*Journal of Electrical and Intelligent Systems (JEIS), Vol. 1, Issue 2*
*September 2026*
*ISSN: 3163-7663*

# Driver Fatigue Detection Using YOLOv5 with Genetic-Algorithm-Based Hyperparameter Optimization and Fine-Tuning Strategies

Hendri Darmawan[1*], Zulfa Muflihah[2], Giran Arisdi[3]
[1,3] *Department of Electrical Engineering, Politeknik Elektronika Negeri Surabaya (PENS), Indonesia*
[2] *Department of Engineering Physics, Institut Teknologi Sepuluh Nopember (ITS), Indonesia*
[*] *Corresponding author: Hendri Darmawan; email: hendridarmawan@pens.ac.id*

Manuscript received 7 April 2026, revised 10 June 2026, accepted 24 June 2026
DOI: https://doi.org/10.68129/jeis.v1i2.7

**Abstract** – *Driver fatigue is a major contributor to traffic accidents, injuries, and economic losses. This study proposes a computer-vision-based driver fatigue detection system using YOLOv5s to identify three facial states: normal, eye closed, and yawning. The model is fine-tuned from pre-trained COCO weights and evaluated under multiple training schemes, including full fine-tuning, partial layer freezing, and hyperparameter evolution based on a genetic algorithm. The dataset consists of 5,706 training images and 2,390 testing images compiled from self-recorded and public sources under varied lighting conditions. Experimental results show that the best model without evolution achieves an mAP of 0.887, while the best evolved model reaches an mAP of 0.888 and runs at 30 FPS on a CPU-based device. These findings indicate that the proposed model is sufficiently accurate, lightweight, and practical for early-warning deployment on low-resource edge devices in advanced driver assistance systems.*

***Keywords***: *Drowsiness detection, Fine-tuning, Genetic algorithm, Hyperparameter optimization, YOLOv5*

## I. Introduction

Indonesia is facing a worrying rise in road accidents. Data Indonesia reports that road accidents in Indonesia climbed to 103,645 in 2021, a 3.62% rise from 100,028 the year before. Sadly, 25,266 people lost their lives in these accidents in 2021, a 7.38% increase from 23,529 deaths in the prior year. The financial impact was also substantial, with losses totaling IDR 246 billion, up 24.24% from IDR 198 billion in 2020. A recent AAA Foundation for Traffic Safety study indicates that driver fatigue might contribute to as many as 9.5% of all crashes, a figure notably larger than the official numbers from the National Highway Traffic Safety Administration (NHTSA). The investigation revealed that around 11% of more serious crashes that were reported by police had drowsy driving as a factor [1].

Microsleep episodes are hard to spot because they're so brief, but they're especially risky when behind the wheel. To combat this, scientists have created systems that identify when someone is getting drowsy, employing different techniques like psychological and physiological measurements [2]. Since psychological methods are difficult to put in cars and are bothersome for drivers, focusing on physiological signs is more popular. Specifically, using computer vision to spot drowsy drivers has been a big area of study. This research introduces a drowsiness detection system using the YOLO algorithm, fine-tuned with optimized hyperparameters. The aim is for this system to act as an early warning, helping to prevent accidents caused by tired drivers.

This study focuses on identifying drowsy drivers using real-time video footage by recognizing facial cues like closed eyes or yawns. The approach uses object detection, a deep learning technique. There are numerous deep learning approaches for object detection, such as Faster R-CNN, Fast R-CNN, R-CNN, SSD, YOLO, among others. Some methods employ a two-step process: first, locating the object within a region (bounding box) and then classifying that region. This is recognized as a two-stage detector, such as Faster R-CNN [3]. In contrast, YOLO, a single-stage detector, simultaneously identifies and classifies objects within a single network. The system segments the image into sections and estimates bounding boxes and likelihoods for every area [4]. The likelihoods of the bounding boxes are adjusted to categorize them as objects. This enables the network to recognize and pinpoint several objects within the image, even if they are partially obscured or of varying sizes. The predicted bounding boxes and probabilities can be further refined through post-processing to boost the precision of object recognition. This paper proposes a system that automatically detects fatigued drivers, aiming to prevent

accidents caused by drowsiness. The paper is segmented up into sections. The first section discusses the problem of drowsy driving and the need for a detection system. The second section reviews previous research on drowsiness detection, including methods like facial landmark detection, template matching algorithms, and machine learning classification models. The third section details the experiments conducted using the YOLO algorithm to test the proposed system. Finally, the fourth section presents the performance evaluation results of the YOLO algorithm in various scenarios and concludes on the effectiveness of the drowsiness detection system.

TABLE I
COMPARISON OF RELATED DRIVER FATIGUE DETECTION STUDIES

| Study | Method | Edge Device | Limitation |
|---|---|---|---|
| Fikriyah et al. [5] | EAR | Yes | Threshold manual |
| Meshram et al. [6] | SURF | Yes | Feature engineering |
| Phan et al. [7] | SSD + MobileNet | Limited | Two-stage |
| Rahman et al. [2] | CNN | Yes | Limited fatigue states |
| Proposed | YOLOv5 + GA | Yes | Class Imbalance |

Many studies use various approaches to identify driver fatigue. This research leverages both independent and public data to detect drowsy drivers using the YOLO algorithm. YOLO is chosen for its speed and performance compared to other object detection methods. Our model is trained on a dataset of facial representations labeled as either drowsy or alert, comprising 5706 training images and 2390 testing images. These images are categorized into three classes: normal, eyes closed, and yawning. Existing research [5] employs a pre-trained dlib model for sleepy eye detection. This model maps 68 facial coordinates using a regression tree method. The eye aspect ratio formula analyzes these coordinates to check whether the eyes are blinking or closed, indicating drowsiness.

This research, however, did not involve model updates. Additionally, a manual threshold is required to distinguish normal eye ratios, which can vary between individuals. Other researchers [6] used a SURF feature extraction algorithm. Eye positions and edges are identified through frame differentiation and template matching. The SURF algorithm then classifies eye states as either alert or drowsy. This method achieved an average accuracy of 96% when tested on five videos with varying lighting conditions. Researcher [7] proposed using computer vision and deep learning to identify drowsy drivers. They used a SSD network with ResNet-10 to detect faces, which were then classified as drowsy or not using either MobileNet-V2 or Resnet-50V2 models. Despite achieving a high accuracy of 97%, this method requires two stages of processing, making it resource-intensive. Implementing this approach in real-time on edge devices is difficult due to their limited processing power. Using two models results in a lower frame rate compared to a single detection model.

