Issue 16, 2024

УДК 004.8

INTELLIGENT DRIVER ASSISTANCE SYSTEMS BASED ON COMPUTER VISION AND DEEP LEARNING

Artem Teliuk[1], Andrii Vasyliuk [2], Andrii Khudyi[3]

[1–3] Lviv Polytechnic National University,
Department of Information Systems and Networks, Lviv, Ukraine
[1] E-mail: artem.teliuk.mnsam.2023@lpnu.ua, ORCID 0009-0000-1180-8492
[2] E-mail: andrii.s.vasyliuk@lpnu.ua, ORCID 0000-0002-3666-7232
[3] E-mail: andrii.m.khudyi@lpnu.ua, ORCID 0000-0003-2029-7270

This article presents an integrated Advanced Driver Assistance System (ADAS) that combines several key functional modules, such as collision warning, lane detection, traffic sign recognition, and pothole detection, which are implemented using modern deep learning models, particularly YOLOv8n. The system is optimized for devices with limited computational resources, such as Raspberry Pi or NVIDIA Jetson Nano, by employing a modular architecture and parallel data processing to ensure real-time performance. This research provides an overview of existing ADAS solutions and proposes new approaches that significantly enhance the efficiency of such systems. Key innovations include an efficient approach to lane detection based on object detection models, real-time traffic sign recognition with a flexible extraction and classification process, and a novel pothole detection system optimized for dashcam recordings. Additionally, the proposed driver alert system, which uses an LED strip, allows for intuitive hazard awareness without distracting the driver. Preliminary results confirm satisfactory detection accuracy across all components, although further optimization is required for successful deployment on low-resource devices.

Keywords: intelligent driver assistance system, object detection, deep learning, real-time, YOLOv8n, lane detection, traffic signs, potholes.

Introduction. Relevance of the Topic

Advanced Driver Assistance Systems (ADAS) play a critical role in improving road safety by assisting drivers with real-time information. Informational systems within ADAS are particularly important, as they provide the driver with critical alerts and data without directly intervening in vehicle control. Unlike autonomous driving systems, which involve complex issues of safety, responsibility, and integration, informational systems place the responsibility in the hands of the driver, enhancing situational awareness. The primary goal is to provide drivers with accurate, real-time information that they may otherwise overlook, improving decision-making and reaction times on the road. The continuous improvement of driving safety remains a global priority, particularly in the development of ADAS solutions. While full automation promises to eliminate human error, current technology still relies heavily on human drivers. Therefore, the need to refine and enhance ADAS informational systems is essential to reducing accidents and improving driver awareness. By providing timely and precise information on potential hazards, ADAS systems can greatly enhance road safety, particularly if they are made accessible and cost-effective for all vehicle types.

Problem Statement

Despite the existing advancements in ADAS technologies, certain subsystems, such as traffic sign recognition and pothole detection, remain underdeveloped or underutilized in real-time applications. Additionally, lane detection, vehicle detection, and pedestrian detection systems often require significant computational resources, limiting their deployment on hardware with restricted processing capabilities, such as those found in low-cost vehicles. The challenge lies in developing an integrated system that can perform these tasks efficiently on devices with limited computational power, without sacrificing accuracy or speed.

Objectives of the Research

The objective of this article is twofold: to explore and evaluate existing methods within ADAS systems, and to propose new, optimized approaches to improve their efficiency and accuracy. Specifically, we focus on improving lane detection, traffic sign recognition, and pothole detection by employing deep learning models like YOLOv8n. This research aims to create scalable and efficient ADAS informational systems that can operate on resource-constrained devices, making them accessible for a wide range of vehicles. Additionally, the article introduces a modular, parallel-processing system architecture to enhance real-time performance, laying the groundwork for future feature expansions.

Literature Review

The development of Advanced Driver Assistance Systems (ADAS) has seen significant advancements in lane detection, object recognition, and road surface monitoring, which are critical for improving driver awareness and safety. Lane detection, a fundamental ADAS component, has faced challenges in non-ideal conditions such as poor weather and irregular road markings. Kaur and Kumar [1] identified the limitations of traditional methods like the Hough transform, which struggle with accuracy in real-world environments. To address these issues, Rachel et al. [3] introduced a CNN-based approach that enhances detection accuracy, achieving over 97% accuracy even in complex conditions. Saha et al. [2] proposed a flood-fill technique that adapts well to varying daylight but still faces limitations under extreme conditions.

Object detection plays a pivotal role in ADAS, particularly for collision warning systems. Murthy et al. [4] utilized YOLOv5 to develop a real-time object detection framework capable of efficiently detecting pedestrians, vehicles, and other road obstacles. This method outperforms older models like R-CNN in terms of both speed and accuracy, making it better suited for real-time applications where quick responses are critical [4].

Traffic sign recognition is another essential ADAS function, as it ensures that drivers receive timely information. Golgire [8] explored the use of CNNs for traffic sign recognition, demonstrating their ability to handle poor visibility and varying light conditions with minimal manual preprocessing. This research showed that CNNs, when optimized through hyperparameter tuning, can operate efficiently on limited-resource hardware, which is crucial for real-time ADAS applications [8]. However, in lower-cost ADAS solutions, real-time traffic sign detection is less common, as many rely on preloaded databases instead of live detection [19].

Pothole detection, a more recent addition to ADAS, addresses the need for road surface monitoring. Buza et al. [5] developed a cost-effective method using image processing and spectral clustering, achieving an 81 % accuracy in identifying pothole regions. Joe et al. [6] took this further by integrating pothole detection with Tiny YOLOv3 into a mobile application, enabling both municipalities and citizens to monitor road conditions in real-time. However, challenges in achieving consistent accuracy under varied

lighting conditions remain [6]. Furthermore, smartphone-based ADAS options for pothole detection also exhibit limitations, indicating a gap in real-time capability compared to more advanced OEM systems [19].

In summary, these studies highlight the evolution of ADAS technologies, where traditional methods provide foundational tools, and AI-driven models such as CNNs and YOLO introduce significant improvements in real-time performance, accuracy, and adaptability. Although progress has been made, there are still challenges in optimizing these systems for resource-constrained environments, where maintaining both speed and accuracy is critical. While higher-end models offer robust ADAS functionalities, there remain near-market gaps in real-time detection and reliability, especially in affordable or aftermarket ADAS solutions [17–19].

Research Results

Collision Warning System

The collision warning system is an important part of Advanced Driver Assistance Systems (ADAS) that helps keep drivers safe by giving real-time alerts about possible collisions. This system uses a mix of sensors, visual tech, and machine learning models to detect obstacles and figure out the chance of a collision, letting the driver know when they need to act. In this section, we'll focus on visual technologies and explain the tools, data flow, and algorithms used, with a special look at machine learning and model structures like YOLO.

1.1.1. Technologies and Hardware Used in Collision Warning Systems

The backbone of any collision warning system is its sensor suite. While some systems may incorporate radar or LiDAR sensors, we focus on camera-based visual technologies due to their cost-effectiveness and adaptability to various environments. A monocular camera mounted on the front of the vehicle is commonly used to capture real-time video streams, which serve as the primary input for detecting obstacles and estimating distances. In more advanced systems, stereo cameras can provide depth information, but monocular setups, combined with machine learning, offer a simpler yet effective solution.

Key hardware components include a **monocular camera**, which captures images or video in real-time and serves as the primary data source. A **processing unit**, such as an NVIDIA Jetson, handles the real-time data processing using machine learning models. Finally, the **display or audio system** provides visual or auditory warnings to the driver when a potential collision is detected.

1.1.2. Data Flow from Input to Warning

In a collision warning system, the process starts with a camera that takes continuous video of the road ahead, which is the main source of information. This raw video goes through some basic processing steps, like adjusting colors, resizing, and reducing noise, so the images are clearer and ready for analysis. After this, the images go into an object detection model, like YOLO, which looks at each frame and finds objects, like cars, people, and other obstacles, and keeps track of them.

After detecting objects, the system works out the distance between the vehicle and each object it finds. This can be done with advanced machine learning models for depth estimation or simpler methods like perspective transformation, especially if there's just one camera. Once the distances are calculated, the system checks how likely a collision is by looking at each object's distance, speed, and path compared to the vehicle's own movement.

If the system thinks a collision might happen, it sends a warning to the driver. This alert could be visual, like a symbol on the dashboard, or a sound, like a beep or a voice message.

