Deep Learning Course Project

Project Title:Seatbelt Detection using YOLO and DenseNet

Submitted by: Tamer Kobba,Abdullah Lababidi
Student ID(s): 202104873, 202104092
CSC462: Fundamentals of Deep Learning
Instructor's Name: Seifedine Kadry

Abstract

The "Seatbelt Detection" project implements an automated system for detecting seatbelt usage using YOLO (You Only Look Once) for object detection and DenseNet for classification. It processes video frames to identify windshields, crop them, and classify whether a person is wearing a seatbelt or not.

Project Repository: The code and resources to recreate and use this project are available on GitHub. Access the repository here: https://github.com/Tamerkobba/Seatbelt_detection

1 Project Proposal

1.1 Objective

In recent times the number of fatal traffic accidents has increased worldwide, so the proper utilization of seatbelts is more vital than ever eventhough wearing a seatbelt is a mandatory traffic regulation alot of individuals don't abide by these laws,so it has become imperative to catch perpetrates to ensure general safety. We propose a deep learning solution to provide an automated program to solve this solution.

1.2 Dataset

The dataset utilized for this project comprises multiple sources of images depicting individuals driving with and without seatbelts. Although several datasets were identified, none were pre-labeled to meet the specific requirements of this study. Consequently, a significant portion of the images required manual annotation to define bounding boxes for object detection. Additionally, a supplementary dataset was created to focus explicitly on the classification task of determining seatbelt usage.

A portion of the images was cropped from the original dataset if it appeared that the seatbelt status (worn or not worn) could be observed and they made up around 3000 images of our classification dataset. Preprocessing steps included resizing images, normalizing pixel values, and augmenting the data to improve model robustness and generalization. Ultimately, two datasets were prepared: one for the object detection task and the other for the classification task.

1.3 Expected Challenges

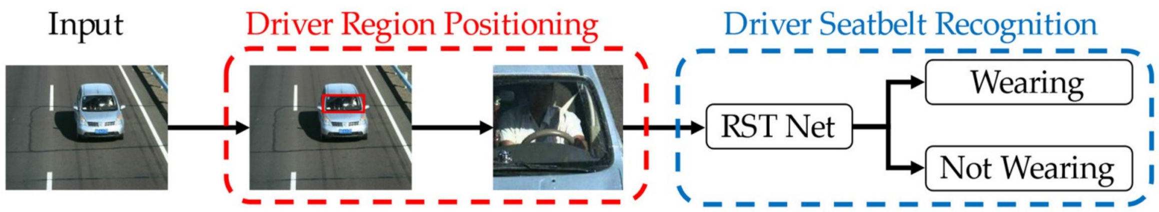
One of the primary challenges anticipated in this project is the limited availability of datasets suitable for the object detection model. Specifically, there is a scarcity of publicly available datasets featuring images taken from a sufficient distance with labeled seatbelt usage. Most existing datasets primarily include close-up images labeled for seatbelt status (i.e., "wearing" or "not wearing"), which are inadequate for a robust object detection task.

To address this, we adopted a two-dataset approach inspired by the methodology outlined in Qiu et al. (2024) [1]. This approach involves:

1. **Windshield Detection Dataset**: A dataset curated for detecting windshields in vehicle images at various distances, annotated with bounding boxes around the windshield regions.

2. **Driver Seatbelt Classification Dataset**: A separate dataset of close-up images of drivers annotated for seatbelt usage, enabling classification of seatbelt status.

Proposed Workflow Diagram:



- Step 1: Detect vehicles and localize the windshield using bounding box annotations.
- Step 2: Crop the windshield area and apply classification to determine seatbelt usage.

2 Data Preparation and Preprocessing

2.1 Data Cleaning

2.1.1 YOLO

No data cleaning was required for YOLO as the images were carefully selected and manually annotated, ensuring high-quality labels from the start.

2.1.2 DenseNet

For DenseNet, while the images were also manually annotated, some images were accidentally labeled twice. These duplicate labels were identified and dropped to ensure the dataset's integrity.

2.2 Data Augmentation

2.2.1 YOLO

Data augmentation was applied to enhance the diversity of the training data. The augmentation techniques included:

- **Random Horizontal Flip**: To simulate different viewing angles.
- **Random Vertical Flip**: To add variability for vertically symmetrical objects.
- **Random Rotation**: Up to 15 degrees to account for tilted object perspectives.
- **Color Jitter**: To adjust brightness, contrast, saturation, and hue, mimicking real-world lighting conditions.
- **Random Resized Crop**: To ensure objects appear in varying scales and positions.

2.2.2 DenseNet

DenseNet used a similar augmentation strategy to prevent overfitting and improve generalization:

- **Resizing to 224x224**: To match the input size expected by the DenseNet architecture.
- **Random Horizontal Flip**: To handle orientation variations.
- **Random Rotation (15°)**: To simulate tilted views.
- **Color Jitter**: To adjust image brightness, contrast, and saturation, increasing diversity.