As summarized in Table I, previous studies have demonstrated promising performance for driver fatigue detection. However, most approaches either rely on manually defined thresholds, employ multi-stage architectures, or provide limited discussion regarding deployment efficiency on resource-constrained edge devices. Furthermore, studies investigating the impact of fine-tuning strategies and genetic-algorithm-based hyperparameter optimization for driver fatigue detection remain limited. Therefore, this work evaluates multiple fine-tuning schemes and GA-based hyperparameter evolution to improve detection performance while maintaining real-time inference capability on CPU-based hardware.

Based on the identified research gaps, this study proposes a driver fatigue detection system based on YOLOv5 for recognizing three fatigue-related facial states, namely normal, eye-closed, and yawning. The study investigates the effectiveness of several fine-tuning strategies, including full training and partial layer freezing, to determine the most suitable transfer learning approach for the proposed task. In addition, a genetic algorithm (GA)-based hyperparameter evolution scheme is employed to optimize the model configuration and evaluate its impact on detection performance. Unlike many previous studies that primarily focus on detection accuracy, this work also emphasizes deployment feasibility on resource-constrained edge devices by evaluating the model's real-time inference performance on a CPU-based platform. The proposed approach aims to achieve a practical balance between detection accuracy, computational efficiency, and real-world applicability for Advanced Driver Assistance Systems (ADAS).

## II. Research Method

Transfer learning is an essential technique in Convolutional Neural Networks (CNN) models. This techniMeshque uses a pre-trained model to solve similar problems in different domains. In some cases, it may be necessary to adjust the pretrained model's weights to solve a new problem in a different domain [8]. This process is called fine-tuning. Fine-tuning can improve the accuracy of a model while requiring fewer epochs than training the model from scratch [9]. This research employed fine-tuning techniques to improve how well the YOLOv5 model could detect drowsiness by analyzing facial characteristics. We started with the pre-trained model’s weights from the COCO dataset and fine-tuned the model for detecting three classes: normal, closed eyes, and yawning mouth. For fine-tuning, we used the small-weight version of the YOLOv5 model.

We explored different ways to fine-tune the YOLO model including full training, partial training, and adaptive learning rate changes. We also compared those

conventional hyperparameter optimization methods in this experiment with the Genetic Algorithm (GA) or evolutionary approach [10]. We adjusted the genetic mutation operator with a variance of 0.04 and an 80% probability, adding some randomness to the evolutionary process every five epochs. The mutation creates new offspring based on the best combination of parents from all previous generations [11]. This allows the GA to find the optimal combination of hyperparameters for our drowsiness detection task. We tested the GA with 150, 300, and 450 generations. After obtaining the highest fitness offspring from the evolutionary process, we used the resulting hyperparameters to train the model and analyze its performance to find the best model.

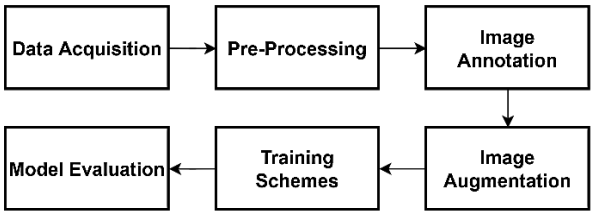



Fig 1. Experiment with the design of the proposed method

Fig. 1 demonstates the block diagram of the research process we followed. first, we conducted a literature review to identify deep learning models that can provide fast computing and accurate prediction. Then, we performed data acquisition to build the model. We also utilized pre-processing methods to refine the image data, aiming to sharpen and accentuate key elements within the images [12]. Next, we used several methods for expanding the diversity of training data through image augmentation [13]. We then fine-tuned the YOLOv5s model using various scenarios as described in the next subsection. Finally, we evaluated the model using real-time test and inference data to ensure its effectiveness. In a practical deployment scenario, the proposed system can be integrated into an Advanced Driver Assistance System (ADAS) using a dashboard-mounted camera positioned in front of the driver. The camera continuously captures facial images and transmits them to an onboard computing unit running the YOLOv5 model. When fatigue-related states such as prolonged eye closure or yawning are detected, the system generates an audible warning to alert the driver. Due to its lightweight architecture and real-time performance on CPU-based hardware, the proposed approach can be deployed on resource-constrained edge devices without requiring dedicated GPU acceleration.

### *A. YOLO*

YOLO, short for You Only Look Once, is a widely favored technique for detecting objects in images [14]. Researchers at Facebook AI Research have found that the YOLO architecture operates swiftly while maintaining a high mean average precision (mAP) value in contrast to other live detectors. Girshick Ross, Divvala Santosh, Farhadi Ali, Redmon Joseph created YOLO, which they first presented in 2016 at the CVPR conference. It was popular, even winning the OpenCV People's Choice Awards there.