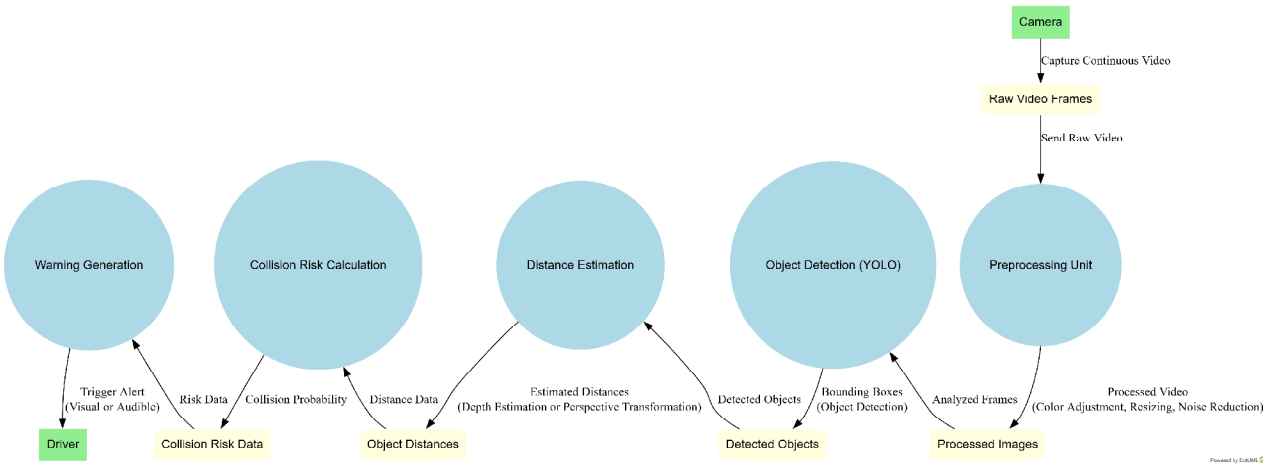


Fig. 13. Data Flow Diagram for Collision Warning System (in Yourdon and Coad notation)

1.1.3. Object Detection with YOLO

Object detection is at the heart of the collision warning system. *The You Only Look Once (YOLO)* model is one of the most widely used architectures due to its efficiency in detecting objects in real-time. YOLO divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously, allowing for rapid processing of video frames without sacrificing accuracy.

YOLO is the ideal candidate model for collision warning due to several key factors. First, **speed** is a significant advantage, as YOLO can process images quickly, achieving up to 60 frames per second on high-end GPUs, which is critical for real-time applications like collision warning. Second, **efficiency** makes YOLO stand out; as a single-stage detector, it is computationally less intensive compared to two-stage object detectors like R-CNN, while still maintaining high detection accuracy. Finally, **robustness** is another strength, as YOLO can detect a wide range of objects, such as vehicles, pedestrians, and road obstacles, even under various lighting and weather conditions.

The structure of YOLO includes three main steps. First, the input image is split into a grid of cells. Next, for each cell, YOLO predicts a few bounding boxes along with class probabilities. Finally, it uses a process called non-max suppression to remove overlapping boxes, keeping only the predictions with the highest confidence.

YOLOv8 demonstrates significant efficiency when deployed on resource-constrained devices such as the Raspberry Pi 4, making it an ideal choice for embedded computer vision tasks. The Raspberry Pi 4, powered by the Broadcom BCM2711 SoC and equipped with up to 8GB of LPDDR4 RAM, can run YOLOv8 models at satisfactory frame rates. Benchmark tests conducted on the Raspberry Pi 4 using the YOLOv8n model in NCNN format have shown an inference time of 414.73 ms per image, while other formats such as PyTorch exhibit slower performance. The table below provides an overview of the inference times for various formats:

Table 4

Inference times for YOLOv8n on Raspberry Pi 4

Format	mAP50-95(B)	Inference Time (ms/im)
PyTorch	0.6092	1068.42
ONNX	0.6092	560.04
OpenVINO	0.6092	534.93
NCNN	0.6092	414.73

Among the tested formats, NCNN provides the fastest inference times on the Raspberry Pi 4, making it the most suitable format for real-time applications. This optimization is particularly beneficial for object detection tasks in ADAS systems, where timely responses are critical.

1.1.4. Distance Estimation for Collision Warning

One of the key tasks for collision warning systems is to estimate the distance between the vehicle and detected objects. A widely used approach for distance estimation is perspective transformation. This method leverages the geometric properties of a monocular camera to estimate distances based on the size and position of detected objects in the image. It requires an initial *calibration* of the camera setup (e.g., height of the camera, focal length) but once calibrated, it is faster and less resource-intensive than deep learning-based methods.

The process of perspective transformation in image processing is essential for applications such as distance estimation, where a top-down view of the road is required for accurate calculations. To achieve this, a perspective transformation is applied to the original image using a predefined set of points. This section outlines the mathematical foundation of the transformation process, with a focus on its application to camera-based systems.

Calibration and Four-Point Mapping: The transformation begins with the calibration of the camera system, which involves identifying four key points in the image that correspond to known real-world coordinates. These four points form the basis of the perspective transformation. Let the points in the image plane be represented by:

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2), \quad P_3 = (x_3, y_3), \quad P_4 = (x_4, y_4) \tag{1}$$

These points are mapped to a set of target points in the transformed plane, denoted by:

$$P'_1 = (x'_1, y'_1), \quad P'_2 = (x'_2, y'_2), \quad P'_3 = (x'_3, y'_3), \quad P'_4 = (x'_4, y'_4) \tag{2}$$

The objective is to construct a matrix that maps the points P_i to P'_i, thereby transforming the perspective of the image.

Homography Matrix Construction: The relationship between the original points P and the transformed points P' is governed by a homography matrix H, which is a 3×3 matrix. The transformation can be described by the equation:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3}$$

where H is defined as:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}. \tag{4}$$

This matrix maps the coordinates from the input image to the transformed plane. The transformation is nonlinear due to the presence of $h_{31}x + h_{32}y + 1$ in the denominator, ensuring that straight lines remain straight, but angles and lengths are not preserved, resulting in a change in perspective.

Solving for the Homography Matrix: The matrix H can be computed by solving a system of linear equations. For each pair of corresponding points $P_i \leftrightarrow P'_i$, two equations are derived:

$$x'_i = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + 1}, \tag{5}$$

$$y'_i = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + 1}. \tag{6}$$

These equations can be expanded into a system of eight linear equations with eight unknowns (the elements of H). The system can be expressed as:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4 x_4 & -x'_4 y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4 x_4 & -y'_4 y_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ \vdots \\ x'_4 \\ y'_4 \end{pmatrix}. \tag{7}$$

Once the homography matrix H is solved, it can be applied to transform the entire image.

Applying the Perspective Transformation: With the homography matrix H computed, the transformation of any point (x, y) in the original image can be carried out using:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}, \tag{8}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}. \tag{9}$$

This transformation is applied to each pixel in the image, resulting in a new perspective where the camera's view is warped into a top-down, bird's-eye view. This is especially useful for distance estimation in ADAS systems, as it simplifies calculations by creating a linear relationship between pixel coordinates and real-world distances.

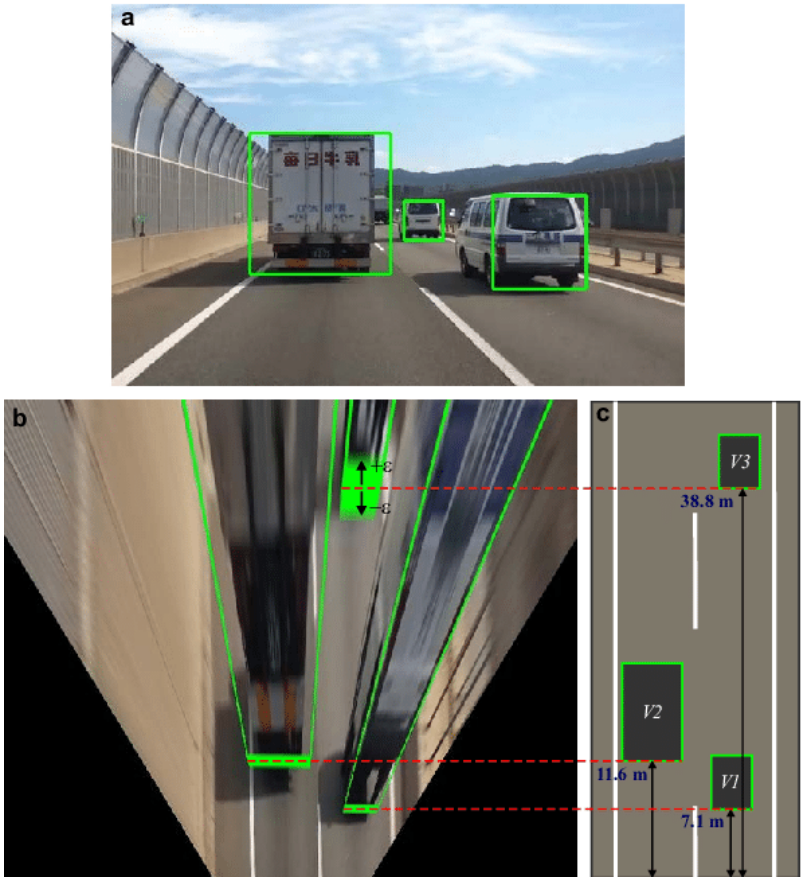


Fig. 2. Getting distance information from perspective transformation [13]

1.1.5. Object Tracking and Time to Collision (TTC)

Object tracking is essential in maintaining awareness of detected objects over time, especially for collision warning systems. High-speed trackers like the Kalman filter and SORT are often used for their ability to efficiently predict object motion in real-time. These algorithms help the system continuously estimate the position and velocity of dynamic objects across consecutive frames, enabling accurate predictions of potential collisions.

