better generalization and may help the model escape shallow minima; however, they can also result in a slower convergence. Considering this, the training parameters, learning rate, and batch size were set to 0.01 and 16, respectively, and the training process was set to run for 600 epochs. Two versions of the YOLOv8n model were trained: one utilizing pretrained weights and the other without pretrained weights. The former model completed training after 149 epochs (6.875 h), reaching a point where no further performance improvements were observed. Conversely, the model trained without pretrained weights required 178 epochs (7.962 h) to achieve similar completion.

3.2 Metrics

The evaluation of the presented ATSADR depends on several metrics, including mAP, recall, precision, F1-Score, and accuracy. Mean Average Precision (mAP) is a widely used metric for appraising the effectiveness of object detection and

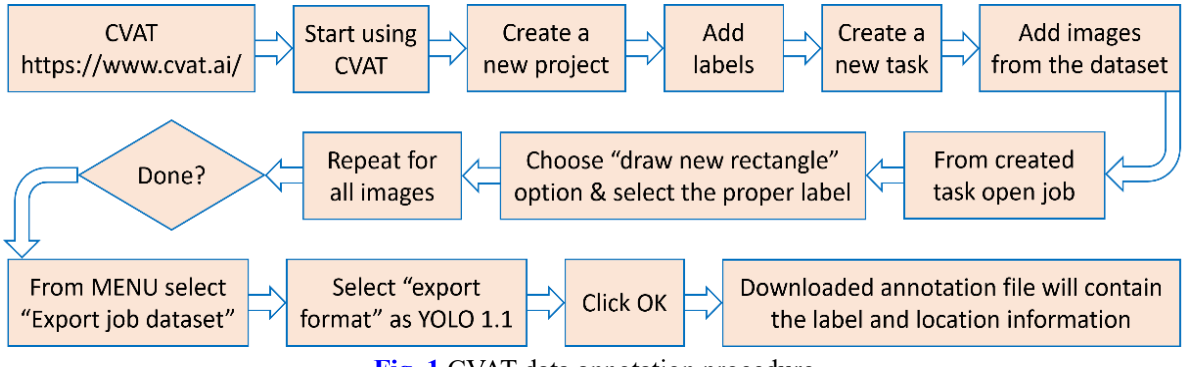

Fig. 1 CVAT data annotation procedure.

recognition models. It consolidates precision-recall values across numerous classes or categories into a singular scalar representation. Calculated as the mean of the average precision (AP) values, The mAP (expressed in equation (6)), calculated as the mean of the AP values, offers a comprehensive overview of the model's ability to accurately detect objects across diverse categories. Notably, the metric "recall" quantifies the model's ability to identify the actual positive classes correctly (expressed in equation (7)), while "precision" gauges the accuracy of the model's positive predictions (expressed in equation (8)). Furthermore, the "F1 score" serves as a gauge of the equilibrium between precision and recall within a model's predictions (expressed in equation (9)). Lastly, "accuracy" is calculated as the proportion of correct predictions to the overall number of predictions made. For a multiclass classification problem, the "accuracy" can be expressed using equation (10). The "miss rate" or "false negative rate" is typically calculated as the ratio of false negatives to actual positives, as shown in equation (11). The " false-positive rate" is a measure of the proportion of false positives to the total number of actual negatives. It can be expressed mathematically as equation (12).

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{6}$$

$$Recall = \frac{TP}{TP+FN} \tag{7}$$

$$Precision = \frac{TP}{TP+FP} \tag{8}$$

$$F1\ Score = 2 * \frac{Precision*Recall}{Precision+Recall} \tag{9}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{10}$$

$$Miss\ rate = \frac{FN}{FN+TP} \tag{11}$$

$$False\ positive\ rate = \frac{FP}{FP+TN} \tag{12}$$

In the above metrics (equations 6 to 12), a true positive (TP) represents the correct prediction of the positive class. Similarly, a true negative (TN) is for predicting the *negative* class correctly. Subsequently, a false positive (FP) *incorrectly* predicts the positive class, and a false negative (FN) represents *incorrectly* predicting the *negative* class. In general, the $TP, TN, FP,$ and FN are computed directly from the confusion matrix obtained after the training process. A confusion matrix is a $N + 1$x$N + 1$ (where N is the number of classes, and the additional class (+1) is the background) table used to evaluate the performance of a classification model by summarizing the counts of $TP, TN, FP,$ and FN. It provides a detailed breakdown of how the model's predictions compare with the actual labels, which is especially useful for understanding model performance on a class-by-class basis.

