Vision Based Drowsiness Detection System Using Deep Learning

COMP7116001 - Computer Vision

Team Members

2702229932 - Bertrand Geraldo Tjahyadi

2702246200 - Jose Christian Wijaya

2702248686 - Yudhistira

2702331100 - Reinhart Gian Widjaja

2025

Abstract

Drowsiness is a critical physiological state that significantly impairs cognitive functions, reaction times, and decision-making capabilities, leading to severe safety risks in high stakes environments such as transportation. This research presents the development of a non-intrusive, Vision Based Drowsiness Detection System designed to monitor fatigue levels in real-time. The proposed system employs two stages of computer vision pipeline. First, we utilized YOLOv2 based architecture for robust eye detection, and second, comparing a custom Convolutional Neural Network (CNN) against a transfer learning based MobileNetV2 for eye state classification (Open vs. Closed).

Experimental results on a composite dataset of 10,000 images demonstrate a distinct trade-off between computational efficiency and classification performance. While the MobileNetV2 model achieved superior theoretical accuracy (97.47%) and perfect recall for closed eyes, the custom CNN offered significantly lower latency (0.54 ms per image) and greater stability during real-world deployment simulations. The system integrates these predictions into a "Drowsiness Safety Monitor" application that calculates Percentage of Eye Closure (PERCLOS) to trigger alerts. This study validates the feasibility of low-cost, webcam-based fatigue monitoring while highlighting the challenges of lighting variations and head pose occlusion in real-world scenarios.

Keywords: *Drowsiness Detection, Computer Vision, Deep Learning, CNN, MobileNetV2, Transfer Learning, PERCLOS, Real-Time Monitoring*

Chapter I
Introduction

1.1 Background

In modern society, the safety of transportation and operational systems relies heavily on human alertness. However, fatigue and drowsiness remain pervasive physiological factors contributing to accidents. According to global safety statistics, drowsy driving is a leading cause of traffic collisions, often resulting in severe injuries and significant economic losses [1]. Unlike driving under the influence, drowsiness is insidious, its onset is gradual, and individuals might fail to recognize the condition or severeness of their own fatigue until it is too late.

The action of "microsleeps", brief, uncontrollable moments of zoning out that can last from a fraction of a second to several seconds poses a severe threat, particularly on monotonous roads or during long shifts [2]. Traditional methods of preventing this rely on driver self-awareness or indirect vehicle signals (such as lane deviation), which are often reactive rather than proactive. With the advent of Artificial Intelligence and Computer Vision, there is a paradigm shift toward "active safety systems." By using indicators such as eye closure rates, intelligent systems can detect the onset of drowsiness before a critical failure occurs, providing a reliable layer of automated protection [3].

1.2 Problem Statement

Detecting drowsiness presents a complex challenge due to the subtle and subjective nature of fatigue signs. While physiological measures like EEG (electroencephalogram) are accurate, they are intrusive and impractical for real-world usage. Conversely, non-intrusive vision-based systems face significant hurdles regarding environmental variability. Drivers often overestimate their alertness and may not admit to high fatigue levels until a safety-critical incident is imminent. Indirect indicators (e.g., steering patterns) often manifest only after drowsiness is well advanced. A delay in detection by even a few seconds can be fatal at highway speeds. Real-time monitoring requires robustness against varying lighting conditions, head movements, and occlusions (e.g., glasses), which simple detection algorithms frequently fail to handle [4]. There is a need for a system that is accurate, computationally efficient for real-time processing, and capable of functioning autonomously without user intervention.

1.3 Objective

The primary objective of this research is to develop an autonomous Vision-Based Drowsiness Detection System. Specifically, this study aims to:

1. Design and train deep learning models to accurately detect faces and classify eye states (Open/Closed) in real-time.
2. Compare the performance of a custom lightweight CNN against a fine-tuned MobileNetV2 architecture to identify the optimal balance between accuracy and latency.
3. Implement a prototype application that utilizes the PERCLOS metric to provide early warnings, thereby maintaining operator focus and safety.

1.4 Significance

This research holds significant value for the field of transportation systems and public safety. By validating a vision-based approach using low-cost hardware (standard webcams), this study enables access to advanced safety features often reserved for luxury vehicles. Furthermore, the comparative analysis between custom and transfer learning models provides technical insights into deploying deep learning on resource constrained devices. Ultimately, reliable real-time drowsiness detection serves as a critical proactive measure that might save lives by intervening before accidents occur.

Chapter II
Related Works

Drowsiness detection has been a subject of extensive research, with methodologies generally categorized into three primary approaches: Vehicle-based, Physiological-based, and Behavioral-based.

2.1 Vehicle-Based Approaches

Vehicle-based methods monitor the deviations in the vehicle's behavior to infer the driver's state. These systems utilize sensors to track metrics such as steering wheel movement patterns, pressure on the acceleration pedal, and lane deviations [5]. For instance, a sudden jerk in the steering wheel or a gradual drift out of the lane often signifies a loss of concentration. While these methods are non-intrusive, they are limited by their reactive nature, they detect the consequence of drowsiness, meaning the driver may already be in a dangerous state before the system alerts them.

2.2 Physiological-Based Approaches

Physiological methods are considered the "gold standard" for accuracy as they measure the body's direct biological responses. These include Electrocardiograms (ECG) to monitor heart rate variability, Electrooculograms (EOG) to measure eye movements, and Electroencephalograms (EEG) to analyze brain wave activity [6]. Research shows that an increase in alpha and theta brain waves is strongly correlated with fatigue. However, the requirement for attaching electrodes or sensors to the user's body makes these systems intrusive and highly impractical for everyday driving or working scenarios.

2.3 Behavioral Based (Computer Vision) Approaches

Behavioral approaches utilize cameras to monitor visual signs of fatigue, such as yawning frequency, head tilting, and eye closure rates. This research focuses on this category due to its balance of accuracy and being not intrusive. The most common metric used is PERCLOS (Percentage of Eye Closure), which calculates the proportion of time the eyes are closed over a specific window. Recent advancements in Deep Learning, specifically Convolutional Neural Networks (CNNs), have significantly improved the robustness of these systems. Techniques ranging from Haar Cascades to advanced architecture like You Only Look Once (YOLO) and MobileNet have been applied to detect faces and eyes with high precision, even in varying environments [7].