In an ego vehicle, tracking focuses on determining whether a detected object's trajectory will intersect with the vehicle's path. If an object is moving toward the vehicle, the system calculates the Time to Collision (TTC), which is the estimated time before impact. TTC is computed using the formula:

$$\mathrm{TTC} = \frac{\mathrm{D}}{\mathrm{V_r}}, \tag{10}$$

where D is the distance between the vehicle and the object, and V_r is the relative velocity. If the TTC falls below a critical threshold (e.g., 2 seconds), a warning is issued.

By combining tracking algorithms and TTC calculation, the system ensures timely and accurate collision warnings, significantly improving road safety.

1.1.6. Delivering the Warning to the Driver

The last part in the collision warning system is giving a warning to the driver. The system must make sure the alerts show up at the right time and in a way that catches the driver's attention, but without distracting too much. The kind of warning depends on how serious the danger is and what's happening on the road at that time.

For **visual alerts**, the system might use a heads-up display (HUD) or show something on the dashboard to mark where the object is, with a warning sign. It could also show extra info, like time left before a possible crash or the speed of the object, so the driver gets all the important details. For **audio warnings**, the system might make beeping sounds or use voice alerts to warn the driver. These sounds are useful because they catch the driver's attention fast, which is important when a quick reaction is needed.

It is crucial for the system to minimize the number of **false positives**, as frequent unnecessary warnings can lead to driver fatigue or desensitization to the alerts. To achieve this, the system's algorithms must be carefully calibrated to balance sensitivity with accuracy, ensuring that warnings are only provided when the risk of collision is significant. Additionally, integrating multiple forms of alerts (visual, auditory, and haptic) can enhance the system's effectiveness by providing the driver with clear, multi-sensory feedback in the event of an imminent collision.

Lane Detection System

Lane detection is a fundamental component of Advanced Driver Assistance Systems (ADAS), responsible for identifying the lane boundaries to assist the vehicle in maintaining its position on the road. There are two primary approaches to lane detection: the conventional computer vision (CV) approach and the deep learning (DL) approach. Both methods have their respective advantages and challenges, and this section will discuss these approaches in detail.

1.1.7. Conventional Computer Vision Approach

The conventional CV approach relies on classical image processing techniques to detect lanes. The key stages of this approach involve image filtering, thresholding, and line-fitting algorithms to derive the lane boundaries. This method is computationally efficient and suitable for real-time applications but may struggle in complex or noisy environments.

In the conventional approach, the first step is preprocessing the input image to enhance lane features while reducing noise. Preprocessing typically involves multiple stages, including undistortion, cropping, blurring, and edge detection. Each of these steps is critical in preparing the image for lane detection.

1.1.1.1. Undistortion

One of the major challenges in lane detection is correcting for the distortion caused by the camera's field of view (FOV). Wide-angle lenses, often used in vehicle cameras, introduce radial and tangential distortion, causing straight lines to appear curved. To correct this, undistortion techniques are applied using camera calibration matrices. Given a distortion matrix D, the undistortion process maps each pixel coordinate $(x_{\text{dist}}, y_{\text{dist}})$ to a corrected coordinate $(x_{\text{undist}}, y_{\text{undist}})$ based on a camera matrix K (lane-detection-1):

$$\begin{pmatrix} x_{\text{undist}} \\ y_{\text{undist}} \\ 1 \end{pmatrix} = K^{-1} D \begin{pmatrix} x_{\text{dist}} \\ y_{\text{dist}} \\ 1 \end{pmatrix}, \tag{1}$$

where D represents the distortion coefficients, and K is the intrinsic camera matrix derived from camera calibration. Correcting the distortion ensures that lane lines appear straight and undistorted, which is essential for accurate lane detection.

1.1.1.2. Cropping

After undistortion, the next step is to focus on the region of interest (ROI), which is the portion of the image where the road is most likely to be located. This involves cropping the image to exclude unnecessary parts of the scene (e.g., sky, buildings) and focusing only on the road surface. Instead of simple cropping, the ROI is selected based on the geometry of the road, and pixels outside the expected road area are set to zero. This reduces the amount of irrelevant data that the algorithm processes, improving computational efficiency.

Given an image with height H and width W, and assuming the bottom half of the image contains the road, the cropping operation can be defined as:

$$\text{ROI}(x, y) = \begin{cases} I(x, y) & \text{if } y > \frac{H}{2} \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

where $I(x, y)$ is the pixel intensity at coordinates (x, y), and all values outside the ROI are set to zero.

1.1.1.3. Blurring and Edge Detection

Before detecting lane lines, the image is often smoothed to reduce noise and enhance the lane features. Gaussian blurring is commonly used for this purpose, where each pixel is replaced by the weighted average of its neighboring pixels. The Gaussian filter is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \tag{13}$$

where σ is the standard deviation of the Gaussian kernel. The result is an image with reduced noise, making it easier to detect edges without interference from small-scale variations in intensity.

After blurring, edge detection is performed to identify the boundaries of lane lines. The Sobel filter is frequently used for this purpose, where gradients in both the x and y directions are computed:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y}. \tag{14}$$

The magnitude of the gradient is then calculated to highlight the edges in the image:

$$G = \sqrt{G_x^2 + G_y^2}. \tag{15}$$

This step emphasizes the lane boundaries by enhancing the vertical edges, which are essential for detecting lanes.

1.1.1.4. HSL and Gradient Thresholding

To further isolate lane lines, thresholding is applied to the processed image. A combination of gradient thresholding and color-based thresholding in the HSL (Hue, Saturation, Lightness) color space is commonly used to focus on lane markings. The color thresholding in the HSL space highlights white and yellow lane lines, which are typically distinct from the road surface:

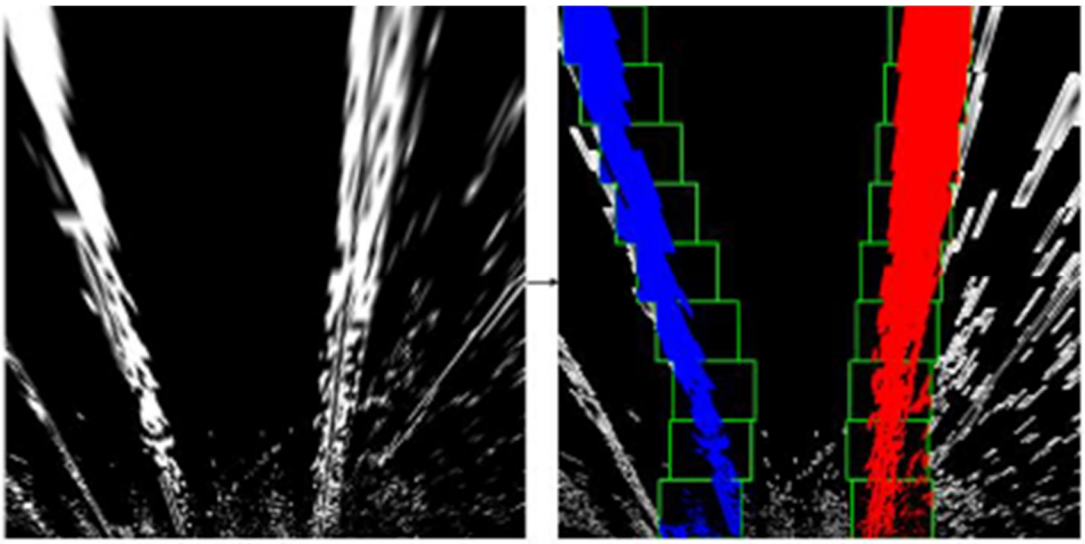
$$T_{\text{HSL}}(h,s,l) = \begin{cases} 1 & \text{if } h \in [H_{\min}, H_{\max}] \text{ and } s \in [S_{\min}, S_{\max}] \\ 0 & \text{otherwise} \end{cases}, \tag{16}$$

where h, s, l are the hue, saturation, and lightness components of the pixel, and $H_{\min}, H_{\max}, S_{\min}, S_{\max}$ are the thresholds that define the lane color range.