3.3 Testing

This study used 90% of the ATSADR dataset for training and the remaining 10% for validation. The confusion matrix obtained after training was utilized to perform statistical analysis by computing evaluation metrics such as mAP, Recall, Precision, F1-Score, and accuracy for both YOLOv8n models (with and without pretrained weights), as presented in Table 2. Notably, the training time for the model with pretrained weights was 6.875 h, whereas the model without pretrained weights required 7.962 h to complete the training. This discrepancy arises from the fact that pretrained weights equip the model with prior knowledge gained from a comprehensive dataset, facilitating faster convergence during the fine-tuning of the custom dataset. Conversely, training a model without pretrained weights requires starting from scratch, resulting in a longer training duration as the model learns features and patterns directly from the custom ATSADR dataset without existing knowledge. Table 2 highlights the performance of both YOLOv8n models on the custom ATSADR dataset, with a slightly higher mAP, as shown by the pretrained weight model on average. However, when examining specific classes, such as the left-turn traffic sign, it is observed that the recall, F1-score, precision, and accuracy remain consistent across both models (however, less recall and F1-score are observed). At the same time, the mAP is higher in the model trained with pretrained weights. It is essential to highlight that while the model with pretrained weights demonstrates a higher mAP, this might not directly translate to superior performance for individual classes. This disparity could be due to the aggregate nature of the mAP, which combines performance across all classes, potentially overshadowing the lower performance for specific classes. Therefore, it is vital to consider class-specific metrics alongside aggregate measures such as mAP, especially when evaluating the performance of individual classes.

Moreover, the YOLOv8n model without pretrained weights demonstrates superior performance in classes such as give way, pedestrian crossing, speed breaker, and stop sign, whereas the pretrained model shows a slight improvement for classes such as intersection, ten mph, 30 km/h, and deer. This highlights the exceptional performance of the YOLOv8n model without pretrained weights in the ATSADR. However, it is important to analyze the computational performance of both models to determine the best choice. Real-time experiments were conducted to determine the computational performance of both models. The results revealed that the YOLOv8n model without pretrained weights requires

Table 2. Evaluation metrics of the detected traffic signs.

Class	YOLOv8n (without pretrained weights)					YOLOv8n (with pretrained weights)				
	mAP 50-95	Recall %	Precision %	F1-Score %	Accuracy %	mAP 50-95	Recall %	Precision %	F1-Score %	Accuracy %
Animal crossing	87.8	100	100	100	100	88.1	100	100	100	100
Intersection	60.9	100	40	57.1	95.8	89.9	100	46.6	63.6	96.9
Cycle parking	91	100	100	100	100	89	100	100	100	100
Dangerous dip	81.5	100	100	100	100	83.8	100	100	100	100
Deer sign	90.9	100	100	100	100	86.9	100	100	100	100
Give way	83.8	100	100	100	100	87.5	100	96	97.9	99.8
Left turn	66.8	12.5	100	22.2	95.8	81.9	12.5	100	22.2	96
Narrow bridge	76.9	100	100	100	100	78.2	100	100	100	100
No overtaking	82.1	100	100	100	100	80.1	100	100	100	100
No parking	74.8	100	100	100	100	78.3	100	100	100	100
No smoking	59.2	100	100	100	100	59.8	100	100	100	100
No urinate	67.3	100	100	100	100	65.8	100	100	100	100
Pedestrian crossing	51.2	100	100	100	100	50.9	100	76.2	86.4	99
Right turn	74	100	100	100	100	80.7	100	100	100	100
Roundabout	73.8	100	100	100	100	73.4	100	100	100	100
Speed breaker	40.3	42.8	100	60	96	39.6	42.8	55.5	48.3	93.9
Stop sign	76.4	100	100	100	100	71.3	100	55.1	71.1	97.5
Ten kmph	86.9	97.8	95.7	96.7	99.4	87	100	97.8	98.9	99.8
Twenty kmph	82.1	100	100	100	100	81.5	100	100	100	100
Thirty kmph	86.5	100	92	93.8	99.4	86.6	95.8	95.8	95.8	99.6
Walk and jog	83.1	100	100	100	100	81.5	100	100	100	100
Deer	68.8	100	85.7	92.3	99	70.8	100	93.7	96.7	99.6
Monkey	48.8	100	90.9	95.2	99.6	47.5	100	95.2	97.5	99.8
Average	73.7	93.6	95.8	92	99.3	75.6	93.5	91.8	90.3	99.2

Table 3. Two-Tailed T-test values.

Test			mAP 50-95	Recall	Precision	F1-Score	Accuracy
two-tailed variance)	p-value	(non-equal	0.200117	0.682769	0.184571	0.278319	0.411789

approximately 28.2 ms to detect traffic signs with improved accuracy, whereas the pretrained model requires approximately 32 ms. Subsequently, to draw a robust conclusion about the selection of the best model, a statistical two-tailed t-test was performed considering there was no specific hypothesis about the direction of the variation between the means of the two sets being compared. The two-tailed t-test results are presented in Table 3. From Table 3 it can be seen that the two-tailed probability (p) is not less than the significance level (β) which is typically 0.05. Hence, there is no significant difference between training with and without pretrained weights on the presented ATSADR dataset.