Chapter III
Methodology

3.1 Eye Detection

3.1.1 Dataset

The dataset that we used to train our eye detection model consists of 1521 portable gray maps (PGM) images. Each image is accompanied by a corresponding .eye file that contains the bounding box annotations for the subject's eyes. The dataset is obtained from the BioID Face Database [8].

Each image is stored in a pgm file that contains a data header and the image data itself. The image files begin with a header that specifies the structure of the data. The first line indicates the image data format, where the identifier "P5" denotes that the data is stored in binary form. The second and third lines specify the image width and height, respectively, both written in text form. The fourth line defines the maximum permitted gray value, which is 255 for the images used in this work. Following the header, the image data is stored as a contiguous data block, with pixel values arranged line by line from top to bottom, using one byte per pixel.

The eye position files are plain text files that begin with a single comment line, followed by four values representing the *x* and *y* coordinates of the left eye and the *x* and *y* coordinates of the right eye, separated by spaces. The left and right eyes are defined from the subject's perspective; as a result, when viewed from the camera, the subject's left eye appears on the right side of the image, and the right eye appears on the left side.

As the dataset provides only the center coordinates of each eye, a fixed-size bounding box was constructed around each center point. Specifically, a bounding box with a width of 50 pixels and a height of 30 pixels was defined for each eye to for model training.

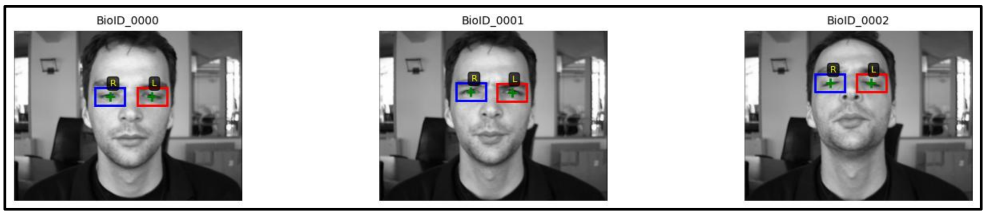

Figure 3.1 Constructed Bounding Box on Dataset Examples

3.1.2 Preprocessing

For the preprocessing stage, several steps were applied to prepare the dataset for effective model training. First, the images were converted to grayscale to reduce computational complexity and training time [9]. Second, all images were resized to 224 x 224 to ensure input dimensional consistency across the dataset. Third, all pixel values were normalized by scaling them to the [0, 1] range through division by 255. Fourth, to enable training of a YOLOv2-style model, the bounding box annotations were transformed into the YOLO format by normalizing the center coordinates and box dimensions to the [0,1] interval [10]. At the end, we split the images for validation with an 80/20 ratio.

3.1.3 Architecture

The model we build begins with a sequence of convolutional blocks that gradually increase the number of feature channels from 32 to 1024, while spatial resolution is reduced using max pooling layers. Each block consists of convolution, batch normalization, and Leaky ReLU activation, which helps stabilize training and improve feature learning. Both standard and pointwise convolutions are used, where the latter refines feature representations without changing spatial size. After feature extraction, a compact detection head is applied, consisting of an additional convolutional block followed by a final convolution layer that produces predictions for multiple anchor boxes at each grid cell. The output represents object confidence, bounding box parameters, and class probability for eye detection.

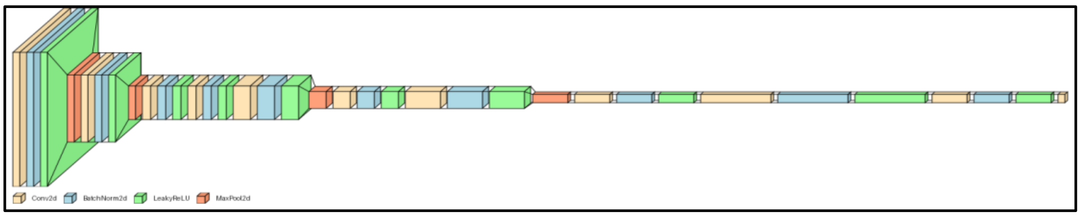

Figure 3.2 Convolutional Neural Network Architecture

3.1.4 Training Setup

The model was trained using a fixed batch size of 16 for a maximum of 50 epochs, with an initial learning rate of 0.001. Input images were resized to 224 by 224 pixels and predictions were made over a 7 by 7 spatial grid. Training was guided by a YOLOv2-style loss function composed of multiple weighted components. The coordinate regression loss applied mean squared error to the predicted bounding box offsets, with a higher weight assigned to localization accuracy. Object and no-object losses were computed using binary cross-entropy to distinguish between grid cells containing an eye and background regions, while a binary cross-entropy loss was used for the single-class classification task.

Optimization was performed using the Adam optimizer with an adaptive learning rate. A ReduceLROnPlateau scheduler was employed to monitor validation loss and reduce the learning rate by half if no improvement was observed for four consecutive epochs, with a minimum learning rate of 1e-6. Early stopping was applied with a patience of three epochs to prevent overfitting, and the model parameters corresponding to the lowest validation loss were saved.

During training, each batch was passed through the network, the loss was computed using anchor-based target encoding, and model parameters were updated through backpropagation. Validation was conducted at the end of each epoch, and training progress was monitored using loss curves, learning rate changes, and epoch-level performance metrics.

The model stopped early in Epoch 12 based on the early stopping criterion. The corresponding training and validation loss curves are shown below.