Finally, gradient thresholding is applied to focus on the high-intensity edges corresponding to lane lines. This step further reduces noise and non-lane objects in the image, ensuring that only lane-related features are passed on for further processing.

1.1.1.5. Sliding Window Search

The sliding window search technique is an efficient method used in lane detection to track lane lines across consecutive frames in video sequences. Once the lane lines are detected in an earlier frame, this information is used to guide the search in subsequent frames. A window is placed around the previously detected lane line centers, and the algorithm searches for lane line pixels within this window from the bottom to the top of the image. This method significantly reduces the search space, leading to faster processing and increased accuracy.



a *b*

Fig. 3. Sliding Window Search for detecting primary stage of lanes from perspective transform image: a) perspective transform image; b) sliding window search on perspective transform image [15]

The pixels corresponding to the left and right lane lines are identified using their x and y coordinates. Once these pixels are detected, a second-order polynomial is fitted to the data to represent the lane lines:

$$f(y) = ay^2 + by + c. \tag{17}$$

In this equation, $f(y)$ is used rather than $f(x)$, because in the warped image, the lane lines are nearly vertical. This orientation can result in multiple y-values corresponding to the same x-value, making $f(y)$ a more appropriate representation. The use of this polynomial fit allows for the lane lines to be tracked smoothly across frames, even if the road or camera conditions change, and ensures that the system maintains accurate lane detection over time.

1.1.1.6. Line Fitting and Outlier Detection

Once the binary image is obtained, lane lines are detected using line-fitting algorithms, such as the Hough Transform, which identifies lines in the image by transforming edge points into the Hough space. The Hough Transform is defined as:

$$\rho = x\cos\theta + y\sin\theta, \tag{18}$$

where ρ is the distance from the origin to the line, and θ is the angle between the x-axis and the line. Points in the Hough space that intersect frequently are considered to be part of a line.

Additionally, machine learning algorithms can be used to detect outliers in the lane points. By analyzing characteristics like color and position, the system can filter out noise and ensure accurate lane detection.

1.1.8. Deep Learning Approach

The deep learning approach to lane detection leverages neural networks, particularly segmentation models, to identify lanes. In this approach, a convolutional neural network (CNN) segments the input image into lane and non-lane regions, allowing the system to focus on relevant parts of the scene.

1.1.1.7. Segmentation Models

The segmentation model divides the image into several regions and labels each pixel according to whether it belongs to a lane line. The most common architectures for this task include U-Net and Fully Convolutional Networks (FCNs). These models are trained on large datasets, allowing them to generalize to a wide range of driving conditions. The output of the segmentation model is typically a binary mask, where pixels corresponding to lanes are set to 1, and all other pixels are set to 0. This output can then be processed similarly to the conventional CV approach, with line-fitting algorithms used to derive the final lane boundaries.

1.1.1.8. Efficiency and Model Selection

While deep learning models offer greater accuracy, especially in challenging conditions, they require more computational resources than conventional CV methods. However, recent advances in model optimization, such as model pruning and quantization, have enabled the deployment of lightweight segmentation models on embedded systems like the Raspberry Pi.

1.1.9. Comparison of Approaches

The conventional CV approach is computationally less demanding and can perform well under controlled conditions, but it often struggles in complex environments with varying lighting or weather conditions. On the other hand, the deep learning approach excels in such scenarios due to its ability to learn from vast amounts of data and generalize across different environments. However, it requires more computational power and may not be as suitable for real-time applications on resource-constrained devices without optimization.

1.1.10. Proposed Approach: Object Detection for Lane Detection

In this approach, we propose the use of object detection (OD) models, specifically the YOLO architecture, to solve the problem of lane detection. The method leverages labeled datasets where lanes are annotated as a set of points. Instead of treating lane detection as a segmentation task or a traditional line-fitting problem, the proposed solution applies object detection to predict lane points through bounding boxes.

The process begins with taking a labeled dataset where the lanes are annotated as a series of discrete points. For each labeled point, a bounding box (BB) is created with the point at its center. The size of the bounding box is treated as a hyperparameter, allowing flexibility based on the specific characteristics of the dataset. If there are overlapping bounding boxes, another hyperparameter is used to determine the threshold for removing high-overlap boxes, thus ensuring that redundant bounding boxes are filtered out.

This approach follows the structure of the CULane dataset, which categorizes lane lines into four distinct classes: far-right, close-right, close-left, and far-left lanes. By assigning these classes to the corresponding points, the object detection model can be trained to predict not just the presence of a lane, but also its specific class. The use of classes allows the system to better differentiate between the lanes and improve the accuracy of the model.After the model is trained, the YOLO architecture is used to predict the bounding boxes for lane points. Once these predictions are made, the lane lines can be reconstructed by connecting the centers of the predicted bounding boxes. This method allows for real-time lane detection with minimal sequential steps and little to no preprocessing.

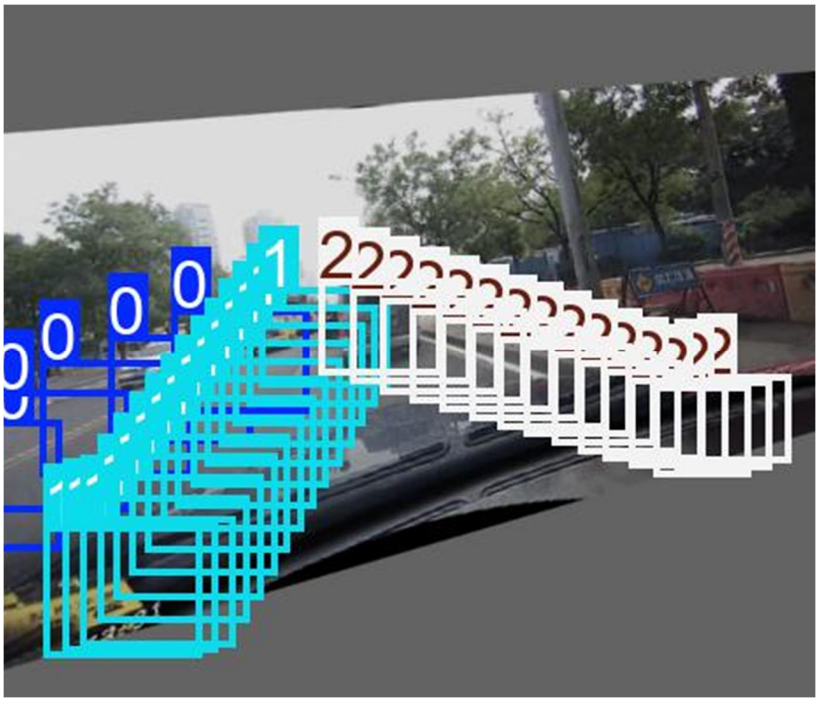


Fig. 4. Labeled Train image with Augmentation

One of the key advantages of this approach is its efficiency, especially on resource-constrained machines. The YOLO model is known for its performance on low-power devices, such as embedded systems and single-board computers, making it an excellent choice for in-vehicle ADAS systems. Furthermore, YOLO's ability to detect objects without requiring a perfectly defined object in the image allows the model to predict lane boundaries even when they are not explicitly painted on the road. This is a significant improvement over conventional and segmentation-based approaches, which often fail in situations where lanes are not clearly visible. As human drivers, we can infer lane boundaries based on contextual clues, and this approach mimics that behavior. By labeling such situations in the dataset (as in the CULane dataset), the model is able to predict lanes even in challenging environments, providing a major advantage over traditional methods.

1.1.1.9. CULane Dataset Overview

CULane is a large-scale, challenging dataset designed for academic research in traffic lane detection. The dataset was collected using cameras mounted on six different vehicles driven by various drivers in Beijing, resulting in over 55 hours of video footage and a total of 133,235 frames. The dataset is divided into 88,880 frames for the training set, 9,675 frames for validation, and 34,680 frames for testing. The test set is further categorized into a "normal" group and eight challenging conditions, such as shadowed lanes, no lane markings, or heavy traffic [16].