However, based on the computational time (without the pretrained weight model is faster than that with the pretrained weight model) and the average of the individual evaluation metrics, the YOLOv8n model without pretrained weights is considered superior. Fig. S2 and Fig. S3 (collage 1 and collage 2, respectively) depict the detection results on the test images using the YOLOv8n model trained without pretrained weights. The GREEN letters in both figures correspond to the traffic sign classes shown in Fig. S1. The results showed the accurate detection of traffic signs ((a) to (u)) and animals, including monkeys (shown as (v)) and deer (shown as (w)), with high confidence. Notably, the model successfully identified small monkeys and deer (represented as 'baby' in Fig. S3 (v) and (w)), thereby serving as an effective tool to alert drivers and prevent road accidents.

It should be noted that traffic signs and animal detection are not two distinct tasks since the YOLOv8n model was trained with the entire dataset. Therefore, if the input image

contains both animals (deer or monkey) and traffic signs, it will detect both classes. However, the computation time for the multiclass object was slightly higher (0.2 times) than that for detecting a single class. Fig. 2 shows the miss and false positive rates for the custom ATSADR dataset. The miss rate (also called as false negative rate) can be computed using equation (11) or using the relation $(100 - Recall)$.

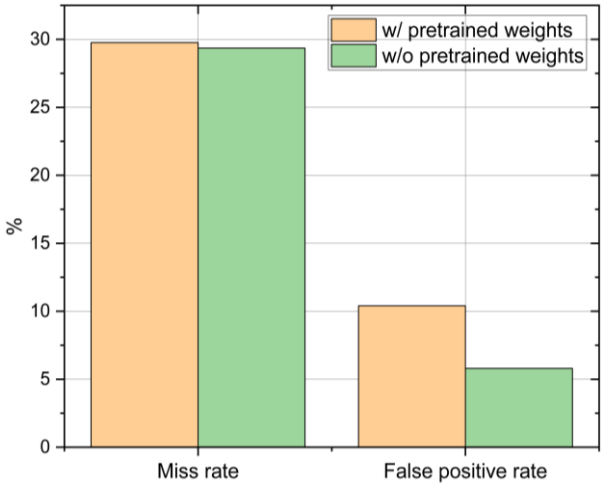

Fig. 2 Comparison of Miss rate and false positive rate for custom ATSADR dataset.

From Fig. 2, it is evident that both models exhibit a higher miss rate, largely attributed to lower recall rates for specific classes such as left-turn, speed breaker, ten kmph, and 30 km/h. However, the model trained without pretrained weights showed a significantly higher recall rate for 30 km/h, resulting in a lower miss rate (29.36%). Additionally, the false positive rate (computed using equation (12)) is considerably lower for the model without pretrained weights (5.8%) than for the model with pretrained weights (10.4%), which is nearly two times higher.

3.4 Implementation on embedded hardware

The trained YOLOv8n model (with and without pretrained weights) was implemented on embedded hardware (Raspberry Pi 4 B). The Raspberry Pi 4 B features a Broadcom BCM 2711 system-on-chip with a quad-core cortex-A72 (ARM v8) 64-bit processor clocked at 1.8GHz and an LPDDR4 RAM of 8GB running on a 64-bit Raspberry Pi operating system. The Broadcom BCM2711 SoC features an upgraded GPU (VideoCore VI clocked at 500 MHz) with a memory of 76MB. As mentioned earlier, the YOLOv8n model training environment utilizes packages, such as PyTorch. Therefore, these essential packages must be installed on Raspberry Pi to run the model. However, the Raspberry Pi supports specific versions of this Pytorch package (*i.e.*, torch version 2.0.1, torchvision version 0.15.2, and torchaudio version 2.0.2). After installing these packages, YOLOv8n runs on the embedded hardware.

The weights of the trained YOLOv8n model were imported to Raspberry Pi and tested on the test images, in addition to testing the model with real-time images obtained during the experimental investigation. When tested with images having a 2633 × 2697 pixel resolution, the PyTorch model (with pretrained weights) on the Raspberry Pi 4 B recognizes the signs at a 0.36 FPS, whereas for a 640 × 480-pixel resolution, the execution speed was 0.57 FPS. On the other hand, when tested without pretrained weights PyTorch model on Raspberry Pi, the execution speed was 0.59 FPS and 0.79 FPS for 2633x2697 pixel, and 640x480 pixel resolution images respectively. It can be noted that on Raspberry Pi 4 B, the model without pretrained weights performs faster; however, the PyTorch model is not suitable for real-time execution on the embedded hardware. Therefore, the PyTorch model has been converted into the NCNN model for its faster performance.[21] Fig. 3 shows the execution speed of YOLOv8n on the embedded hardware.