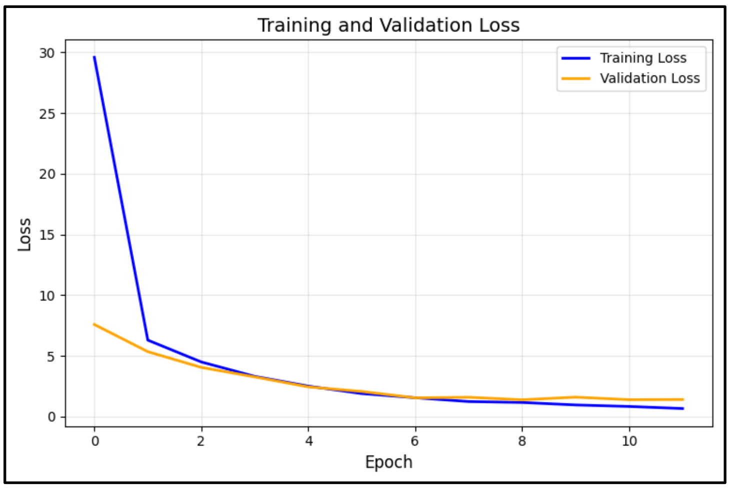

Figure 3.3 Training and Validation Loss of YOLO Eye Detector

3.1.5 Evaluation Metrics

To evaluate the model, we used the Average Precision metric over an Intersection-over-Union of 0.5. This metric tells us how well the model's ability is to correctly localize and classify objects while minimizing false positives. More specifically, AP is computed as the area under the precision–recall curve. As the confidence threshold varies, precision and recall change; AP captures the overall performance across this entire range rather than at a single operating point. A higher AP value means the model consistently achieves high precision without sacrificing recall.
From our evaluation, the model produced a mean AP score of 0.7327 across 305 images. This tells us that the model shows reliable detection performance and can detect eyes accurately with relatively few false positives. The distribution of AP is shown below:

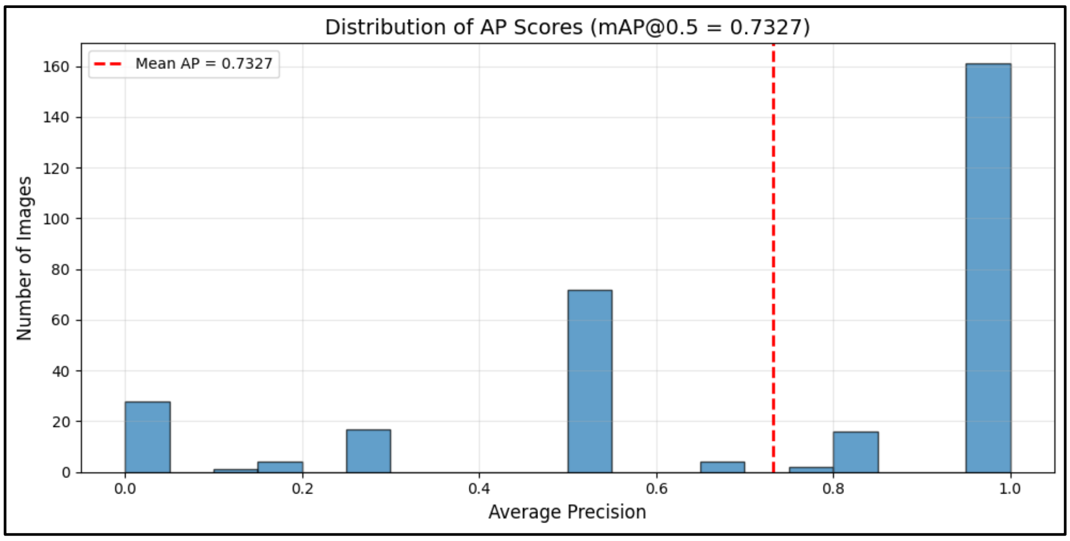

Figure 3.4 Average Precision Scores Distribution

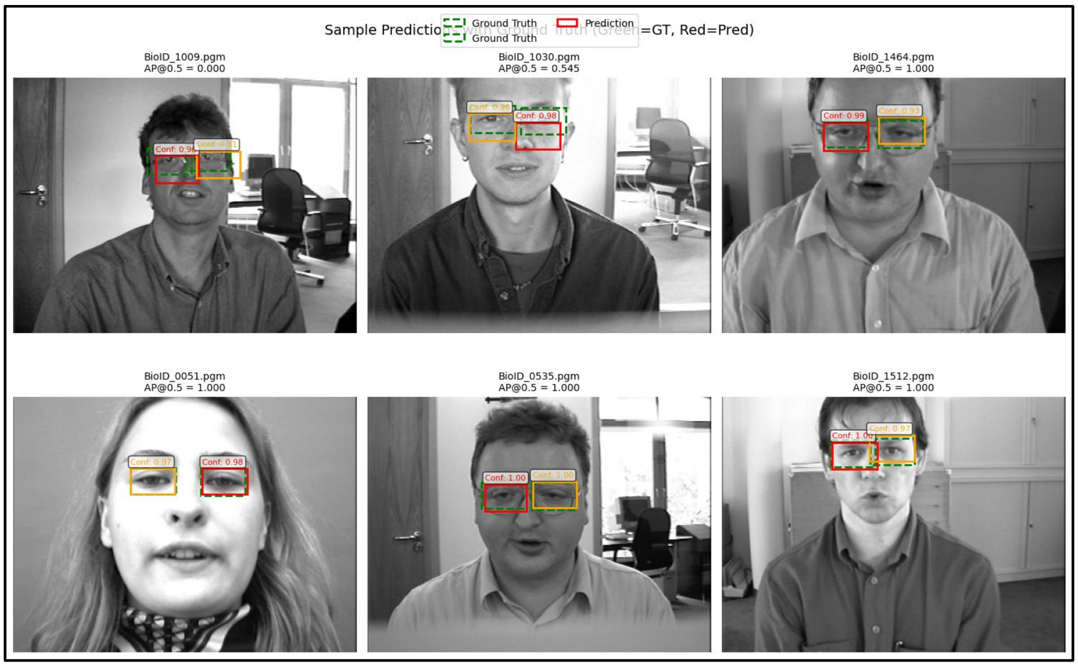

Figure 3.5 Sample Predictions using YOLO model

3.2 Eye State Classification

3.2.1 Dataset

The dataset used in this experiment is an aggregation of the MRL Eye Dataset (Version 3) by Prasad V Patil [11], the Closed Eyes in Wild (CEW) datasets, as well as collected images from the author, a total of exactly 10,000 images. The task is a binary classification task, to distinguish between two classes, 'Open' and 'Closed' eyes. The images were taken under varying conditions such as lighting, distance, resolution, face angle, and eye angle.