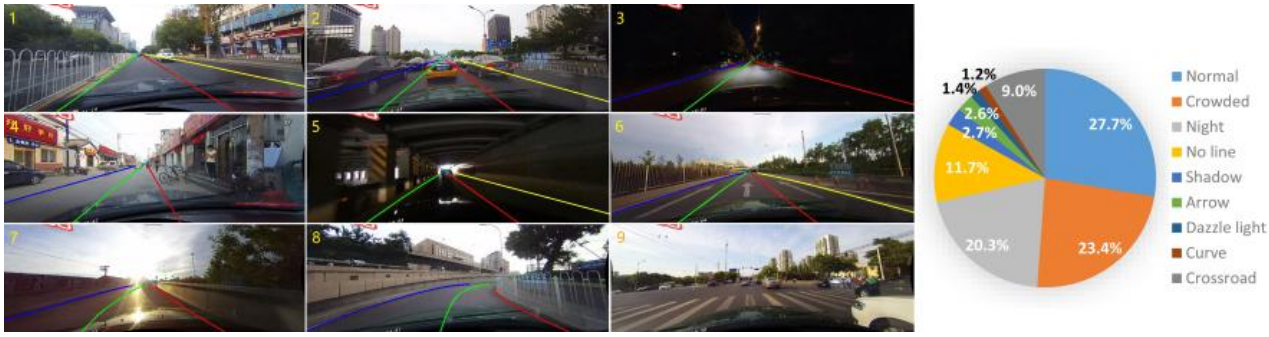


Fig. 5. CULane Dataset Overview [16]

Each frame in CULane is manually annotated with cubic splines to accurately represent the lane markings. In cases where lane markings are occluded by vehicles or are not visible, lanes are still annotated based on contextual information. This feature allows the dataset to train models to infer lane boundaries even in difficult conditions, where conventional detection methods may fail. Importantly, only the four main lane markings relevant to practical applications (far-left, close-left, close-right, far-right) are annotated, while other markings, such as those beyond barriers, are excluded. This dataset provides a rigorous and comprehensive benchmark for lane detection algorithms and is widely used in academic research to develop and evaluate advanced lane detection models.

1.1.11. Preliminary Training Results

In order to evaluate the feasibility and performance of using the YOLOv8n model for lane detection, preliminary training was conducted on the lane detection dataset. The model was trained using the Ultralytics YOLOv8.2.79 framework, with Python 3.11.8 and PyTorch 2.2.1 on an NVIDIA GeForce RTX 3080 Ti GPU. The YOLOv8n model, consisting of 168 layers and 3,006,428 parameters, achieved a processing capacity of 8.1 GFLOPs.

The training was done with these settings: an SGD optimizer was used with a learning rate of 0.01 and a momentum of 0.9. Both training and validation images were sized at 640 x 640 pixels. The training lasted for 75 epochs, finishing in 7.708 hours.

The evaluation metrics for the model included Precision (P), Recall (R), mean Average Precision at IoU threshold 0.50 (mAP50), and mean Average Precision across IoU thresholds from 0.50 to 0.95 (mAP50-95). A summary of the results is provided in Table 2.

Table 2

Preliminary training results of YOLOv8n on lane detection task

Class	Precision (P)	Recall (R)	mAP50	mAP50-95
Far Left Lane (LL)	0.562	0.843	0.786	0.568
Close Left Lane (LC)	0.763	0.831	0.872	0.664
Close Right Lane (RC)	0.659	0.818	0.823	0.588
Far Right Lane (RR)	0.485	0.487	0.511	0.295
All Lanes (Overall)	0.617	0.745	0.748	0.529

The results indicate that the YOLOv8n model performed well in detecting close-left (LC) and close-right (RC) lanes, with high precision, recall, and mAP values. However, far-right (RR) lane detection proved more challenging, with lower scores across all metrics, particularly for mAP50-95. These results validate the potential of using object detection models for lane detection tasks in real-time systems, but also suggest areas for improvement, especially in handling far-right lanes.

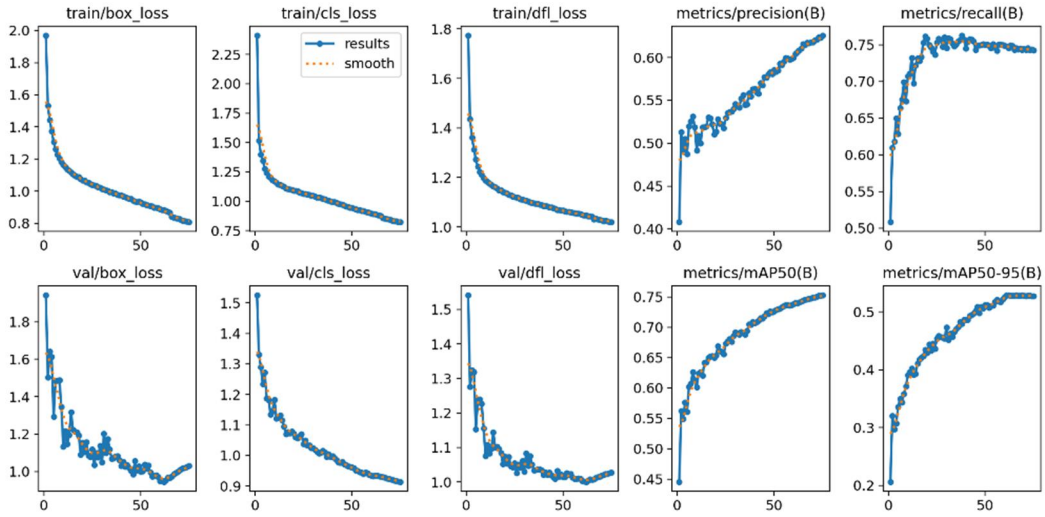


Fig. 6. Training metrics for Lane Detection

Fig. 7. Predicted Images from Validation Set

Traffic Sign Detection Systems

Traffic sign detection is a critical component in Advanced Driver Assistance Systems (ADAS) and autonomous driving, aimed at providing the driver with essential information about road regulations and hazards. However, traffic sign detection is often not utilized in real-time applications. Instead, many navigation systems rely on preloaded databases, which contain information about the traffic signs in specific geographic regions. These databases are baked into navigation maps, providing drivers with basic sign-related information. While convenient, this method has several limitations: the number of signs is limited, the data may be outdated, and it does not reflect real-time road conditions, such as newly placed or temporarily altered traffic signs.

1.1.1.10. Real-Time Traffic Sign Detection

Real-time traffic sign detection aims to address the shortcomings of relying on static databases by using an object detection model to locate and classify traffic signs as they appear in the driving environment. The process is relatively straightforward: the model detects the traffic signs in the camera feed and classifies them based on predefined categories. Object detection models such as YOLOv8 are commonly used for this purpose due to their high accuracy and fast processing capabilities.

The detection process works in two main steps. First, **sign extraction** happens, where the object detection model finds traffic signs in the image and separates them from the rest of the scene. This makes it easier to focus on just the sign itself without distractions from the background. After that comes **sign classification**, where the model identifies what type of sign it is – like whether it's a speed limit, a warning sign, or a prohibition sign.

Real-time traffic sign detection faces challenges due to the large variety of signs, with over 200 distinct types and many variants per country. As more classes are added, even optimized models like YOLOv8n become larger and more complex, increasing the computational load and slowing down inference times. This makes such models less suitable for minicomputers or embedded systems with limited resources, highlighting the trade-off between model size and real-time performance in low-resource environments.

1.1.1.11. Optimizing Traffic Sign Detection

To mitigate the performance issues, an alternative approach can be adopted by separating the processes of sign extraction and classification. Instead of detecting and classifying signs simultaneously, the system can first focus on extracting the potential traffic signs from the camera feed without classifying them. This reduces the number of categories the object detection model needs to handle at once, thus simplifying the model and improving inference speed.

Once the signs are extracted, the classification step can be deferred or handled by a lighter model that runs in parallel or asynchronously. Alternatively, a practical solution for low-resource systems may be to simply present the driver with the extracted image of the sign. In most cases, human drivers are able to recognize the meaning of the sign from its visual appearance, even without formal classification. By bypassing the classification step altogether, the system can still provide valuable information to the driver in real-time while maintaining high performance on minicomputers.

This separation of sign extraction and classification offers a balanced trade-off between accuracy and real-time performance, making it more feasible to deploy traffic sign detection systems on embedded platforms without sacrificing the timeliness of the information presented to the driver.

1.1.12. Dataset: DFG Traffic Sign Dataset

The dataset used for training the traffic sign detection system is the **DFG Traffic Sign Dataset**, which contains a wide variety of traffic signs captured on Slovenian roads. This dataset is well-suited for the task because the traffic signs in most European countries are very similar, being largely icon-based.

The dataset contains 200 traffic sign categories across around 7,000 high-resolution images. The RGB images were captured using a vehicle-mounted camera, covering both urban and rural areas across six Slovenian municipalities. Importantly, only images that contain at least one traffic sign were selected, and the dataset was curated to ensure a significant scene change between consecutive images.

The dataset is divided into 5,254 training images and 1,703 testing images, with a total of 13,239 tightly annotated traffic sign instances larger than 30 pixels. Signs with bounding boxes smaller than 30 pixels are flagged as difficult and are ignored during training. The images are anonymized by blurring faces and vehicle license plates to comply with EU GDPR regulations. The high-resolution images (1920x1080) make the dataset suitable for detecting small traffic signs in real-world driving conditions.