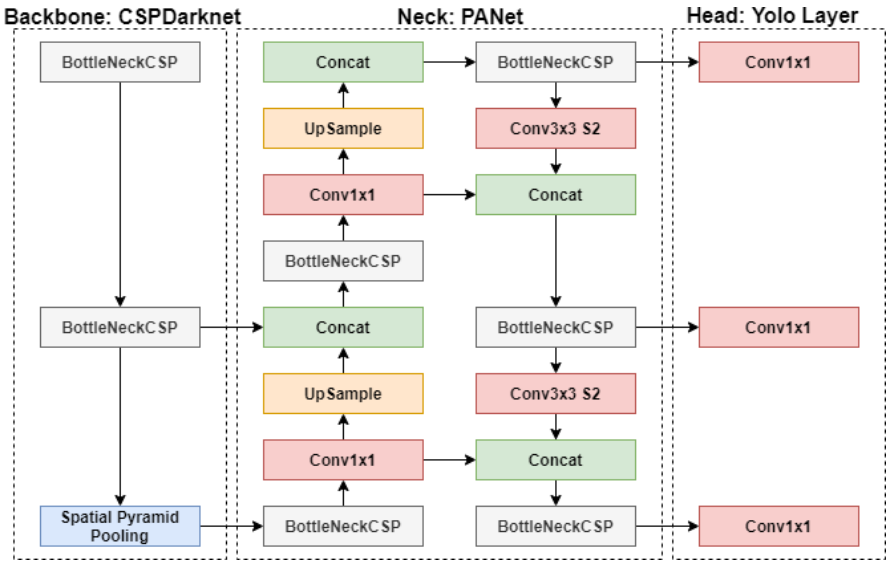


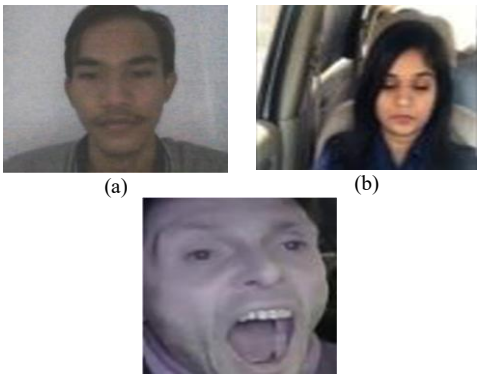

Fig 2. YOLOv5 architecture

YOLOv5 works by first examining an image to identify key characteristics. It then uses these features in a prediction process, which involves drawing boxes around detected objects and determining what those objects are. As illustrated in Fig.2, the YOLOv5 network is built from three key components: the backbone, neck, and head. The backbone is essentially a convolutional network that extracts and merges image features at varying scales. The neck acts as a connector, integrating these features and feeding them forward for prediction. Finally, the head utilizes the recognized elements from the neck to outline the bounding boxes and the object classes within those boxes. YOLOv5 uses a backbone: CSPDarknet, neck: PANet, and head: YOLO Layer [15] as seen in Fig. 2. YOLO computes its overall loss by combining three

different loss functions: one for GIoU, one for objectness, and one for classification. The objective of this loss function is to maximize the mAP.

### B. Data Acquisition

In machine learning, data is an essential component used for training models. The more diverse and relevant to the data, the more accurate and reliable the model will be. Fig. 3 portrays an example of the dataset used to build the model.



(a) (b) (c)

Fig 3. Sample image: (a) Normal (b) Eye closed (c) Normal and yawning

We collected our own data using a webcam camera under various lighting conditions. We also obtained data from various open data sources such as Kaggle and Roboflow. Kaggle is a popular platform for data scientists to find and share datasets, and Roboflow provides tools to organize and label the information used for building computer vision applications. However, we did not use the entire dataset for training. We discovered that some images had similar characteristics, and using duplicates would only use up memory without significantly improving model performance. Therefore, we selected only unique images for training to ensure the best use of the available data. This process is crucial for creating a well-performing machine learning model that can accurately handle data it hasn't seen before.

### C. Pre-processing and Image Annotation

The data we obtained from Kaggle had a low resolution of 120x120 pixels, whereas the proposed research used a resolution of 320x320 pixels for the YOLO model input. Therefore, we used the Topaz Gigapixel AI to upsample the low-resolution images and increase their resolution. Gigapixel AI's system uses a neural network, studying countless pairs of photos to understand the common ways details degrade when images are made larger. The system figures out how to fill in the missing parts of the fresh image, making it bigger, sharpening it, and producing realistic-looking elements. We annotated the images by assigning bounding boxes and category classes to the objects in the images that we want to detect [16]. Fig. 4 shows some examples of annotated images. Table I shows the distribution of data in each class.




Fig 4. Sample of annotated image

TABLE II
DATASET LABELS DISTRIBUTION

| Class | Frequency | Type |
|---|---|---|
| Normal | 2653 | Training Set |
| Eye Closed | 2347 | |
| Yawning | 706 | |
| Normal | 1295 | Testing Set |
| Eye Closed | 675 | |
| Yawning | 420 | |

From Table II, it can be seen that there is an imbalanced class problem, where the "Yawning" class has a significantly different number of labels than the "Normal" and "Eye Closed" classes. We fine-tuned the YOLOv5 model without any special handling for the minority class. This was to test if hyperparameters alone could solve the imbalanced class problem. Fig. 5 shows the annotation heatmap for each class category.

Annotation heatmap shows the density of objects in different regions of the image, highlighting areas where there are more objects and dimming areas where there are fewer objects. It is used to check the quality of annotations and ensure that objects are properly labeled and localized in the images.



(a) (b) (c)

Fig 5. Annotation heatmap: (a) Normal (b) Eye closed (c) Yawning

To make the image look better, we also enhanced the contrast by using a technique called contrast stretching. Contrast stretching refers to adjusting the intensity intervals of an image and stretching them to new intervals (generally, the target intervals are the lowest and highest values that the image can take, such as 0 and 255) [17]. It helps improve the image's contrast and make it easier for the YOLO model to detect the objects in the image. Contrast stretching is a technique that expands the histogram of intensity levels in an image to cover the entire range of intensities in the input range. Before applying contrast stretching, the first step is determining the lower bound (a) and upper bound (b) values of the image pixels. We then scan all image pixels to find the lowest pixel value (c) and the highest pixel (d) in the image. Each pixel is then rescaled using (1) [18]. Eq. (1) stretches the image's contrast by mapping the pixels' intensity values to the new range [a, b]. This helps improve the image's contrast and make it easier for the YOLO model to detect the objects in the image. If there is a scaling result value less than zero, it is considered zero. If there is a pixel value greater than 255, it is considered 255.

$$P_{out} = \left(P_{in} - c\right)\frac{b-a}{d-c} + a \quad (1)$$

*D. Image Augmentation*

In YOLOv5, an object detection model, we have an online data augmentation feature that allows for various transformations to be applied to images during training. The changes offered include scaling, color map adjustments, and mosaic augmentation. Mosaic augmentation is a particular method that merges four different images into a single image, arranged in a four-part grid with randomly sized sections. This technique is helpful for enhancing the identification of smaller items within images, as these tiny objects might not be recognized as reliably as larger ones. In this research, we only performed online augmentation due to limited memory resources available for training the model. Table III in our study shows the online augmentation parameters used before hyperparameter evolution. By using online augmentation, we increased the variation of the training data and improved the performance of the YOLOv5 model. This resulted in a better ability to detect objects accurately, improving the model's overall performance.