Each image follows the naming convention of MRL, where the prefix sXXXX signifies the subject's identity. The naming convention is not explicitly stated by the author, but the file names follow the naming convention of MRL Eye Dataset [12]. In this subject, only the subject id and eye open/closed label were used for training and evaluation. The structure in order is as follows:

- subject ID
- image ID
- gender [0 - man, 1 - woman]
- glasses [0 - no, 1 - yes]
- eye state [0 - closed, 1 - open]
- reflections [0 - none, 1 - small, 2 - big]
- lighting conditions [0 - bad, 1 - good]
- sensor ID [01 - RealSense, 02- IDS, 03- Aptina]

3.2.2 Preprocessing

To accommodate the different input requirements of the two model architectures used in this experiment, two preprocessing pipelines were created using *torchvision.transforms* library.

- Custom CNN Pipeline
  First the input images were converted to grayscale and resized to a fixed resolution of 64 x 64 pixels. In order to improve the model's generalization capabilities, extensive augmentation was also performed, specifically:
  - Random Horizontal Flip: Applied with a probability of 0.5
  - Random Rotation: Applied with a range of $\pm 15°$
  - Random Affine: Applied with translations up to 10% vertically/horizontally
  - Color Jitter: Brightness and contrast are randomly adjusted with a factor of 0.4
  - Gaussian Blur: Applied with a kernel size of 3x3 with sigma range of (0.1, 2.0) to simulate out of focus frames
  - Gaussian Noise Injection: Add random noise sampled from a normal distribution (mean = 0, std = 0.05)
  - Random Erasing: Applied with a probability of 0.2 with scale range of 0.02 to 0.15
  - Normalization: Applied with a mean of 0.5 and std 0.5
- MobileNetV2 Pipeline
  This pipeline was adjusted to adhere to the input specifications for transfer learning with ImageNet weights. First, we convert the image to RGB and resize it to 224 x 224 pixels. The augmentation techniques applied here were much lighter to preserve the integrity of the pre-learned features:
  - Random Horizontal Flip: Applied with a probability of 0.5
  - Random Rotation: Applied with a range of $\pm 10°$
  - Color Jitter: Brightness, contrast adjusted with a factor of 0.2

- Normalization: Normalized the pixel values using the standard ImageNet values (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

3.2.3 Model Architecture

The custom CNN is a sequential model constructed using *torch.nn* library. It features a feature extractor that progressively deepens followed by a heavily regularized classifier. The architecture details are as follows:

1. First Convolutional Block
   - Conv2d: 32 filters, kernel size 3x3, padding 1
   - Batch Normalization
   - Activation Function: ReLu
   - Regularization: Dropout2d with probability 0.2
   - Pooling: MaxPool2d with 2 x 2 kernel
2. Second Convolutional Block
   - Conv2d: 64 filters, kernel size 3x3, padding 1
   - Batch Normalization: 64 channels
   - Activation Function: ReLu
   - Regularization: Dropout2d with probability 0.3
   - Pooling: MaxPool2d with 2x2 kernel
3. Third Convolutional Block
   - Conv2d: 128 filters, kernel size 3x3, padding 1
   - Batch Normalization: 128 channels
   - Activation Function: ReLu
   - Regularization: Dropout2d with probability 0.4
   - Pooling: MaxPool2d with 2x2 kernel
4. Classifier Head
   - Flatten: Converts 3D feature map (128 x 8 x 8) to 1D vector of size 8,192
   - Dropout: Applied with probability 0.6
   - Fully Connected: Linear layer mapping 8,192 to 128 units
   - Activation Function: ReLu
   - Dropout: Applied with probability 0.5
   - Output Layer: Mapping from 128 to 2 classes (‘Closed’ or ‘Open’)

For the second model, the MobileNetV2 is initialized with IMAGENET1K_V1 weights. Since we want to retain the feature extraction capabilities of the model, we did not make as many changes:

1. Feature Extractor
   Froze the initial layers to prevent weight updates during training, gradients were disabled for the feature blocks so only few inverted residual blocks were allowed to be fine-tuned
2. Classifier Head
   Since MobileNetV2 returns 1280 channels[4] from the backbone, it will be the input layer, while the rest of the structure will be:
   - Dropout: Applied with probability 0.5
   - Linear Layer: Maps 1280 features into 128
   - Normalization: BatchNorm1d
   - Activation function: ReLU
   - Second Dropout: Applied with probability 0.4
   - Output Layer: Maps 128 features to 2 classes (‘Closed’ or ‘Open’)

3.2.4 Training Setup

The optimizer used in this experiment was Adam optimizer, for its adaptive learning rate capabilities which would help stable convergence. Both models also use a Batch Size of 32 because of hardware limitations, number of folds are both 5, and the early stopping patience are both 10. Aside from that, the other hyperparameter configurations for both models can be seen in figure 3.1.

Hyperparameter	**Custom CNN**	**MobileNetV2**
Input Size	64 x 64	224 x 224
Learning Rate	5 x 10-5	1 x 10-5
Weight Decay	1 x 10-4	1 x 10-3
Max Epochs	50	30
Scheduler	ReduceLROnPlateau	CosineAnnealingWarmRestarts

Table 3.1 Hyperparameter Configurations of Both Models

Since CNN is trained from scratch, we designated a higher learning rate so it can escape the initial random state and learn the features, while MobileNet is just finetuning, as such we lower the learning rate, so it does not forget the pre-trained filters. As MobileNet also has far more parameters (ca. 2.2M), we use higher regularization to prevent the large model from overfitting since the dataset is also relatively small. Epochs are also given more to CNN because it takes time to build the feature hierarchies from scratch rather than MobileNet, which only requires more training on the classifier head. For the Scheduler, we also use different ones for different purposes, for CNN we only lower the learning rate when the model is stuck, while for MobileNet we "warm restart" it to help prevent or jump out of a sharp local minimum since we are just basically refining the weights.