2.3 Normalization/Standardization

2.3.1 YOLO

For YOLO, normalization involved rescaling pixel values to the range [0, 1] by dividing by 255. This ensured uniform input for the model, which improves numerical stability and speeds up convergence.

2.3.2 DenseNet

For DenseNet, standardization was applied by normalizing images using the ImageNet dataset's mean and standard deviation:

- **Mean**: [0.485, 0.456, 0.406]
- **Standard Deviation**: [0.229, 0.224, 0.225]

This standardization ensures compatibility with DenseNet's pretrained weights and enhances performance.

2.4 Data Splitting

2.4.1 YOLO

The dataset for YOLO was split into training, validation, and test sets based on the following proportions:

- **Training Split**: 82.38% (10,251 images)
- **Validation Split**: 12.05% (1,499 images)
- **Test Split**: 5.53% (688 images)

2.4.2 DenseNet

For DenseNet, the dataset was split into 80% training and 20% validation subsets. No separate test set was used at this stage, as validation was performed iteratively:

- **Training Split**: 80% of the dataset
- **Validation Split**: 20% of the dataset

3 Model Architecture and Design

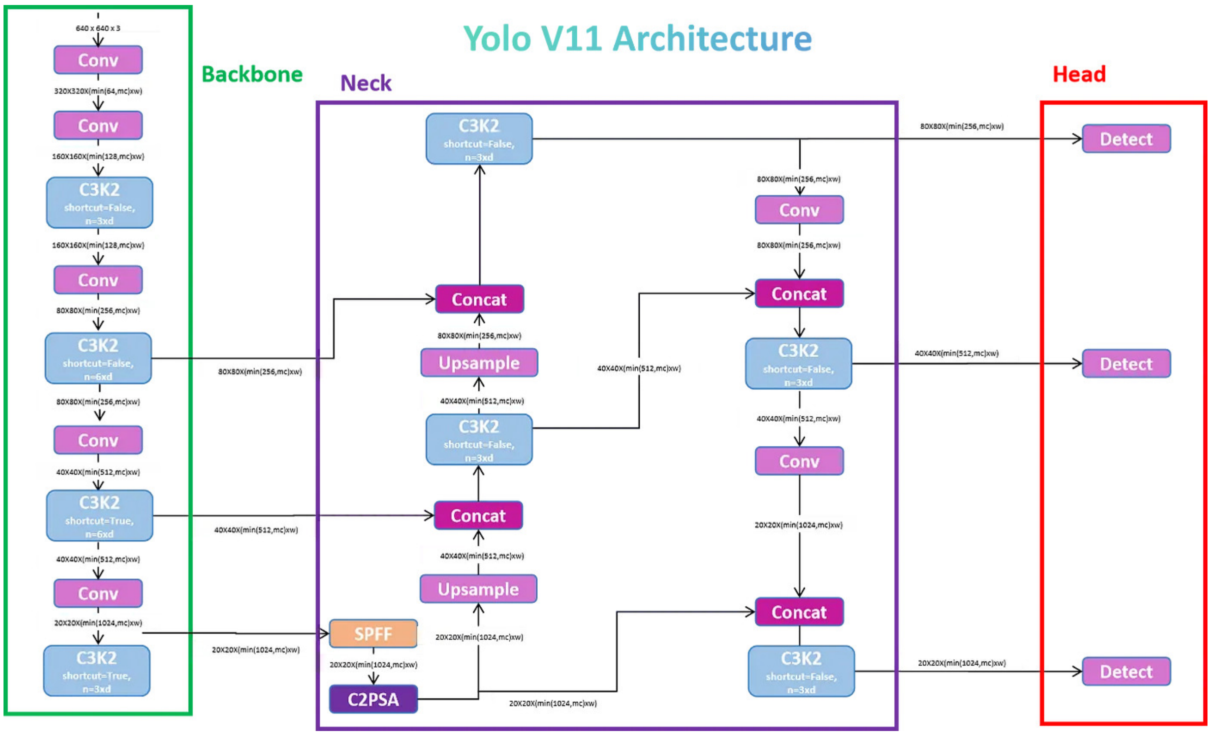
3.1 Model Choice

For this project, YOLO11n and DenseNet121 were selected as the primary architectures due to their efficiency and suitability for the task.

3.2 Layer Design

The YOLO11n includes:

Layer Type	Key Features	Purpose
Convolutional	Filters (3 × 3), Batch Norm, SiLU: $f(x) = x \cdot \sigma(x)$	Extract spatial features and hierarchies.
SPPF	Multi-scale max pooling (5 × 5, 9 × 9, 13 × 13)	Aggregate multi-scale information for small objects.
C3K2 Block	CSP Bottleneck + Concat + Smaller Kernels (3 × 3)	Efficient feature extraction with fewer parameters.
Attention (C2PSA)	Spatial Attention + FFNN	Focus on critical image regions, enhance detection accuracy.
Upsampling	Bilinear Interpolation	Increase resolution of feature maps for multi-scale prediction.
Fully Connected (FFNN)	Position-Sensitive Attention + Linear Activation	Refine spatial attention features.
Final Convolutional Layers	Bounding Box, Objectness, Class Scores	Generate object detection predictions.