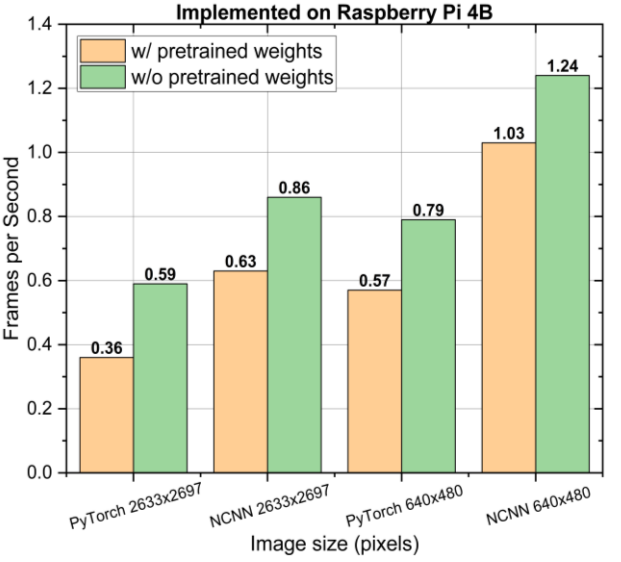

Fig. 3 Execution speed of YOLOv8n model on Raspberry Pi 4 B.

3.5 PyTorch model to NCNN model conversion

The NCNN is a lightweight, high-performance neural network inference framework optimized for mobile platforms, developed by Tencent. It was designed to be minimalistic, making it particularly suitable for embedded and mobile applications. The framework was optimized for ARM-based processors and other mobile hardware to ensure fast inference speeds. NCNN supports various platforms, including Android, iOS, Linux, and Windows, and can convert models from popular frameworks such as Caffe, TensorFlow, ONNX, and PyTorch. It provides straightforward APIs, facilitating the

integration of neural network models into applications. This combination of features makes NCNN an ideal choice for deploying sophisticated AI models in real-time image and video processing, object detection and recognition, natural language processing, augmented reality, and other AI-based mobile applications. After training the YOLOv8n, the model's weights will be saved in the $'model\_path'$. Using the Python code snippet, as given in (13), the model is selected and exported as the NCNN model.

$$model = YOLO(model\_path)$$
$$model.export(format = "ncnn") \quad (13)$$

The exported NCNN model was used for inference in Raspberry Pi 4B embedded hardware. It was observed that the NCNN model performed faster than PyTorch. From Fig. 3, it can be seen that the execution speed was 0.86 FPS and 1.24 FPS for 2633 × 2697 pixels and 640 × 480-pixel resolution images, respectively, which are nearly 1.5 times faster than the PyTorch model.

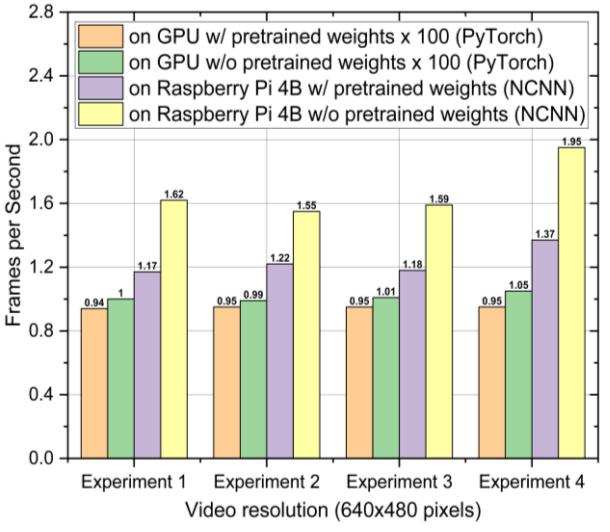

Fig. 4 Comparison of execution speed of YOLOv8n model on GPU and Raspberry Pi 4 B for real-time experiments.

When compared with the laptop GPU (4 GB Nvidia RTX 3050Ti), the Raspberry Pi 4B's 76MB GPU underperforms even for the NCNN model. From Fig. 4, it can be seen that the laptop GPU implementation reaches approximately 100 FPS for the model without pretrained weights, whereas that with the pretrained model reaches 95 FPS. On the other hand, the NCNN implementation on the embedded hardware reaches approximately 1.67 FPS (without pretrained weights), whereas the with pretrained weights model reaches 1.23 FPS. Although the NCNN model performs faster than the PyTorch model on the embedded hardware, it is not suitable for real-time implementation since the Raspberry Pi uses limited GPU memory. Additionally, the benchmark inference time for the NCNN model on Raspberry Pi 4B with a 640×480 pixel image is approximately 415 ms (*ie.*, 2.41 FPS),[21] whereas our YOLOv8n model's maximum inference speed is 1.67 FPS. One possible reason for this performance drop compared to the benchmark is thermal throttling. It was observed that the CPU temperature reached around 56 ℃ when running the experiment even though the Raspberry Pi was provided with a heat sink and cooling fan.