TABLE III
IMAGE AUGMENTATION PARAMETERS

| Augmentation Type | Scale | Scale Type |
|---|---|---|
| Hue-HSV | 0.015 | Fraction |
| Saturation-HSV | 0.7 | Fraction |
| Value-HSV | 0.4 | Fraction |
| Translation | 0.1 | Fraction (±) |
| Scaling | 0.5 | Fraction (±) |
| Horizontal Flip | 0.5 | Probability |
| Mosaic | 1.0 | Probability |

*E. Hyperparameters*

Hyperparameters are values that determine the architecture and training of a machine learning model. Hyperparameters differ from model parameters because they are established before any learning takes place. While model parameters are adjusted as the model learns from data, hyperparameters are not modified during the training phase and stay constant [19]. The right hyperparameter settings change based on the specific problem the model is trying to tackle. The effectiveness of a model can be drastically impacted by carefully selecting the correct hyperparameters [20]. In object detection, several hyperparameters can be set, including the learning rate, batch size, anchor box, weight decay, and others. In this research, we conducted scenarios by changing the learning rate hyperparameters and experimented with full training and partial training on the pre-trained YOLOv5 COCO model dataset can be seen in Table IV.

TABLE IV
MODELING SCENARIOS

| Key | Scheme | Remarks |
|---|---|---|
| A | Baseline | Adam (lr = 0.01); Freezed 0 |
| B | Fully training | Adam (lr = 0.005); Freezed 0 |
| C | Fully training | Adam (lr = 0.0001); Freezed 0 |
| D | Partial training | Adam (lr = 0.01); Freezed 2 |
| E | Partial training | Adam (lr = 0.01); Freezed 6 |
| F | Evolution 1 | GA: 150 generations |
| G | Evolution 2 | GA: 300 generations |
| H | Evolution 3 | GA: 450 generations |

The key column in the table is a unique scenario that we did. Partial training involves freezing some the pre-trained model layers so that the weights on those layers do not change during the fine-tuning process [21]. This approach aids in boosting the model's effectiveness by keeping the information it has previously learned intact. The anchor box is a hyperparameter used to detect objects with different scales and aspect ratios [22]. When adapting a pre-trained model, it's crucial to select appropriate anchor box values. This is because the dimensions and positions of the objects in your specific training data might not match the default anchor box values the pre-trained model was initially designed with. If the anchor values are not properly set, the model may not perform well in detecting objects in the training data. In previous versions of YOLO, anchor boxes were determined using the K-Means clustering algorithm. In YOLOv5, a new method called auto-anchor is introduced that automatically learns the anchor box values from the training data [23]. This is especially useful when fine-tuning the model with custom datasets because it ensures that the anchor values are properly set and the model can accurately detect objects in the training data. To prevent the model from being overfitted, we used early stopping. This means the training stopped automatically if the model didn't get any better after a set amount of training cycles. Specifically, we chose 100 cycles without improvement as the cutoff point in this study [24]. In our study, we utilized the mini-batch

gradient descent method to update the model weights. This approach updates the importance of each neuron using a group of training examples all at once, instead of adjusting them individually after each single example. A commonly used batch size of 32 was used in all scenarios. By using mini-batches, we were able to train our model more efficiently. This allowed the model to learn from a smaller subset of the training data, which can help reduce overfitting to the training data. Additionally, updating the weights more frequently with mini-batches can help improve the model's performance.

## III. Results and Discussion

This section concentrates on evaluating how well the fine-tuned YOLOv5 model performs in identifying drowsy drivers. To rigorously assess the efficacy of our fine-tuned model, we will employ a suite of established performance metrics. These metrics, crucial for evaluating the model's capabilities, include recall, precision, and mean average precision (mAP). These metrics will help us understand how well the model can identify and locate the signs of fatigue. We will also benchmark the performance of the fine-tuned build with and without hyperparameters tuning evolution.

### A. Performance Metrics

In considering the model's performance, we employed a suite of benchmark metrics, namely mean average precision (mAP), recall, and precision. Precision (P) quantifies the precision of the model's positive predictions, reflecting the proportion of instances identified as positive that are indeed truly positive. Complementary to precision, recall (R) examines the model's ability to recognize all pertinent positive instances within the dataset, representing the proportion of actual positive cases correctly classified as such. In object detection, the Intersection over Union (IoU) metric is utilized to calculate the intersection between the actual bounding box and the estimated bounding box. The IoU is 1.0 if the boxes overlap perfectly, and 0.0 if they do not. The mAP50 score shows how accurate the model is, using an IoU of 0.5. Essentially, a predicted object is considered a true positive if its bounding box has an overlap of 50% or more with the actual, correct bounding box. The equations used to evaluate the model performance are shown in Eq. (2)-(5) [25]-[26].

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$AP = \frac{1}{n}\sum_{i=1}^{n} \text{Precision}_i \tag{4}$$

$$mAP = \frac{\sum_{i=1}^{Q} AP_i}{Q} \tag{5}$$

### B. Loss and Performance Metrics Graphs Analysis

We plot the loss and performance metrics graphs for the fine-tuned YOLOv5 model under various scenarios. The loss graph in Fig. 6-8 showcases the box, objectness, and classification losses over the number of epochs, while the performance metrics graph are shown in Fig. 6-8 exhibit precision, recall, and mean mAP for each scenario. Box loss indicates how precisely the model locates the center of an object and how closely the drawn box fits around the object it has identified. Objectness loss is used to assess the probability of object presence in the proposed region of interest. Classification loss is used to assess the model's proficiency in categorization the class of detected objects [27]. As the training advances through more epochs, the loss values go down, suggesting that the model gets better at predicting the object's central location, the size and position of the box around it, and what type of object it is. The precision, recall, and mAP values also increase with the number of epochs, indicating that the model's effectiveness in identifying and categorizing objects improves with more training.