In addition to the base dataset, an **augmentation dataset** is provided, which introduces additional variability by distorting existing traffic signs and placing them into new scenes. This dataset includes over 30,000 augmented traffic sign instances across 8,775 additional images. The augmentation techniques include perspective changes, scaling, and variations in brightness and contrast. By incorporating these augmented images, the training process benefits from increased diversity and robustness.

This dataset allows the model to generalize to various conditions and ensures that the system can detect traffic signs even under challenging situations, such as poor lighting or occlusions.

1.1.13. Training of YOLOv8n for Traffic Sign Detection

To implement the proposed traffic sign detection system, the **YOLOv8n model** was trained using the **DFG Traffic Sign Dataset**. Instead of classifying individual signs, the focus was on detecting one class: **traffic sign**, simplifying the task. This approach allows for handling classification separately or simply presenting the extracted sign image to the driver.

The training spanned **50 epochs** and was completed in **5.5 hours**. The model, consisting of **168 layers** and **3 million parameters**, was trained using the **AdamW optimizer** with an automatically determined learning rate (**lr=0.002**) and momentum (**0.9**). The input image size was set to **640x640 pixels**, ensuring high-resolution inputs to capture smaller traffic signs.

The model achieved excellent performance: **Precision (P): 0.977**, **Recall (R): 0.956**, **mAP50: 0.983**, and **mAP50-95: 0.854**. Inference speed was **1.8 ms per image**, making it highly suitable for real-time applications, as it can quickly detect signs in dynamic driving environments. The slight delay in delivering sign information to the driver is not critical, as long as signs are captured reliably.

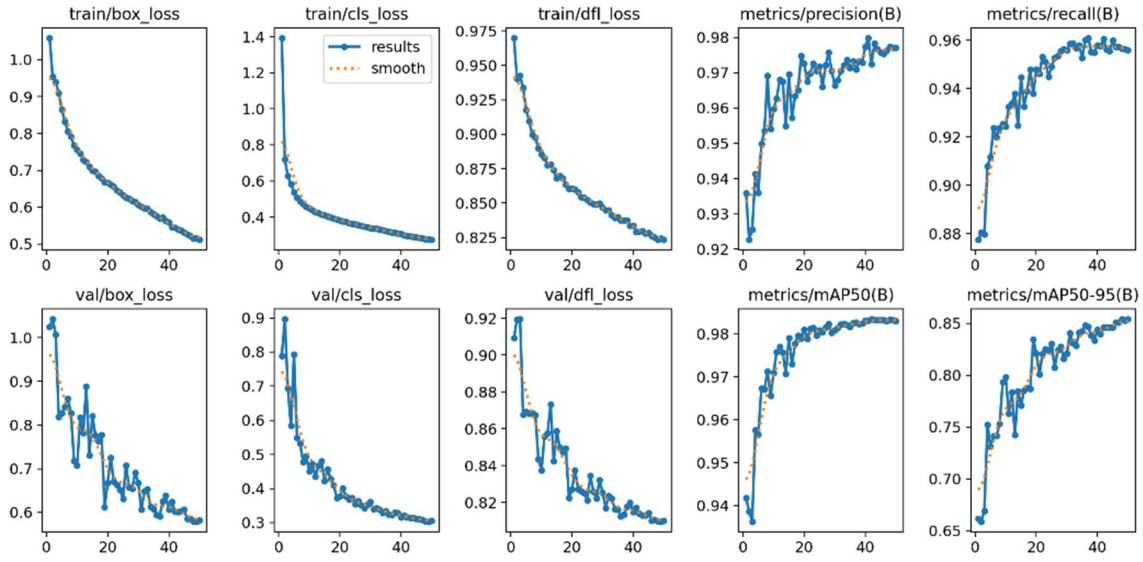


Fig. 8. Training metrics for Traffic Sign Detection

Fig. 9. Predicted Images from Validation Set

This approach offers a balance between performance and efficiency, making it suitable for deployment on embedded systems with limited computational resources. By simplifying the task to sign detection and deferring classification, the system ensures that it does not miss critical information on the road due to low frame rates, while still providing timely warnings to the driver.

Pothole Detection System

Pothole detection is not commonly found in existing ADAS solutions, especially in the context of real-time visual systems. While there are examples of pothole detection used for road monitoring or maintenance purposes, real-time pothole detection aimed at driver assistance remains an open field for innovation. Our proposition is to use an object detection model, specifically YOLOv8n, trained to detect potholes in real-time based on dashcam footage. Given YOLOv8n's ability to handle object detection efficiently on low-resource devices, it is well-suited for this task, where both detection speed and timely delivery of the information to the driver are crucial.

The task is straightforward: detect potholes in dashcam footage in real time to warn drivers about potential road hazards. However, the challenge lies in finding suitable datasets, as there are no large public datasets that specifically focus on potholes in the context of dashcam footage. To address this, we combined an existing pothole dataset with our own labeled dashcam images to create a more robust dataset. The following sections describe the dataset, training process, and results.

1.1.14. Dataset: Pothole Detection Data

For this task, we started with a small public dataset from Kaggle, known as the **Potholes Dataset**, which contains 665 images with pothole annotations in PASCAL VOC format. The dataset was originally created for road maintenance purposes and includes images from a variety of sources, though they are not specifically dashcam footage. Given that these images do not fully represent real-time driving conditions, we anticipated that the results from using this dataset alone would not be satisfactory for our real-time detection needs.

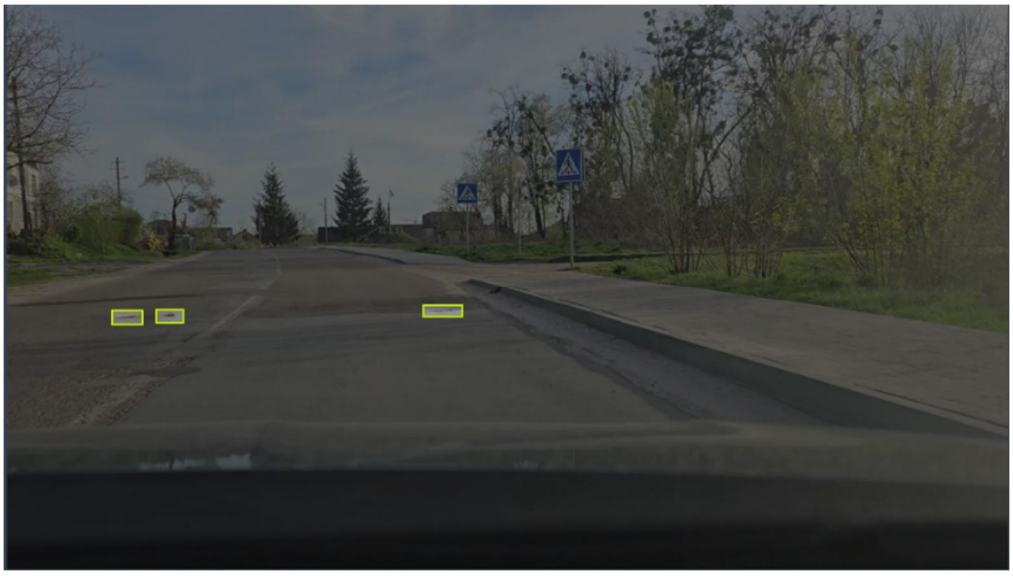


Fig. 10. Labelled Frame with Potholes in Bruhovychi

To improve the dataset and make it more applicable to real-time ADAS, we expanded it by adding over 300 images sourced from public YouTube videos and footage captured in the cities of Bruhovychi and Lviv using dashcams. The newly collected images were labeled using the Roboflow framework, where all visible potholes were annotated. The final dataset contains over 900 images with labeled potholes, providing a diverse and representative set of training data for the detection task.

1.1.15. Training of YOLOv8n for Pothole Detection

The YOLOv8n model was trained on the enhanced pothole dataset to detect potholes as a single class. The training process utilized extensive data augmentation techniques to increase the variety of training samples and improve the model's robustness. Augmentations included geometric transformations, brightness and contrast adjustments, and random cropping, ensuring that the model would generalize well to various road and lighting conditions.

It was trained on a YOLOv8n-based training setup, with 168 layers and a total of 3,005,843 parameters. The selected optimizer was AdamW, with a learning rate of 0.002 and momentum of 0.9. Image sizes for both training and validation were set to 640x640 pixels. Training was conducted for a total of 100 epochs.

The augmented dataset allowed for better generalization, and the YOLOv8n model was able to learn the characteristics of potholes under various conditions, from urban to rural environments. The goal was to create a model capable of detecting potholes in real-time and delivering this information to the driver in a timely manner.