Moreover, it may be noted that the laptop GPU executes ATSADR at 100 FPS on video; however, when tested with images, the execution speed reaches approximately 35 FPS. This is mainly due to the batch processing of video data, less overhead of image loading, and better GPU utilization and memory management. When processing a video, the algorithm can often handle frames in batches, leading to improved parallelization and more efficient use of the GPU. This batch processing significantly increased the inference time per frame. In contrast, processing individual images introduces additional overhead related to loading and preprocessing each image separately, reducing the overall throughput compared with processing a continuous stream of video frames. Videos provide a steady stream of data, allowing the GPU to maintain a high utilization. By contrast, processing individual images can result in idle periods between images, leading to a lower overall efficiency. Additionally, handling a continuous stream of video frames is more memory efficient than managing multiple individual images as the latter involves repeated memory allocation and deallocation, further increasing the processing time.

4. Discussion

According to the results presented in the previous section, the YOLOv8n model trained without pretrained weights outperformed the model trained with pretrained weights on the custom ATSADR dataset. The training and validation losses of the YOLOv8n model trained without pretrained weights are shown in Fig. 5. As discussed in Section 2.2, the loss function is a weighted sum of the class, box, and distribution focal losses. Examining Figs. 5(a), (b), and (c), it is evident that the loss follows an exponentially decreasing trend. Initially, the loss was high at the start of training, but gradually decreased as the training progressed. Furthermore, the loss curve (BLUE line) is smoother and free of spikes, thereby ensuring high-quality training. Figs. 5(d), (e), and (f) illustrate the validation loss. Similar to the training, the validation loss also exhibited an exponentially decreasing trend, although the magnitude of the loss was smaller than that of the training. Moreover, although the validation loss curve is not as smooth as the training loss curve, there are no sudden spikes, which ensures that the model does not experience overfitting. Therefore, the

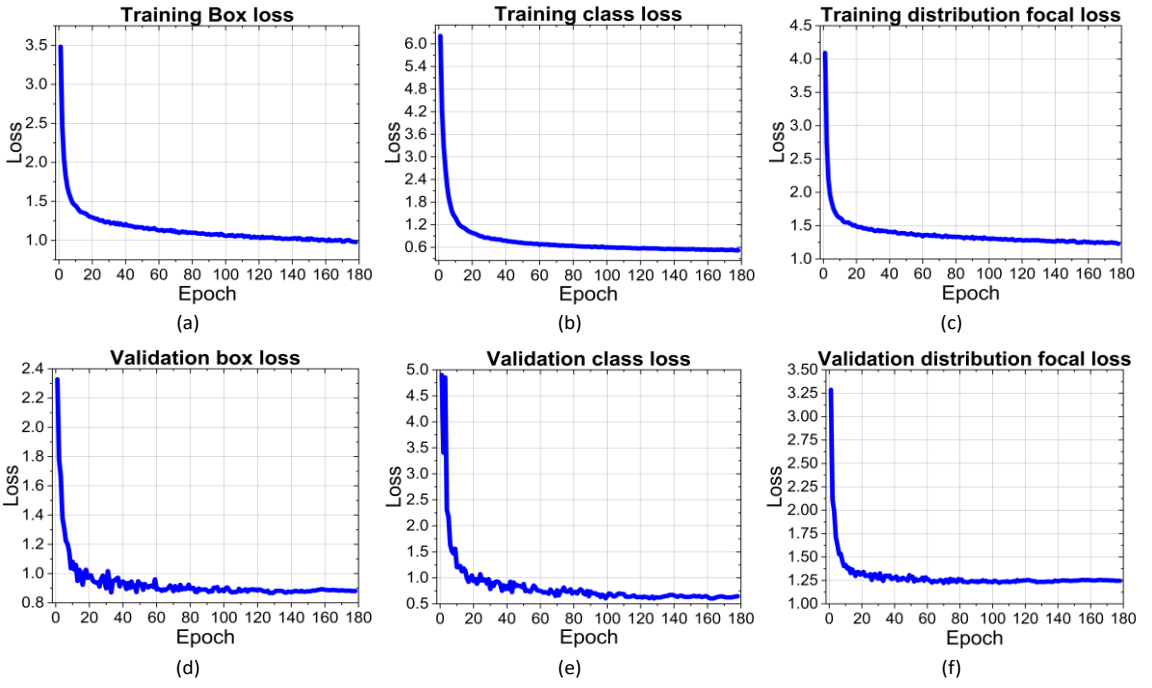

Fig. 5 Training and validation losses. (a), (b), and (c) show the box loss, class loss, and distribution focal loss during training, respectively, and (d), (e), and (f) show the box loss, class loss, and distribution focal loss during validation, respectively.

non-smooth trend between the iterations was normal. Eventually, the loss converges to a lower value than that of training.

Figure 6 illustrates the trends in precision and recall during the training process. At the beginning of the training, both metrics show a rapid increase owing to the model's quick adaptation (rapid adjustments to weights) to the data. However, between the 10th and 70th epochs, the model exhibited oscillation between underfitting and overfitting as it began to learn from the dataset. After the 70th epoch, both the precision and recall stabilize, reaching a higher value, which confirms that the model's learning has stabilized and achieved optimal weights.