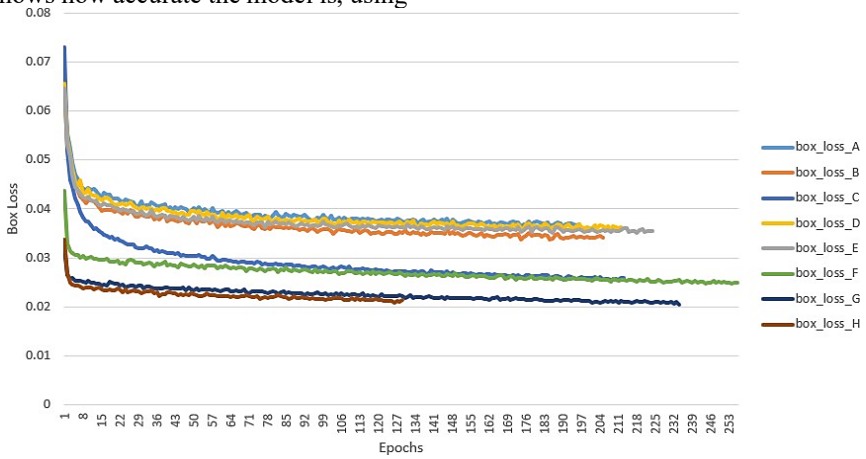



Fig 6. Box loss: This loss function is used to fine-tune the accuracy of the predicted bounding box locations. It quantifies the difference between where the model thinks the box is, and where it actually is according to the labeled data.

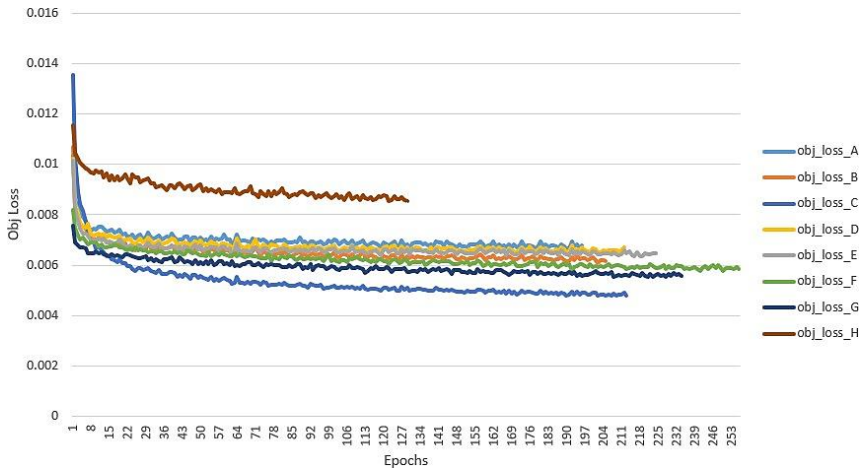



Fig. 7. Objectness loss: This loss function is responsible for determining whether something is there in a particular grid cell or not. It helps in reducing false positives.

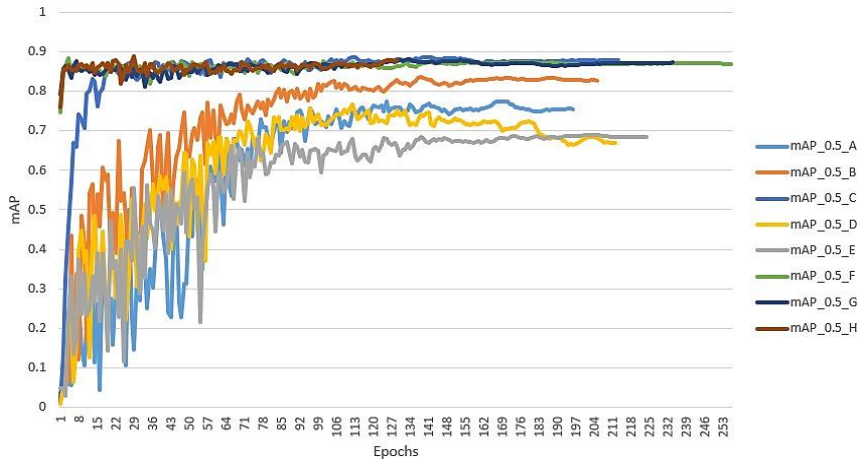



Fig. 8. mAP: it is the average precision at different recall levels across all queries, where each query represents the detection of a particular class of objects in the image.

From the graph, it can be seen that each scenario reaches a different number of epochs before stopping. This happens because we use a technique called early stopping. It cuts off the training when the model's performance plateaus and doesn't get any better for a set number of training cycles. The graph shows that scenario E uses the most epochs, while scenario B uses the least. An interesting finding in this study is that freezing the first six layers of the backbone requires more epochs to train. However, this scenario results in the worst accuracy, possibly due to the model getting stuck in a local optimum and failing to converge. Scenario C, which used the lowest learning rate, required the most epochs to train the model. However, this resulted in the best performance of all the scenarios. This is because using a low learning rate means that the weights in the model are updated slowly, with minor changes at each iteration. According to the mAP graph, the most effective model is the one that has the slowest learning rate (scenario C), followed by scenario B.

With early stopping, the best weights can be saved at each additional epoch. Scenarios F, G, and H are training scenarios that use evolved hyperparameters for particular generation. The graph in Fig. 6 -8 shows that these evolved hyperparameters from GA can improve the model's performance regarding box and classification loss compared to the other scenarios. The longer the evolution process, the better the model's performance regarding these metrics. The hyperparameters developed using GA did not show a significant improvement in objectness loss, precision, recall, or mAP values compared to the best hyperparameters from scenario C that did not use GA. To achieve a larger improvement in performance, it may be necessary to use additional epochs in each evolutionary generation and to generate more evolutionary generations. However, this would require a significant amount of computation. These results indicate that further research may be necessary to identify more effective ways adjust the model using evolved hyperparameters.

### *C. Model Performance and Analysis*

The outcomes of each experiment are presented in Table V, which demonstrates the performance of the model for each scenario. These tables provide an overview of the Precision (P), Recall (R), and mAP50 of the model in detecting drowsiness, and allow us to compare the results of each experiment to identify the best scenario.