1.1.16. Results and Performance

The **YOLOv8n model** was evaluated on a validation set of **181 images** containing **538 pothole instances**. The results were: **Precision (P): 0.782**, **Recall (R): 0.641**, **mAP50: 0.719**, and **mAP50-95: 0.385**. While the model performed well with a precision of **0.782** and **mAP50** of **0.719**, the lower **mAP50-95** score (**0.385**) indicates challenges with higher IoU thresholds, likely due to the variability in dashcam footage where potholes are less distinct.

However, the model's **inference speed of 1.2 ms per image** makes it highly suitable for real-time applications, providing a solid foundation for pothole detection in ADAS. The combination of public and custom-labeled dashcam data improved the model's generalization to real-world driving conditions, though further improvements in dataset size and variety, as well as advanced post-processing techniques, could enhance performance, particularly at higher IoU thresholds.

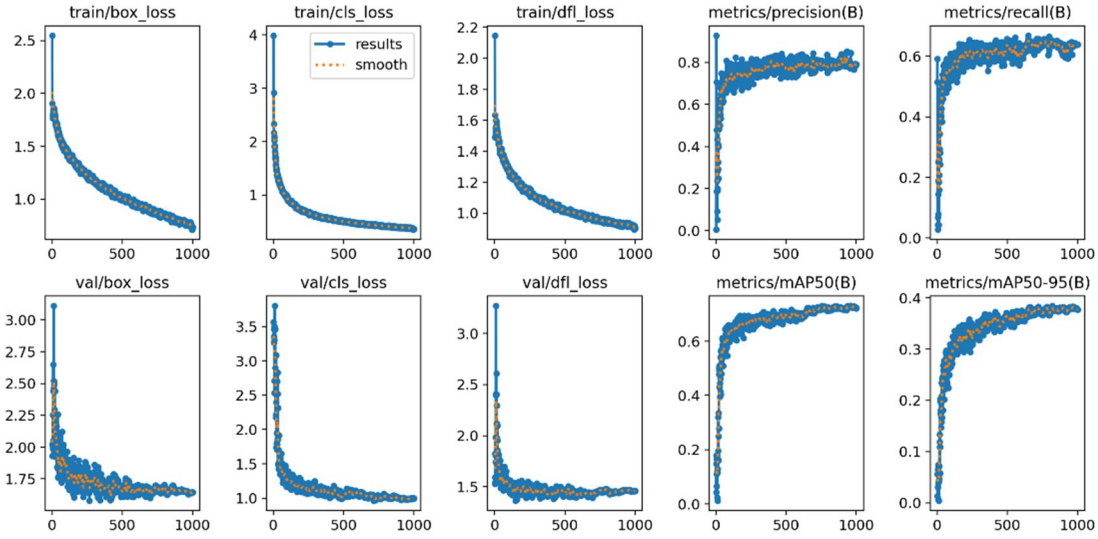


Fig. 11. Training metrics for Pothole Detection

Fig. 12. Predicted Images from Validation Set

1.1.1.12. Delivering the Information to the Driver

One of the main challenges in deploying pothole detection systems on minicomputers is processing speed. While the YOLOv8n model is optimized for real-time detection, it still takes some time to process each image, particularly on resource-constrained hardware. The delivery of pothole information to the driver must be quick and effective to ensure safety. Displaying warnings on a screen, such as a dashboard or heads-up display (HUD), is not the ideal solution. Such visual warnings may distract the driver, drawing their attention away from the road during a critical moment when full concentration is required.

Our proposed solution is to replace screen-based warnings with a more intuitive and non-distracting system: an LED strip mounted under the windshield, paired with auditory warnings. This system works by lighting up the LED strip in the direction of the detected pothole, without requiring the driver to look at a screen.

The concept is as follows: if the system detects a pothole on the left side of the road, the corresponding left section of the LED strip will light up in a red color, accompanied by a beeping sound. The driver will instinctively understand that the danger is located on the left side, allowing them to focus on the road in that direction. The LED strip will highlight the approximate location of the pothole relative to the center of the ego vehicle, helping the driver react quickly without unnecessary distractions.

This approach ensures that the driver's attention remains on the road while still receiving crucial information about potential hazards. By directing the driver's attention to the correct area of the road, this solution provides a more seamless and effective warning system, enhancing both safety and reaction times.

Next Steps – Combined System

In this section, we propose a combined system that integrates each individual component (collision warning, traffic sign detection, lane detection, and pothole detection) into a unified driver assistance system. The objective of this system is to perform real-time image analysis and deliver actionable information to the driver using a resource-constrained device, ideally a minicomputer such as the NVIDIA Jetson Nano or Xavier. If optimized sufficiently, the system could also be implemented on a Raspberry Pi, which would be a significant breakthrough in terms of performance on limited hardware.

1.1.17. System Components and Workflow

The system comprises several **key components** working together. **Image Capturing** utilizes a front-facing camera mounted on the dashboard to continuously capture video footage. This footage undergoes **Image Preprocessing**, where steps like undistortion, cropping, and filtering ensure data quality. The central element is the **Universal YOLO Model**, a deep learning model trained to detect vehicles, pedestrians, traffic signs, potholes, and lanes, optimized for real-time inference on limited hardware. To enhance FPS performance, **Parallel YOLO Model Instances** can run simultaneously, processing multiple frames and minimizing delays.

An **Object Tracker**, such as SORT or Kalman filter, maintains awareness of dynamic objects between frames, while the **Kalman Filter** predicts object speed and position, crucial for accurate collision warnings. The **Perspective Transformer** converts object data into a bird's-eye view for better spatial understanding. Other tasks like distance estimation and lane curvature detection are handled by **Additional Processing Modules** operating in parallel. The **Final Frame Composer** integrates all outputs into a coherent frame, while the **Warning Processor** evaluates the situation and generates alerts when necessary. Finally, the **Output Display** delivers information to the driver via LED indicators, dashboard displays, or audio warnings, designed to minimize distraction.

1.1.18. Module-Based Parallel Processing Approach

To achieve real-time performance, the system will adopt a modular architecture, where each component or processing step is wrapped in an independent module that operates in parallel. This architecture will reduce latency by eliminating the need for sequential processing wherever possible. Each module will store its last processed result and provide the information to other modules when required, allowing the system to remain responsive even under high loads.

Multithreading will be employed extensively to ensure that the system's performance scales with the available hardware resources. For example, the image preprocessing module can run concurrently with the object detection model, while the warning processor can operate in parallel to the object tracker and perspective transformer. If the system detects that it has enough available resources, multiple instances of the same YOLO model can be deployed, each working on different frames to further increase the FPS rate.

The result is a highly modular and scalable system that is capable of adapting to the hardware it is deployed on, ensuring maximum efficiency and real-time performance.

1.1.19. Challenges and Optimizations

While this approach offers a promising solution, it also comes with significant challenges, especially in terms of optimization for resource-constrained hardware. For example, running multiple instances of a YOLOv8n model on a Raspberry Pi would require aggressive optimization techniques, such as model pruning, quantization, and multithreading optimization. Further optimizations will need to focus on reducing the latency of each module, particularly the object detection and tracking modules, which are the most computationally demanding. Nonetheless, if successfully implemented, this approach will enable real-time driver assistance systems to run on inexpensive hardware platforms, offering a massive potential impact on the accessibility and availability of such systems in consumer vehicles.

1.1.20. System Architecture Representation

The following diagram (Figure 13) represents the preliminary structure of the combined system, illustrating how each module interacts and processes data in parallel.