3.2.5 Evaluation Metrics

To assess the performance of both models, the following metrics were tested on a test set, consisting of unseen data in the training and validation phases:

- Accuracy: the ratio of correctly predicted eye states to the total predictions, can be misleading so we also used more granular metrics with it
- Recall: measure the model's ability to identify all positive instances (in this case, "Closed" eyes), an important metric since we do not want to flag that subject's eye is open when it is closed. A high number of False Negative is what we tried to avoid.
- Precision: measures the accuracy of positive predictions, False Positive is less dangerous than missing an eye "Closed" state, but it might still be annoying to the user since it produces false alarms.
- F1 - Score: harmonic means of precision and recall, so we can see the score that balances minimizing false alarms (Precision) and missed detections (Recall)
- Inference Time: measured in milliseconds per image, crucial to determine whether the model is ready for deployment in a real-time system

Chapter IV
Implementation and Results

4.1 System Details

The development, training, and evaluation of the proposed models were conducted on a local workstation. The specific hardware and software configurations are detailed below:

4.1.1 Hardware Specifications

The experiments were executed on a system equipped with a CUDA-enabled NVIDIA GPU, which was utilized for accelerating the training of deep learning models and performing inference tasks.

- Processor: 13th Gen Intel(R) Core (TM) i7-13650HX, 2600 Mhz
- Graphics Processing Unit (GPU): NVIDIA GeForce RTX 4060
- Memory (RAM): 24 GB
- Storage: SSD

4.1.2 Software Environment

The system was operated on Windows 11, but Windows 10 should also work. The core implementation was developed using Python 3.10.19 within a Conda virtual environment. The key libraries and frameworks utilized include:

- Deep Learning Framework: PyTorch (with CUDA support) for model architecture, training loops, and tensor operations
- Computer Vision: torchvision for image transformations, augmentations, and loading pre-trained weights (MobileNetV2)
- Data Manipulation: Pandas and NumPy for handling dataset metadata and numerical computations
- Evaluation & Metrics: scikit-learn for generating confusion matrices, classification reports, and managing K-Fold cross-validation splitting
- Visualization: Matplotlib and Seaborn for plotting loss curves and confusion matrices
- Data Acquisition: kagglehub for automated dataset downloading and caching
- Converting to ONNX: onnx and onnxruntime for converting to ONNX for lighter deployment but not needed for model training

4.2 Experiments

To ensure the model generalizes and to prevent information leakage[3], this dataset of 10,000 images, evenly distributed across the 2 classes, 'Open' and 'Closed', is split into a test set and training set. The data is split based on subject ID.

- Test Set: Originally test set gets a 0.15 ratio, which means 3 subjects of the total 19. However, the test images are 2725 images since the datasets are not distributed equally across all subjects, resulting in about 0.2725 ratio for the test set.
- Training/Validation Set: The remaining images were used for training by employing a 5-Fold Cross-Validation strategy grouped by subject ID. In each fold, specific subjects were separated for validation while the remainder were used for training.

This approach generated five distinct performance values for each model, which were averaged to obtain a robust estimate of the model's true capability. This strict separation prevents data leakage, ensuring that the validation performance reflects the model's ability to handle new, unseen individuals rather than memorizing subject-specific features.

4.3 Visualizations and Results

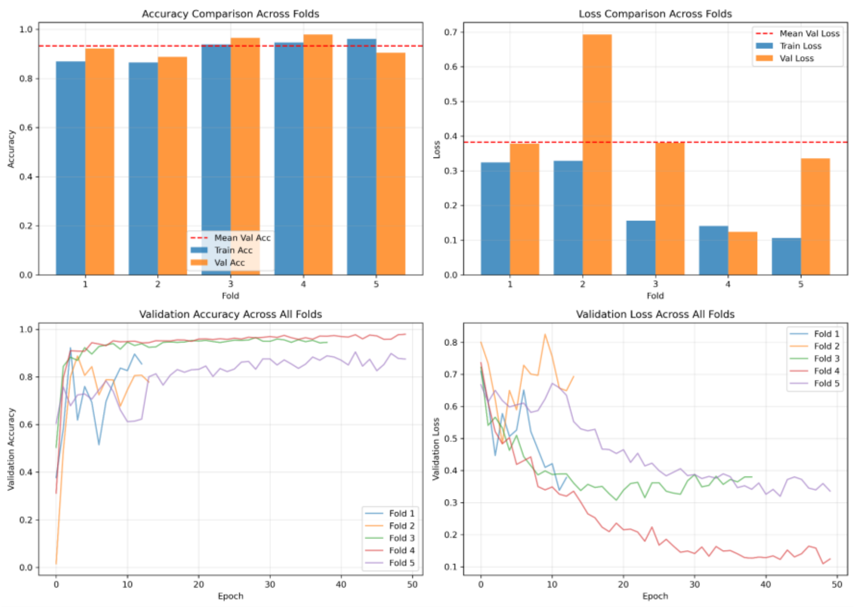

Figure 4.1 Training and Validation Accuracy and Loss Across Folds for Custom CNN

The 5-fold cross-validation results using Custom CNN show a model with highly variable performance depending on the data split, with Fold 4 emerging as the clear winner by achieving the highest accuracy and the lowest, most stable loss curve over 50 epochs. Conversely, Fold 2 is a significant outlier that performed poorly, evidenced by a massive spike in validation loss reaching nearly 0.7 which likely triggered early stopping around epoch 15. Fold 5 also might have issues with overfitting since its validation accuracy remains lower than the others and fluctuates significantly in the bottom charts. This inconsistency implies that while the model architecture is capable of excellent results as seen in Fold 4, it might be too sensitive to specific data variations such as in Folds 2 and 5.