Based on the test results in Table V, it can be seen that scenario C (full training with a learning rate of 0.0001) produces the highest mAP value for the scenario before hyperparameter evolution. This suggests that lower learning rates tend to have better model performance. It is supported by the test results in scenario B, where the learning rate is set to 0.005, and the mAP is lower than in scenario C. Scenario A, which uses the default hyperparameters with a learning rate of 0.01, has an mAP value of 0.757, which is lower than in scenario C and B. Scenario E, which involves partial training by freezing the first six layers of the backbone, has an mAP of 0.681, which is lower than the baseline model (scenario A). However, scenario D, which involves partial training with the first two layers of the backbone frozen, has an mAP of 0.767, which is higher than the baseline model. This suggests that partial training can improve the model's performance if the proper layers are frozen.

TABLE V
PERFORMANCE METRICS WITHOUT EVOLUTION SCHEME

| Key | Class | Precision | Recall | mAP |
|---|---|---|---|---|
| A | All | 0.707 | 0.73 | 0.757 |
| | Closed | 0.766 | 0.71 | 0.695 |
| | Normal | 0.534 | 0.931 | 0.892 |
| | Yawning | 0.822 | 0.549 | 0.686 |
| B | All | 0.831 | 0.797 | 0.826 |
| | Closed | 0.822 | 0.778 | 0.768 |
| | Normal | 0.905 | 0.888 | 0.937 |
| | Yawning | 0.767 | 0.724 | 0.772 |
| C | All | 0.832 | 0.86 | 0.887 |
| | Closed | 0.725 | 0.908 | 0.873 |
| | Normal | 0.9 | 0.929 | 0.959 |
| | Yawning | 0.872 | 0.742 | 0.829 |
| D | All | 0.787 | 0.745 | 0.767 |
| | Closed | 0.77 | 0.793 | 0.739 |
| | Normal | 0.805 | 0.89 | 0.899 |
| | Yawning | 0.788 | 0.552 | 0.664 |
| E | All | 0.705 | 0.693 | 0.681 |
| | Closed | 0.746 | 0.643 | 0.666 |
| | Normal | 0.698 | 0.915 | 0.86 |
| | Yawning | 0.671 | 0.52 | 0.517 |

Low learning rates result in high mAP values because the pre-trained weights are not updated extensively, so the weights trained on the COCO dataset are not damaged during the training process. In contrast, partial training with more layers frozen produce poor model performance because the drowsiness detection with facial structures is fairly different from the pre-trained weights in the COCO dataset. Therefore, many layers need to be updated with small, gradual changes to the weights. However, freezing the first two layers (scenario D) produces better results than full training because the first layer of pre-trained weights already extracted low-level features such as basic local shapes, edges, and corners, which are useful for drowsiness detection. Therefore, the first layer does not need fine-tuning in this case.

According to Table VI, the model's performance in detecting the "yawning" class is poor, especially when considering the recall metric. This is because the dataset used has an imbalanced class distribution, with only 706 "yawning" labels, while the "normal" and "eye closed" classes have 2653 and 2347 labels, respectively. This shows that YOLOv5 is unable to effectively handle imbalanced classes on this dataset. One possible way to address the imbalanced class problem is to use data augmentation on specific classes. This can be done by applying transformations to the images in the minority class to generate additional data. This method can boost how well the model works by adding more examples of the less common type, leading to a more even spread of different types of data.

TABLE VI
PERFORMANCE METRICS WITH EVOLUTION SCHEME

| Key | Class | P | R | mAP |
|---|---|---|---|---|
| F | All | 0.835 | 0.836 | 0.874 |
| | Closed | 0.722 | 0.907 | 0.849 |
| | Normal | 0.894 | 0.917 | 0.956 |
| | Yawning | 0.89 | 0.685 | 0.815 |
| G | All | 0.844 | 0.83 | 0.882 |
| | Closed | 0.738 | 0.88 | 0.869 |
| | Normal | 0.889 | 0.917 | 0.95 |
| | Yawning | 0.905 | 0.693 | 0.828 |
| H | All | 0.854 | 0.85 | 0.888 |
| | Closed | 0.76 | 0.893 | 0.876 |
| | Normal | 0.941 | 0.906 | 0.959 |
| | Yawning | 0.86 | 0.749 | 0.83 |

Another way to address the imbalanced class problem is to generate synthetic data. This means using computer programs that learn to produce fresh information that resembles the original data, but possesses some altered qualities. This approach can make the model learn better and improve its output on the minority class. In addition to these techniques, another possible approach to addressing the imbalanced class problem is to use a weighted loss function. It involves assigning higher weights during training to the samples in the minority class, so the model gives them more importance. This approach allows the model to concentrate more on understanding the features of the less frequent group, which can lead to better results when dealing with that specific group.

To address class imbalance, consider employing techniques that balance the dataset, such as boosting the count of minority class examples (oversampling) or reducing the number of majority class instances (undersampling). This helps to address imbalances in the class distribution, ultimately boosting the model's capability to accurately classify minority classes. Ultimately, the method for tackling the class imbalance

issue will depend on the dataset's specific characteristics and the model's goals. Table VII compares the performance of the YOLOv5 model across three different evolution scenarios, with 150, 300, and 450 generations. The results show that scenario H, with 450 generations of evolution, outperforms scenario C, achieving an mAP of 0.888. This suggests that longer evolution processes can improve the performance of the detection model. Table VII present the best hyperparameters and augmentation scales, obtained from 450 generations of the evolution process.

TABLE VII
THE HYPERPARAMETERS AND AUGMENTATION SCALES FROM EVOLUTION PROCESS

| Augmentation Type | Scale | Scale Type |
|---|---|---|
| lr0 | 7.0e-05 | hsv_h |
| momentum | 0.97767 | hsv_s |
| weight_decay | 0.00041 | hsv_v |
| warmup_epochs | 2.4762 | degrees |
| warmup_momentum | 0.87464 | translate |
| box | 0.04431 | scale |
| cls | 0.59496 | shear |
| cls_pw | 0.74023 | perspective |
| obj | 1.8035 | flipud |
| obj_pw | 1.1885 | fliplr |
| iou_t | 0.2 | mosaic |
| anchor_t | 2.9315 | anchors |

Fig. 9 presents example images illustrating the top-performing model's real-time inference capabilities. The pictures demonstrate the model's precise detection of tiredness states. The model was tested using an Intel Core i5-8250 CPU device and 8GB of RAM, and it was able to achieve an FPS of 30. This shows that the model is efficient and can be easily deployed on edge devices with low specifications.