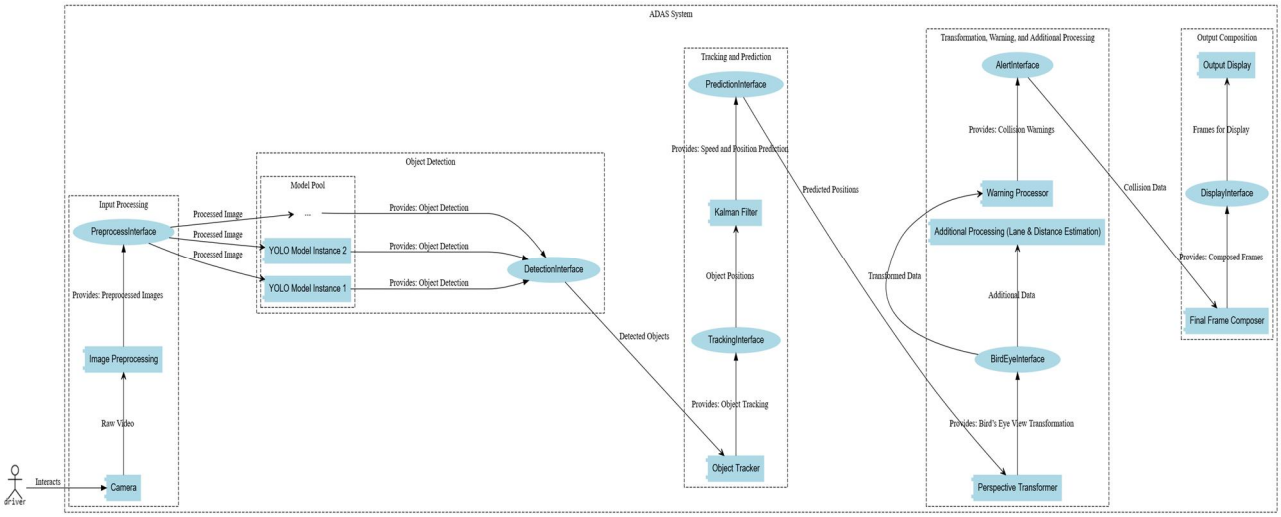


Fig. 13. Preliminary structure of the combined system

Conclusions

In this research, we have proposed an advanced ADAS system integrating multiple real-time components, including collision warning, lane detection, traffic sign recognition, and pothole detection,

using cutting-edge machine learning and computer vision technologies. The system architecture is designed to be modular, leveraging parallel processing to ensure real-time performance even on resource-constrained devices such as Raspberry Pi or NVIDIA Jetson Nano.

We presented comprehensive approaches for each system, detailing the benefits and challenges of both conventional and deep learning methodologies. The lane detection system showed significant improvements with the application of object detection models like YOLO, allowing for real-time lane boundary detection with minimal preprocessing. Similarly, the proposed traffic sign detection method provides a flexible solution, balancing detection accuracy and speed by separating the sign detection and classification processes.

A novel approach to pothole detection using YOLOv8 was also explored, demonstrating the potential of real-time road surface monitoring as part of an integrated driver assistance system. The implementation of an innovative warning delivery system, utilizing LED strips to provide non-distracting alerts to drivers, showcases the importance of designing user-friendly interfaces in ADAS solutions.

While the results of our preliminary experiments are promising, particularly in terms of detection speed and accuracy, challenges remain in optimizing the models for embedded systems. Extensive model pruning, quantization, and multithreading will be required to fully realize the system's potential on low-resource platforms.

Future work will focus on further optimizing the system architecture, enhancing detection accuracy in adverse conditions, and expanding the dataset for pothole detection to ensure broader generalization. Successful implementation of this integrated ADAS solution could significantly improve road safety, making advanced driver assistance technology more accessible to a wide range of vehicle owners.

REFERENCES

1. Kaur, G., Kumar, D. (2015). Lane detection techniques: A review. *International Journal of Computer Applications, 112*(10), 1–8.

2. Saha, A., Roy, D. D., Alam, T., Deb, K. (2012). Automated road lane detection for intelligent vehicles. *Global Journal of Computer Science and Technology, 12*(6), 1–6.

3. Rachel, M. J. S., Kalaiselvi, S., Salini, R. (2020). Lane detection using neural networks. *International Research Journal of Engineering and Technology, 7*(3), 3578–3582.

4. Murthy, J. S., Siddesh, G. M., Lai, W.-C., Parameshachari, B. D., Patil, S. N., Hemalatha, K. L. (2022). ObjectDetect: A real-time object detection framework for advanced driver assistance systems using YOLOv5. *Wireless Communications and Mobile Computing*, 1–10. https://doi.org/10.1155/2022/9444360

5. Buza, E., Omanovic, S., & Huseinovic, A. (n.d.). Pothole detection with image processing and spectral clustering. Recent Advances in Computer Science and Networking, 48-53.

6. Joe, H., Blessingh, J., Cherian, J. (2020). An intelligent pothole detection system using deep learning. *International Research Journal of Engineering and Technology, 7*(2), 1591–1594.

7. Jumaa, B. A., Abdulhassan, A. M., Abdulhassan, A. M. (2019). Advanced driver assistance system (ADAS): A review of systems and technologies. *International Journal of Advanced Research in Computer Engineering & Technology, 8*(6), 231–234.

8. Golgire, V. (2021). Traffic sign recognition using machine learning: A review. *International Journal of Engineering Research & Technology (IJERT), 10*(5), 872–876.

9. Tyagi, H., Saroj, V. K., Shahzad, M., Agarwal, A. (2023). Evolution of YOLO: Exploring the advancements in YOLOv8 for real-time wildlife detection. *Journal of Computer Vision in Wildlife Monitoring, 1*(2), 1–10.

10. Tabernik, D., Skočaj, D. (2019). Deep learning for large-scale traffic-sign detection and recognition. IEEE *Transactions on Intelligent Transportation Systems*, 1524–9050. https://doi.org/10.1109/TITS.2019.2913588

11. Make ML. (n.d.). Potholes dataset. https://makeml.app/datasets/potholes

12. Jocher, G., Chaurasia, A., Qiu, J. (2023). Ultralytics YOLOv8 (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

13. Rezaei, M., Terauchi, M., Klette, R. (2015). Robust vehicle detection and distance estimation under challenging lighting conditions. *IEEE Transactions on Intelligent Transportation Systems.* https://doi.org/10.1109/TITS.2015.2421482

14. Budagam, D., Kumar, A., Ghosh, S., Shrivastav, A., Imanbayev, A., Akhmetov, I., Kaplun, D., Antonov, S., Rychenkov, A., Cyganov, G., Sinitca, A. (2024). Instance segmentation and teeth classification in panoramic X-rays. *arXiv*. https://doi.org/10.48550/arXiv.2406.03747

15. Haque, M. R., Islam, M. M., Alam, K. S., Iqbal, H., Shaik, M. E. (2019). A computer vision-based lane detection approach. *Khulna University of Engineering & Technology*. Received: 25 October 2018; Accepted: 17 January 2019; Published: 08 March 2019.

16. Pan, X., Shi, J., Luo, P., Wang, X., Tang, X. (2018, February). Spatial as deep: Spatial CNN for traffic scene understanding. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

17. Souweidane, N., Smith, B. (2023). State of ADAS, Automation, and Connectivity. Center for Automotive Research, Ann Arbor, MI.

18. Tomasch, E., Smit, S. (2023). Naturalistic driving study on the impact of an aftermarket blind spot monitoring system on driver behavior of heavy goods vehicles and buses on reducing conflicts with pedestrians and cyclists. *Accident Analysis and Prevention, 192,* 107242. https://doi.org/10.1016/j.aap.2023.107242

ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ДОПОМОГИ ВОДІЮ НА ОСНОВІ КОМП'ЮТЕРНОГО ЗОРУ ТА ГЛИБИННОГО НАВЧАННЯ

Артем Телюк[1], Андрій Василюк[2], Андрій Худий[3]

Національний університет “Львівська політехніка”,
кафедра інформаційних систем та мереж, Львів, Україна
[1] E-mail: artem.teliuk.mnsam.2023@lpnu.ua, ORCID 0009-0000-1180-8492
[2] E-mail: andrii.s.vasyliuk@lpnu.ua, ORCID 0000-0002-3666-7232
[3] E-mail: andrii.m.khudyi@lpnu.ua, ORCID 0000-0003-2029-7270

У статті представлено інтегровану інтелектуальну систему допомоги водію (ADAS), яка об’єднує кілька ключових функціональних модулів, як-от: система попередження про зіткнення, виявлення смуг руху, розпізнавання дорожніх знаків та виявлення ям на дорогах, що реалізовані за допомогою сучасних моделей глибинного навчання, зокрема YOLOv8n. Система оптимізована для роботи на пристроях Raspberry Pi або NVIDIA Jetson Nano із обмеженими обчислювальними ресурсами із застосуванням модульної архітектури та паралельного опрацювання даних для забезпечення швидкодії в режимі реального часу. В межах цього дослідження проведено огляд наявних рішень в ADAS та запропоновано нові підходи, що значно підвищують ефективність таких систем. Ключовими інноваціями є ефективний підхід до виявлення смуг руху на основі моделей виявлення об'єктів, виявлення дорожніх знаків у реальному часі з гнучким процесом екстракції та класифікації, а також нова система виявлення ям, оптимізована для відеозаписів із відеореєстратора. Крім того, запропонована система оповіщення водія за допомогою світлодіодної смуги дає змогу інтуїтивно привертати увагу до потенційних небезпек. Попередні результати підтверджують задовільну точність виявлення у всіх компонентах, проте для успішного впровадження на пристроях із низькими ресурсами потрібна додаткова оптимізація.

Ключові слова: інтелектуальна система допомоги водію, виявлення об’єктів, глибинне навчання, реальний час, YOLOv8n, виявлення смуг руху, дорожні знаки, ями на дорогах.