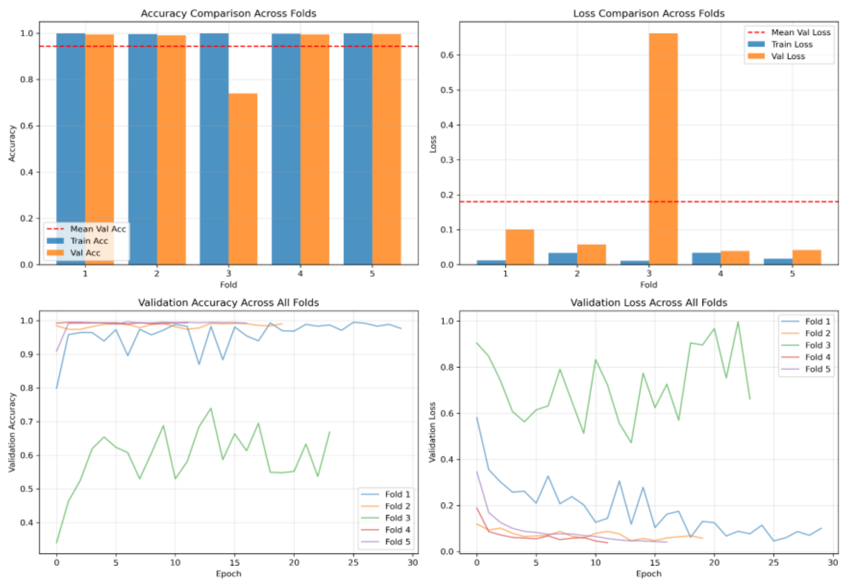

Figure 4.2 Training and Validation Accuracy and Loss Across Folds for MobileNetV2

The 5-fold cross-validation results using MobileNetV2 reveal a “capable” model that is generally performing almost exceptionally well, with Folds 1, 2, 4, and 5 achieving almost perfect alignment between training and validation metrics, having close to 100% accuracy and minimal loss. However, Fold 3 looks to be an anomaly that significantly impacts the overall mean performance. Unlike the others, it suffers from severe overfitting that can be seen by a massive gap where training accuracy stays high, but validation accuracy sharply drops to roughly 74% and loss spikes over 0.6. The bottom-row line charts confirm this instability, as the Fold 3 curves fail to converge and fluctuate wildly, strongly suggesting that this specific data split contains a non-representative sample, class imbalance, or "hard" examples that the model struggles to generalize against compared to the rest of the dataset.

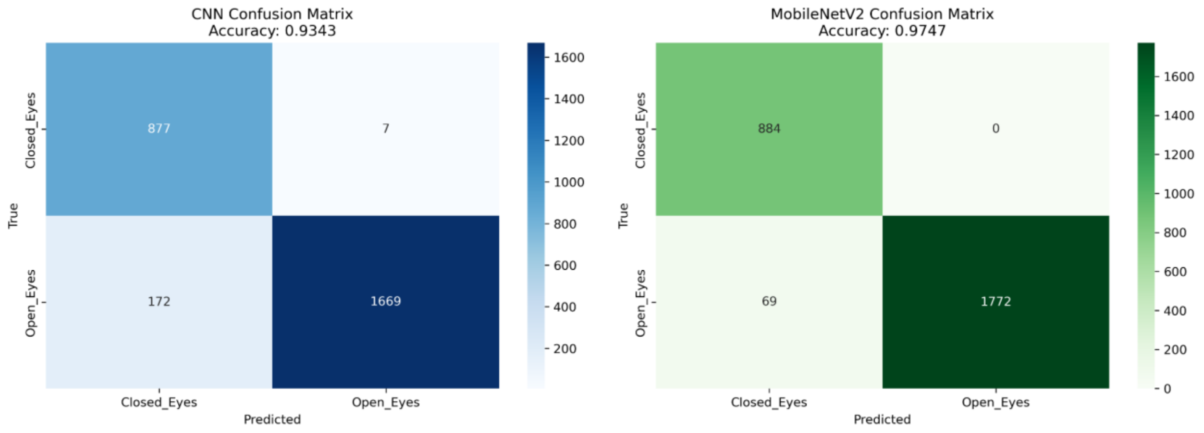

Figure 4.3 Confusion Matrix Comparison Custom CNN Model (Left) and MobileNetV2 (Right)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Total Inference Time (s)**	**Average Time per Image (ms)**
CNN	0.934312	0.943987	0.934312	0.935575	1.467618	0.538575
MobileNet V2	0.974679	0.976512	0.974679	0.974913	5.515333	2.023975

Table 4.1 Model Comparison Summary

From Figure 4.3 and Table 4.1, we can surmise that the comparative analysis between the custom CNN and MobileNetV2 models reveals a tradeoff between classification precision and computational efficiency. MobileNetV2 demonstrates superior predictive performance across all metrics, achieving an accuracy of 97.47% compared to CNN's 93.43%. Most notably, MobileNetV2 exhibited very good results for the "Closed_Eyes" class, eliminating false negatives thoroughly (0 misclassifications, while CNN misclassified 7) and significantly reducing false positives for "Open_Eyes" from 172 to 69. However, this robustness does come with more computational cost, the custom CNN is substantially more efficient, having an average inference time of 0.54 ms per image, approximately four times faster than MobileNetV2's 2.02 ms. Therefore, while CNN is better suited for extremely low-latency environments, MobileNetV2 offers a more reliable and robust solution for accurate eye state detection.

4.4 Real-Time Deployment Simulation

To validate the practical applicability and simulate the models in real-life simulations, a "Drowsiness Safety Monitor" application was developed. The interface is designed to provide real-time feedback on driver fatigue levels using webcam feed, made using vanilla html, css, and js. In this testing, we used the custom CNN model, not the theoretically and tested better MobileNetV2, because the MobileNetV2 did not manage to predict properly at all. It will detect every face as open, but it changes randomly, even the confidence levels between the left and right eye really varies.