Figure 7 illustrates the trend of mAP at the intersection over union (IOU) thresholds of 50% and 50-95%. It is evident from the curve that the map follows a pattern similar that to of precision and recall, showing a rapid increase during the initial stages of training. However, as the model learns from the dataset, mAP stabilizes as the training progresses and finishes at a higher value. The difference in magnitude between mAP@50 and mAP@50-95 can be attributed to the fact that the former considers only those bounding box predictions that have at least 50% overlap with the ground truth as true positives, whereas the latter provides a more comprehensive evaluation of the performance of the model across varying levels of localization precision. The mAP@50 reached 95.1%, as shown in Table 4, and mAP@50-95 achieved 73.7%, as shown in Table 2, which is better than the existing methods.

Table 4 shows a comparison with the existing models for the ATSADR dataset. It should be noted that the YOLOv8n

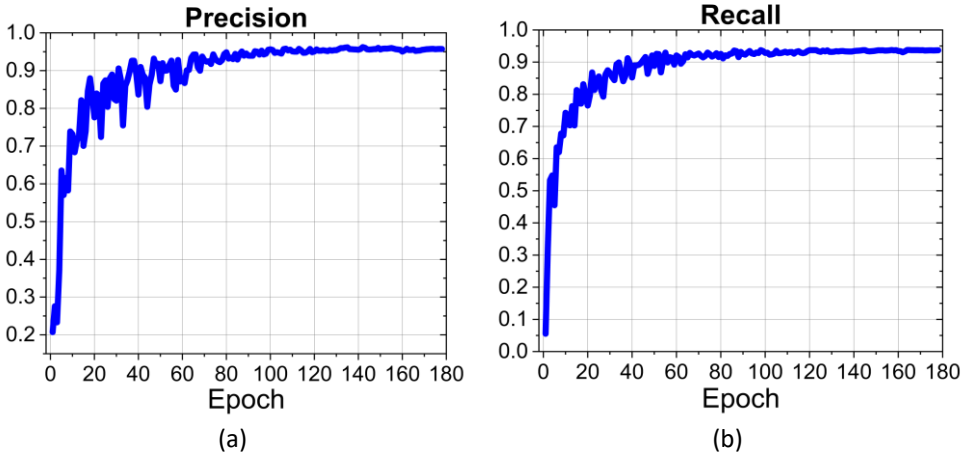

Fig. 6 Precision and recall during training.

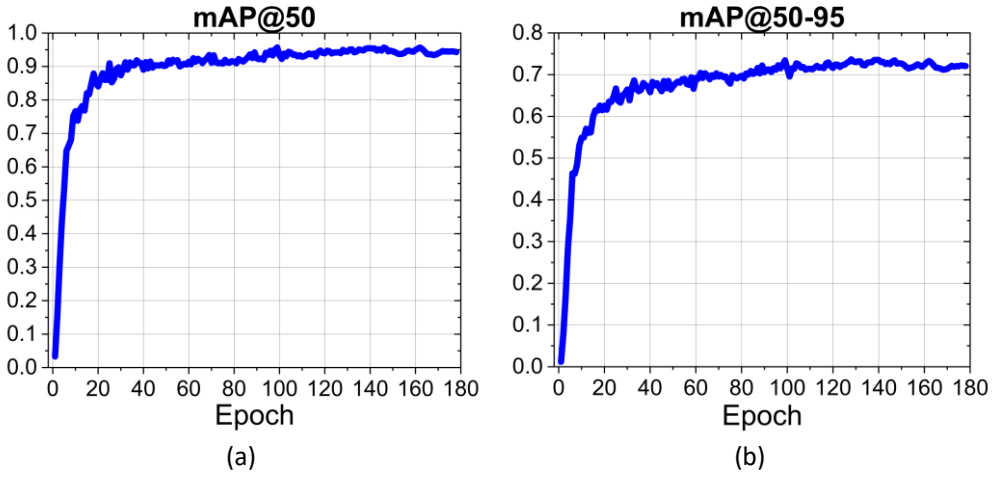

Fig. 7 mAP@50 and mAP@50-95 during training.

Table 4. Precision, Recall, FPS, mAP-50, and F1-Score comparison with existing models.

Reference	Models	mAP@50 (%)	FPS	Precision (%)	Recall (%)	F1-Score (%)
Shukla *et al*., (2019)	Faster RCNN	80.6	8	93	92	89.4
Gong *et al*., (2022)	YOLOv3 (without pretrained weights)	90.4	34	91.4	92.3	90.8
Zhu *et al*., (2022)	YOLOv5 (without pretrained weights)	92.6	35	93.9	91.6	91.3
Megalingam *et al*., (2023)	Mask RCNN	86.2	28	94.4	93.7	90.5
Present study	YOLOv8n (without pretrained weights)	95.1	100	95.8	93.6	92
	YOLOv8n (without pretrained weights)	97.3	95	91.8	93.5	90.3