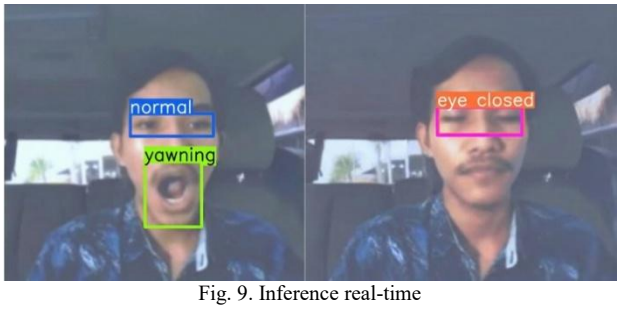


Fig. 9. Inference real-time

### D. Limitations

One limitation of this study is the imbalanced distribution of dataset classes, where the yawning class contains substantially fewer samples than the normal and eye-closed classes. This imbalance affects the recall performance of the yawning category and may limit the model's ability to generalize to rare fatigue events. The dataset was compiled from both public sources and self-recorded images. Although this increases dataset diversity, potential demographic and environmental biases may still exist due to limited subject representation and recording conditions.Although the dataset includes various lighting conditions, the model has not been systematically evaluated under extreme illumination scenarios such as nighttime driving, strong backlighting, or rapidly changing lighting environments. The real-time evaluation was conducted on a single CPU-based platform. Therefore, additional validation on embedded automotive hardware such as Raspberry Pi, Jetson Nano, or in-vehicle computing systems is required before large-scale deployment.

## IV. Conclusion

This This study proposed a driver fatigue detection system based on YOLOv5s for recognizing three facial states associated with fatigue, namely normal, eye-closed, and yawning. Transfer learning was applied using pre-trained COCO weights, and multiple fine-tuning strategies were evaluated, including full training, partial layer freezing, and genetic-algorithm-based hyperparameter evolution. The experimental results showed that the best model without hyperparameter evolution achieved an mAP of 0.887, while the best evolved model achieved an mAP of 0.888. These results indicate that GA-based hyperparameter optimization can provide a modest improvement in detection performance when combined with an appropriate fine-tuning strategy.

In addition to detection accuracy, this study emphasized deployment feasibility on resource-constrained edge devices. The proposed model achieved real-time inference at approximately 30 FPS on a CPU-based platform (Intel Core i5-8250U with 8 GB RAM), demonstrating its suitability for practical implementation in Advanced Driver Assistance Systems (ADAS) and other driver monitoring applications. Furthermore, unlike conventional fatigue detection approaches that rely on manually defined threshos, the proposed method automatically learns fatigue-related facial features directly from image data, improving adaptability across different users and driving conditions.

Future work will focus on improving the robustness and generalization capability of the proposed system through class imbalance handling techniques, such as weighted loss functions and focal loss, as well as expanding dataset diversity across subjects and environmental conditions. Additional validation under real-world driving scenarios and deployment on embedded automotive hardware will also be investigated to further assess the practical applicability of the proposed approach.

## Conflict of Interest

The authors declare that they have no conflict of interest regarding the publication of this paper.

## References


[1] J. M. Owens *et al.*, "Prevalence of Drowsy Driving Crashes: Estimates from a Large-Scale Naturalistic Driving Study," Research Brief, AAA Foundation for Traffic Safety, Washington, D.C., Feb. 2018.

[2] A. Rahman, M. B. H. Hriday, and R. Khan, "Computer vision-based approach to detect fatigue driving and face mask for edge computing device," *Heliyon*, vol. 8, no. 10, 2022, Art. no. e11204, doi: 10.1016/j.heliyon.2022.e11204.

[3] L. Du, R. Zhang, and X. Wang, "Overview of two-stage object detection algorithms," *J. Phys. Conf. Ser.*, vol. 1544, no. 1, 2020, Art. no. 012033, doi: 10.1088/1742-6596/1544/1/012033.

[4] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using YOLO: challenges, architectural successors, datasets and applications," *Multimedia Tools Appl.*, vol. 82, no. 6, pp. 9243–9275, 2023, doi: 10.1007/s11042-022-13644-y.

[5] F. D. H. Fikriyah, A. Nugroho, and A. F. Suni, "Eye Fatigue Detection in Vehicle Drivers Based on Facial Landmarks Features," *REKAYASA J. Penerapan Teknol. dan Pembelajaran*, vol. 19, no. 1, pp. 11–19, 2021, doi: 10.15294/rekayasa.v19i1.28699.

[6] S. B. Meshram and S. Bodkhe, "Fatigue Detection based on Image Eye Tracking and Detection for avoiding accidents on Roads," 2016.

[7] A. C. Phan, N. H. Q. Nguyen, T. N. Trieu, and T. C. Phan, "An efficient approach for detecting driver drowsiness based on deep learning," *Appl. Sci.*, vol. 11, no. 18, Sep. 2021, Art. no. 8441, doi: 10.3390/app11188441.

[8] R. Ribani and M. Marengoni, "A Survey of Transfer Learning for Convolutional Neural Networks," in *Proc. 32nd SIBGRAPI Conf. Graphics, Patterns Images Tutorials (SIBGRAPI-T)*, 2019, pp. 47–57, doi: 10.1109/SIBGRAPI-T.2019.00010.

[9] N. Ho and Y.-C. Kim, "Evaluation of transfer learning in deep convolutional neural network models for cardiac short axis slice classification," *Sci. Rep.*, vol. 11, no. 1, 2021, Art. no. 1839, doi: 10.1038/s41598-021-81525-9.

[10] D. L. Shanthi and N. Chethan, "Genetic Algorithm Based Hyper-Parameter Tuning to Improve the Performance of Machine Learning Models," *SN Comput. Sci.*, vol. 4, no. 2, 2022, Art. no. 119, doi: 10.1007/s42979-022-01537-8.