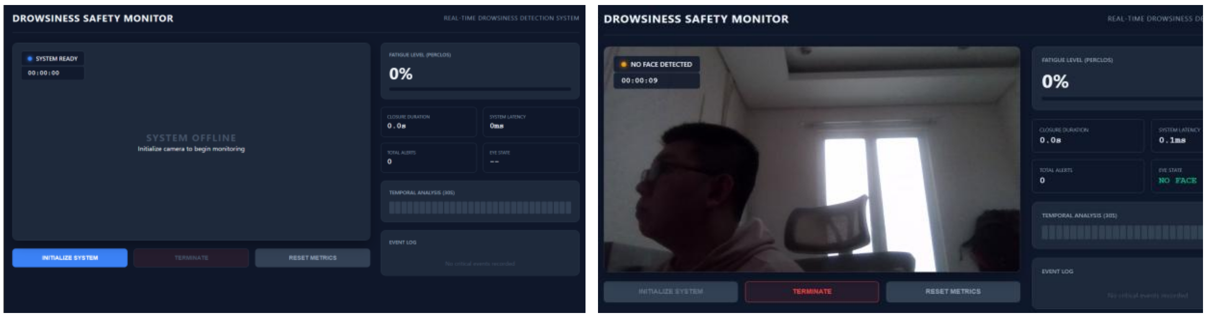

Figure 4.4 Deployment App UI, Not Started (Left) and No Face Detected (Right)

As shown in Figure 4.4 (Left), the system begins in an idle state where the monitor and webcam are offline until the user initializes the camera. Once active, it will try to locate the user's face. Figure 4.4 (Right) demonstrates the system's handling of error, when the user turns away or the face is not detected, the status indicator will turn orange with a "No Face Detected" warning, ensuring that the system does not generate false drowsiness metrics when the driver is not visible. It first detects if a face exists first, then it will predict the state of the eyes.

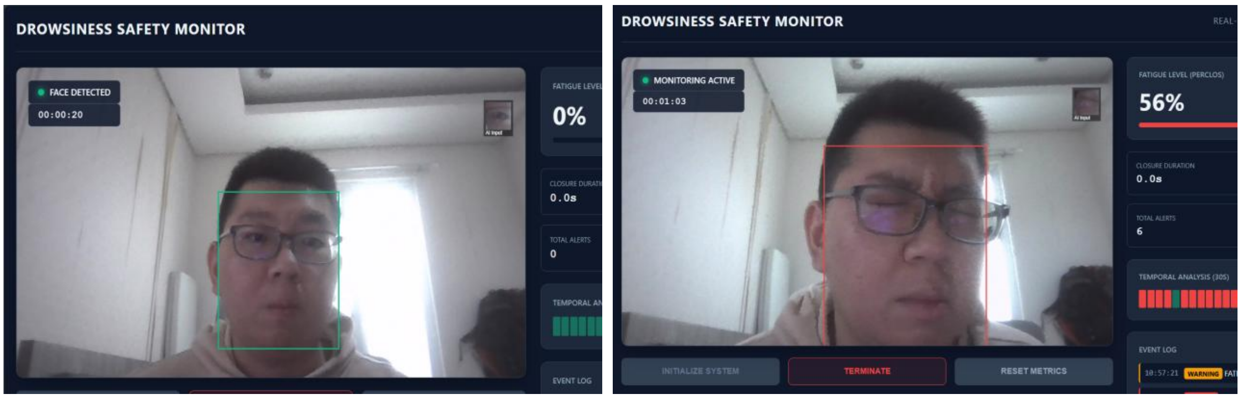

Figure 4.5 Deployment App UI, Eye State Open (Left) and Closed (Right)

Figure 4.5 illustrates the states when a face is detected. A bounding box will be drawn around the user’s face, and its color will depend on the prediction returned by the model, green for “Open” like in the left, and red for “Closed” like in the right. A small box on the top right is also included to properly validate and debug what images are being sent to the model. Not only that, but the prediction results will also be logged to calculate PERCLOS and when it hits certain thresholds, a warning will appear like in the bottom right image.

Chapter V
Discussion and Limitations

5.1 Performance Analysis

Through the quantitative evaluation results, we can see that MobileNetV2 significantly outperforms the custom CNN in classification, with an accuracy of 97.47% compared to CNN's 93.43%. Not only that, for a safety-first application, MobileNetV2 having perfect Recall for "Closed_Eyes" class with 0 false negatives displays its robustness as well as preventing dangerous "misses" when detecting drowsiness. CNN did not do much worse either, having 7 misclassified results for "Closed_Eyes", indicating that it is also functional and might have potential for future improvements. However, the real-life simulation does prove otherwise, that MobileNetV2, despite having better performance results, did not manage to perform well in real-life testing. This might be because the test set really fits the model, showing a better performance but not robust enough for real-life differences, whether it be subject or lighting conditions.

5.2 Challenges

During training and testing, several challenges were identified:

- Head Pose Variations
  As seen in figure 4.4, when the head is tilted to the right, the face would not be identifiable. It is not a problem if the camera is placed directly in front of the user and their eyes can be perceived, but in real life situations the user might not always stay in one place.
- Eyeglasses
  The presence of eyeglasses may introduce glare, reflection, and other factors that might make the model confused whether the eyes are open or closed. In the test set in figure 4.5 our model still managed to detect open or closed eyes through glasses, but sometimes when eyes are fully closed the model can still detect it as open. The real problem is sunglasses since they would almost block the model from functioning properly.
- Lighting Conditions
  In our test set the lighting conditions are quite good, but in real-life scenarios, especially when driving, the lighting may be really dark or bright. The model might not be able to properly detect the eye states in those cases, not to mention if a dark shadow is also cast because of hairstyles or other apparels.