model without pretrained weights shows improved performance compared to the other YOLO variants, Faster R-CNN, and Mask R-CNN. Moreover, the YOLOv8 nano model recognizes traffic signs and animals at a higher speed owing to its fewer parameters with a significant improvement in 50% mean Average Precision. Additionally, it is worth noting that while the recall rate is comparable for Mask R-CNN and YOLOv8n, the precision of YOLOv8n without pretrained weights demonstrates a slight advantage, thus resulting in a higher F1-Score. Furthermore, both mAP@50 and mAP@50-90 were marginally higher for the model with pretrained weights, which is attributable to the aggregation of all classes. Conversely, the model without pretrained weights exhibits a balanced performance and shows an average mAP improvement of 7% compared to the other models. It can be noted that the 100 FPS achieved without the pretrained weight model was for video implementation; for images, the inference speed was 35.34 FPS. Additionally, it was found that the YOLOv8n model detects traffic signs at an approximate distance of 30 to 35 m ahead of the vehicle, and the measured pixel size of the traffic sign is 11 × 14 pixels. Therefore, it can be noted that, if the actual size of the traffic sign is less than 11 × 14 pixel size on the captured image, that makes the sign to be unrecognized by the YOLOv8n.

It should be noted that as the application (present study) expands, scalability challenges may emerge, including increased computational demands for processing larger datasets and real-time inference as well as heightened requirements for storage and memory capacity. To address these obstacles, leveraging distributed computing frameworks and cloud-based solutions holds promise in bolstering an application's scalability through parallel data processing and efficient resource allocation. Moreover, optimizing the model architecture for parallelized training and inference coupled with the implementation of data-sharing techniques can effectively alleviate the scalability concerns. Additionally, incorporating an auto-scaling infrastructure and efficient caching mechanisms is key to ensuring seamless performance during periods of high demand, ultimately facilitating application growth while upholding high levels of reliability

and responsiveness.

In the future, integrating advanced technologies, such as geofencing, augmented reality overlays, and real-time data augmentation, can significantly enhance the effectiveness of the application. Additionally, leveraging multisensory fusion with a camera and LiDAR could improve the capability of the model to detect traffic signs and animals under diverse environmental conditions. Thus, feature-based (camera) and geometry-based (LiDAR) approaches can be integrated for an effective ATSADR. In particular, this approach is more effective for traffic sign recognition, considering the geometry is similar among the traffic signs (triangles, circles, and rectangles). Moreover, the exploration of federated learning approaches across decentralized datasets from varying geographical locations can enhance the adaptability of the model to real-world scenarios. Hence, it is possible to regulate parameters such as vehicle acceleration/deceleration and braking of autonomous vehicles by augmenting the vehicle control algorithms for safer navigation. Furthermore, it should be noted that the ATSADR is not interoperable with existing healthcare infrastructure. However, its functionality can be enhanced by integrating it with the institute's hospital system, enabling the application to notify relevant authorities in the event of a detected road accident, thus improving its overall effectiveness and relevance in the healthcare setting.

5. Conclusion

This study presents a deep learning-based approach using YOLOv8n for automatic traffic signs, animal detection, and recognition (ATSADR), focusing on mitigating human-animal conflicts on roads. Comprehensive testing and analysis revealed the effectiveness of the selected YOLOv8n model in real-time scenarios, with promising applications for enhancing road safety. A thorough evaluation of both the pretrained and non-pretrained models demonstrated the superior performance of YOLOv8n. However, a model trained without pretrained weights showed improved precision and computational efficiency, especially in certain critical classes, which makes it an appropriate model for ATSADR. The results confirm the potential of the selected model to significantly reduce the miss rate (5.8%) and false-positive rates, thereby contributing to enhancing road safety. With a higher accuracy (99.3%) and efficient computational performance (100 FPS on video data and 35 FPS on image frames) in detecting traffic signs and animals, particularly smaller wildlife such as baby monkeys and deer, the model presents a proactive solution to alert drivers and prevent road accidents. The findings suggest that the YOLOv8n model trained without pretrained weights holds promise for practical implementation in addressing human-animal conflicts and improving overall road safety. The present work sets the stage for pioneering advancements in exploiting deep learning to tackle intricate real-world challenges such as road safety and wildlife conservation. Future work will integrate the identified traffic signs/animals into vehicle control algorithms to enable autonomous vehicle navigation with enhanced safety.

Conflict of Interest

There is no conflict of interest.

Supporting Information

Applicable.

References

[1] X. R. Lim, C. P. Lee, K. M. Lim, T. S. Ong, A. Alqahtani, M. Ali, Recent advances in traffic sign recognition: approaches and datasets, *Sensors*, 2023, **23**, 4674, doi: 10.3390/s23104674.

[2] M. S. Prieto, A. R. Allen, Using self-organising maps in the detection and recognition of road signs, *Image and Vision Computing*, 2009, **27**, 673-683, doi: 10.1016/j.imavis.2008.07.006.