[11] T. Alzyadat, M. Yamin, and G. Chetty, "Genetic algorithms for the travelling salesman problem: a crossover comparison," *Int. J. Inf. Technol.*, vol. 12, no. 1, pp. 209–213, 2020, doi: 10.1007/s41870-019-00377-9.

[12] M. Sonka, V. Hlavac, and R. Boyle, "Image pre-processing," in *Image Processing, Analysis and Machine Vision*, M. Sonka, V. Hlavac, and R. Boyle, Eds. Boston, MA, USA: Springer US, 1993, pp. 56–111, doi: 10.1007/978-1-4899-3216-7_4.

[13] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J. Big Data*, vol. 6, no. 1, 2019, Art. no. 60, doi: 10.1186/s40537-019-0197-0.

[14] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *Procedia Comput. Sci.*, vol. 199, pp. 1066–1073, 2022, doi: 10.1016/j.procs.2022.01.135.

[15] S. Han, X. Dong, X. Hao, and S. Miao, "Extracting Objects' Spatial-Temporal Information Based on Surveillance Videos and the Digital Surface Model," *ISPRS Int. J. Geo-Inf.*, vol. 11, no. 2, Feb. 2022, Art. no. 103, doi: 10.3390/ijgi11020103.

[16] A. C. Aqsa, H. Mahmudah, and R. W. Sudibyo, "Detection and Classification of Road Damage Using CNN with Hyperparameter Optimization," in *Proc. 6th Int. Conf. Informatics Comput. Sci. (ICICoS)*, 2022, pp. 101–104, doi: 10.1109/ICICoS56336.2022.9930607.

[17] Erwin, "Improving Retinal Image Quality Using the Contrast Stretching, Histogram Equalization, and CLAHE Methods with Median Filters," *Int. J. Image, Graphics Signal Process.*, vol. 12, no. 2, pp. 30–41, Apr. 2020, doi: 10.5815/ijigsp.2020.02.04.

[18] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Contrast Stretching," The University of Edinburgh School of Informatics. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm. (Accessed: Nov. 27, 2022).

[19] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb," *J. Electron. Sci. Technol.*, vol. 17, no. 1, pp. 26–40, 2019, doi: 10.11989/JEST.1674-862X.80904120.

[20] Z. S. Hernanda, H. Mahmudah, and R. W. Sudibyo, "CNN-Based Hyperparameter Optimization Approach for Road Pothole and Crack Detection Systems," in *Proc. IEEE World AI IoT Congr. (AIIoT)*, 2022, pp. 538–543, doi: 10.1109/AIIoT54504.2022.9817316.

[21] F. J. P. Montalbo, "Truncating a densely connected convolutional neural network with partial layer freezing and feature fusion for diagnosing COVID-19 from chest X-rays," *MethodsX*, vol. 8, 2021, Art. no. 101408, doi: 10.1016/j.mex.2021.101408.

[22] Y. Zhong, J. Wang, J. Peng, and L. Zhang, "Anchor Box Optimization for Object Detection," in *Proc. IEEE Winter Conf. Appl. Comput. Vision (WACV)*, 2020, pp. 1275–1283, doi: 10.1109/WACV45572.2020.9093498.

[23] J. Wang *et al.*, "Toward surface defect detection in electronics manufacturing by an accurate and lightweight YOLO-style object detector," *Sci. Rep.*, vol. 13, no. 1, 2023, Art. no. 7062, doi: 10.1038/s41598-023-33804-w.

[24] H. Darmawan, M. Yuliana, and Moch. Z. S. Hadi, "GRU and XGBoost Performance with Hyperparameter Tuning Using GridSearchCV and Bayesian Optimization on an IoT-Based Weather Prediction System," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 13, no. 3, pp. 851–862, 2023, doi: 10.18517/ijaseit.13.3.18377.

[25] H. Darmawan, M. Yuliana, and Moch. Z. Samsono Hadi, "Realtime Weather Prediction System Using GRU with Daily Surface Observation Data from IoT Sensors," in *Proc. Int. Electron. Symp. (IES)*, 2022, pp. 221–226, doi: 10.1109/IES55876.2022.9888468.

[26] G. Dai, L. Hu, J. Fan, S. Yan, and R. Li, "A Deep Learning-Based Object Detection Scheme by Improving YOLOv5 for Sprouted Potatoes Datasets," *IEEE Access*, vol. 10, pp. 85416–85428, 2022, doi: 10.1109/ACCESS.2022.3192406.

[27] A. Hussain, B. Barua, A. Osman, R. Abozariba, and A. T. Asyhari, "Low Latency and Non-Intrusive Accurate Object Detection in Forests," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2021, doi: 10.1109/SSCI50451.2021.9660175.


## Author Biography



**Hendri Darmawan** is a Full-Time Faculty member at the Department of Internet Engineering Technology, Politeknik Elektronika Negeri Surabaya (PENS), Indonesia. He received both his Bachelor's and Applied Master's degrees from PENS with cum laude honors. His research interests include artificial intelligence, computer vision, machine learning, edge computing, Internet of Things (IoT), and smart agriculture. He has published numerous papers in IEEE and Elsevier-indexed journals and serves as a reviewer for several international journals.



**Zulfa Muflihah** is currently pursuing the Master's degree in Engineering Physics at Institut Teknologi Sepuluh Nopember (ITS), Indonesia. She received her Bachelor's degree from ITS and a Diploma in Instrumentation System Automation from Universitas Airlangga. Her research interests include data science, artificial intelligence, computer vision, automation, and intelligent instrumentation systems. She has participated in research internships at BRIN and Taiwan Experience Education Program (TEEP) and has earned several professional certifications in data science.



**Giran Arisdi** is an undergraduate Master Student in Department of Electrical Engineering at Politeknik Elektronika Negeri Surabaya (PENS), Indonesia. His research focuses on artificial intelligence, computer vision, deep learning, edge AI, embedded systems, and smart agriculture. He is actively involved in research on lightweight deep learning models, edge-device deployment, and intelligent vision systems for precision agriculture and real-time monitoring applications.