5.3 Trade-offs

For the two models, the primary trade-offs concern inference speed and classification accuracy/safety. Compared to CNN's inference time of 0.54 ms, MobileNetV2's inference time of 2.02ms is almost 4 times it. However, for cases where this will be used, such as driver safety or student drowsiness detection, the better the accuracy and other evaluation metrics, the better. The cost of failing to detect a sleeping driver is far more catastrophic compared to a slightly higher latency. 2.02 ms still allows for theoretical frame rates exceeding standard webcam capabilities of 30-60 FPS. The slight reduction in latency and speed is an acceptable trade-off for the substantial gain in accuracy, precision, recall, etc. 4% difference is very huge for providing safety, but of course we would still need to do more testing, since the model may just be suitable for the test dataset but not on actual real-life conditions.

Chapter VI
Conclusion and Future Work

6.1 Conclusion

This experiment successfully developed and evaluated a real-time drowsiness detection system capable of monitoring fatigue through eye state analysis. The comparative assessment between a custom Convolutional Neural Network (CNN) and the MobileNetV2 architecture highlighted a clear distinction in performance suitability. While the custom CNN offered superior computational speed with an average inference time of 0.54 ms, it lacked the necessary sensitivity for safety critical applications, as can be seen with it misclassifying several closed-eye instances.

In contrast, MobileNetV2 became the optimal architecture theoretically for this system. It achieved a higher overall accuracy of 97.47% and, most importantly, demonstrated perfect recall for the "Closed_Eyes" class. The real time deployment simulation, however, failed to validate the model, since it showed very bad results compared to its test set performances. The CNN model instead predicted very well, despite still occasionally mistaking closed eye states as open. However, the simulation still validated the practical viability of the model, successfully calculating PERCLOS metrics and making alerts appear when fatigue thresholds were exceeded. Ultimately, the system successfully managed to detect drowsiness based on vision. It cannot be said that it is fully usable for driver drowsiness detection since it is still not considering a more variety of factors such as driver head tilt, sunglasses, and more metrics that might not be discovered or tested yet. However, this will serve as a baseline for building a robust model for detecting eye state.

6.2 Future Work

To expand the real-world capabilities of the model and the robustness of the system itself, several areas that can be further developed are as follows:

- Low-Light and Night Vision Support
  Our testing was conducted in well-lit environments, and even though the dataset contains images taken in varying conditions including lighting, our model still could not adapt to it as much as we expected. Future research should consider incorporating datasets featuring Infrared or night vision images, considering we also did not have the resources to test it, to ensure the reliability of the model in different lighting conditions. This is especially important for driver's drowsiness since it frequently occurs when driving at night.
- Head Pose and Gaze Estimation
  The model currently only works if the subject faces frontal to the camera. Especially for driver drowsiness, integrating head pose and gaze estimation would allow the system to further distinguish between whether the driver is doing a safe action (e.g. checking mirrors) or a dangerous one (e.g. head drooping due to fatigue).
- Metrics other than PERCLOS
  Currently our system only judges based on Percentage of Eye Closure (PERCLOS), other metrics could be integrated such as yawn detection, mouth aspect ratio, or other metrics to create a weighted and more robust fatigue score, so it will reduce the likelihood of false positives for example from blinking, though it may need more than that since blinking very often can also be considered a sign of fatigue.

References

[1] World Health Organization. (2018). Global status report on road safety 2018. World Health Organization.

[2] Akerstedt T. (2000). Consensus statement: fatigue and accidents in transport operations. Journal of sleep research, 9(4), 395. https://doi.org/10.1046/j.1365-2869.2000.00228.x

[3] Ji, Q. (2002). Real-Time Eye, Gaze, and Face Pose Tracking for Monitoring Driver Vigilance. Real-Time Imaging, 8(5), 357–377. https://doi.org/10.1006/rtim.2002.0279

[4] Sahayadhas, A., Sundaraj, K., & Murugappan, M. (2012). Detecting Driver Drowsiness Based on Sensors: A Review. Sensors, 12(12), 16937-16953. https://doi.org/10.3390/s121216937

[5] Liu, C. C., Hosking, S. G., & Lenné, M. G. (2009). Predicting driver drowsiness using vehicle measures: recent insights and future challenges. Journal of safety research, 40(4), 239–245. https://doi.org/10.1016/j.jsr.2009.04.005

[6] Lal, S. K. L., & Craig, A. (2001). A critical review of the psychophysiology of driver fatigue. Biological Psychology, 55(3), 173–194. https://doi.org/10.1016/s0301-0511(00)00085-5

[7] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 1. https://doi.org/10.1109/cvpr.2001.990517

[8] BioID GmbH. (2022, November 18). BioID Face Database. BioID. https://www.bioid.com/face-database/

[9] Putri, N. A. (2025). Exploration of techniques for converting color images to grayscale through the implementation of the luminosity method for digital image processing. In Proceedings of the International Conference on Islamic Community Studies. Universitas Pembangunan Panca Budi. https://proceeding.pancabudi.ac.id/index.php/ICIE/article/view/740

[10] Song, J., Kim, D., Jeong, E., & Park, J. (2025). Determination of Optimal Dataset Characteristics for Improving YOLO Performance in Agricultural Object Detection. Agriculture, 15(7), 731. https://doi.org/10.3390/agriculture15070731

[11] Prasad V Patil, MRL Eye Dataset, Kaggle, 2023. Available: https://www.kaggle.com/datasets/prasadvpatil/mrl-dataset

[12] MRL Eye Dataset | MRL. (2020). Cs.vsb.cz. https://mrl.cs.vsb.cz/eyedataset.html

[13] Srivastava, N., Hinton, G., Krizhevsky, A., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929–1958. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b4

[14] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/cvpr.2018.00474

Appendix

Team Contribution Statement

Coordinated through WhatsApp group chat

	Bertrand Geraldo	Jose Christian	Reinhart Gian	Yudhistira
Idea	v	v	v	v
Detector Training				v
Classifiers Training			v	
Model Deployment		v	v	
Report	v	v	v	v
Video				v

Table A.1 Team Contribution Distribution

GitHub Repository

https://github.com/reinhartgw/vision-based-drowsiness-detection

Demonstration Video

https://drive.google.com/drive/folders/1fyCIvz_KhnXbo-MUcTJd0Ouv8YfZB6X9