[3] U. Zakir, I. Zafar, E. A. Edirisinghe, Road sign detection and recognition by using local energy-based shape histogram (LESH), *International Journal of Image Processing*, 2011, **4**, 566–582.

[4] R. Laguna, R. Barrientos, L. F. Blázquez, L. J. Miguel, Traffic sign recognition application based on image processing techniques, *IFAC Proceedings Volumes*, 2014, **47**, 104-109, doi: 10.3182/20140824-6-za-1003.00693.

[5] Y. Ouerhani, A. Alfalou, M. Desthieux, C. Brosseau, Advanced driver assistance system: road sign identification using VIAPIX system and a correlation technique, *Optics and Lasers in Engineering*, 2017, **89**, 184-194, doi: 10.1016/j.optlaseng.2016.05.020.

[6] X. Xu, J. Jin, S. Zhang, L. Zhang, S. Pu, Z. Chen, Smart data driven traffic sign detection method based on adaptive color threshold and shape symmetry, *Future Generation Computer Systems*, 2019, **94**, 381-391, doi: 10.1016/j.future.2018.11.027.

[7] C. G. Kiran, L. V. Prabhu, A. R. V., K. Rajeev, Traffic sign detection and pattern recognition using support vector machine, *International Conference on Advances in Pattern Recognition*, 2009, 87-90, doi: 10.1109/ICAPR.2009.58.

[8] Y. Wu, Y. Liu, J. Li, H. Liu, X. Hu, Traffic sign detection based on convolutional neural networks, *International Joint Conference on Neural Networks*, 2013, 1-7, **doi:** 10.1109/IJCNN.2013.6706811.

[9] Z. Zuo, K. Yu, Q. Zhou, X. Wang, T. Li, Traffic signs detection based on faster R-CNN, *International Conference on Distributed Computing Systems Workshops*, 2017, 286-288, doi: 10.1109/ICDCSW.2017.34.

[10] T. Yang, X. Long, A. K. Sangaiah, Z. Zheng, C. Tong, Deep detection network for real-life traffic sign in vehicular networks, *Computer Networks*, 2018, **136**, 95-104, doi: 10.1016/j.comnet.2018.02.026.
[11] L. Wu, H. Li, J. He, X. Chen, Traffic sign detection method based on Faster R-CNN, *Journal of Physics: Conference Series*, 2019, **1176**, 032045, doi: 10.1088/1742-6596/1176/3/032045.
[12] U. Shukla, A. Mishra, S. G. Jasmine, V. Vaidehi, S. Ganesan, A deep neural network framework for road side analysis and lane detection, *Procedia Computer Science*, 2019, **165**, 252-258, doi: 10.1016/j.procs.2020.01.081.
[13] J. Cao, C. Song, S. Peng, F. Xiao, S. Song, Improved traffic sign detection and recognition algorithm for intelligent vehicles, *Sensors*, 2019, **19**, 4021, doi: 10.3390/s19184021.
[14] A. Avramović, D. Sluga, D. Tabernik, D. Skočaj, V. Stojnić, N. Ilc, Neural-network-based traffic sign detection and recognition in high-definition images using region focusing and parallelization, *IEEE Access*, 2020, **8**, 189855-189868, doi: 10.1109/ACCESS.2020.3031191.
[15] F. Franzen, C. Yuan, Z. Li, Traffic sign recognition with neural networks in the frequency domain, *Journal of Physics: Conference Series*, 2020, **1576**, 012015, doi: 10.1088/1742-6596/1576/1/012015.
[16] H. Wan, L. Gao, M. Su, Q. You, H. Qu, Q. Sun, A novel neural network model for traffic sign detection and recognition under extreme conditions, *Journal of Sensors*, 2021, **1**, 9984787, doi: 10.1155/2021/9984787.
[17] C. Gong, A. Li, Y. Song, N. Xu, W. He, Traffic sign recognition based on the YOLOv3 algorithm, *Sensors*, 2022, **22**, 9345, doi: 10.3390/s22239345.
[18] Y. Zhu, W. Q. Yan, Traffic sign recognition based on deep learning, *Multimedia Tools and Applications*, 2022, **81**, 17779-17791, doi: 10.1007/s11042-022-12163-0.
[19] R. K. Megalingam, K. Thanigundala, S. R. Musani, H. Nidamanuru, L. Gadde, Indian traffic sign detection and recognition using deep learning, *International Journal of Transportation Science and Technology*, 2023, **12**, 683-699, doi: 10.1016/j.ijtst.2022.06.002.
[20] P. DRAP, Automatic deep-sea amphorae detection using optimal 2D ultralytics deep learning, *International Journal of Computing and Digital Systems*, 2024, **17**, 1-11.
[21] D. Danopoulos, G. Zervakis, K. Siozios, Adapt: Fast emulation of approximate dnn accelerators in pytorch, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022, **42**, 2074-2078, doi: 10.1109/TCAD.2022.3212645.