The YOLOv11n architecture introduces several key innovations to enhance detection accuracy and efficiency. Below are detailed explanations of the key architectural components and their significance:

- **C3K2 Block:** The C3K2 block is a computationally efficient variant of the Cross Stage Partial (CSP) Bottleneck. By replacing larger convolutions with smaller kernels, this block significantly reduces the computational overhead while maintaining high feature extraction capabilities. Its integration throughout the backbone, neck, and head enhances multi-scale feature representation, critical for detecting small objects [2].

- **SPPF Block:** The Spatial Pyramid Pooling - Fast (SPPF) block aggregates multi-scale spatial information using max pooling at different kernel sizes (5×5, 9×9, 13×13). This enables YOLOv11n to detect objects of varying sizes and orientations with improved precision [2].

- **C2PSA Block:** The Cross Stage Partial with Spatial Attention (C2PSA) block introduces advanced spatial attention mechanisms. It enables the model to focus more effectively on critical regions within an image, which is particularly beneficial for identifying small or occluded objects. This mechanism enhances detection accuracy and ensures that key features are retained during feature map processing [2].

- **Attention Mechanisms:** By incorporating attention mechanisms like the C2PSA block, YOLOv11 demonstrates an improved ability to process and highlight relevant image regions. This results in a significant boost in performance for complex detection scenarios, such as objects with varying scales or partial occlusions [2].

- **Upsampling and Final Convolutional Layers:** The use of bilinear interpolation for upsampling ensures that multi-scale features are preserved without introducing unnecessary artifacts. The final convolutional layers generate bounding box coordinates, objectness scores, and class predictions, completing the object detection pipeline [2].

These architectural advancements make YOLOv11n a highly efficient and accurate object detection model, suitable for real-time applications in resource-constrained environments. Compared to its predecessors, YOLOv11n achieves a superior balance between computational efficiency and detection accuracy [2].

3.3 Layer Design (continued)

DenseNet121 includes:

Layers	Output Size	DenseNet-121 Design
Convolution	112 × 112	7 × 7 conv, stride 2
Pooling	56 × 56	3 × 3 max pool, stride 2
Dense Block (1)	56 × 56	Bottleneck [1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	28 × 28	1 × 1 conv, 2 × 2 average pool, stride 2
Dense Block (2)	28 × 28	Bottleneck [1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	14 × 14	1 × 1 conv, 2 × 2 average pool, stride 2
Dense Block (3)	14 × 14	Bottleneck [1 × 1 conv, 3 × 3 conv] × 24
Transition Layer (3)	7 × 7	1 × 1 conv, 2 × 2 average pool, stride 2
Dense Block (4)	7 × 7	Bottleneck [1 × 1 conv, 3 × 3 conv] × 16
Classification Layer	1 × 1	7 × 7 global average pool, 1000D fully-connected, softmax

Table 1: Detailed architecture of DenseNet-121.

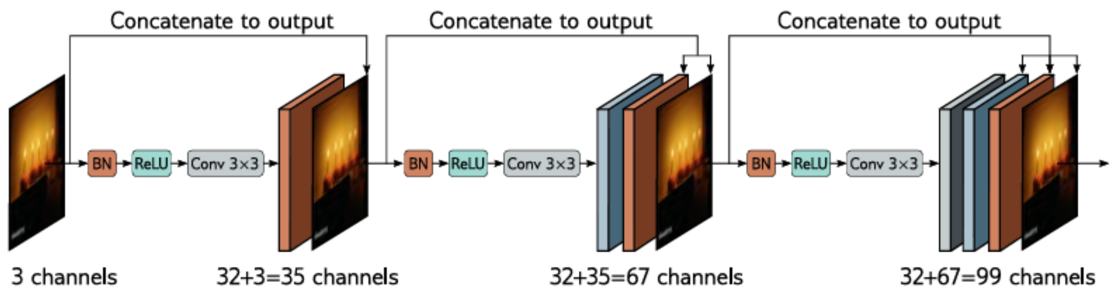


Figure 1: DenseNet-121 architecture showing convolution, pooling, bottleneck, and dense blocks.

Explanation of Bottleneck Layers: Each dense block in DenseNet-121 employs bottleneck layers consisting of a 1×1 convolution followed by a 3×3 convolution. The 1×1 convolution reduces the number of input feature maps, thereby decreasing computational overhead and improving parameter efficiency. This design significantly contributes to DenseNet's compactness and computational efficiency while maintaining its high representational power.

Advantages of Dense Connectivity: DenseNet's design ensures that each layer has direct access to the feature maps of all preceding layers, which promotes feature reuse and alleviates the vanishing gradient problem. This dense connectivity pattern also enables implicit deep supervision, allowing gradients to flow more effectively during backpropagation, facilitating the training of very deep networks [3].

Transition Layers: To maintain computational efficiency and prevent feature map explosion, transition layers are introduced between dense blocks. These layers consist of a 1×1 convolution followed by 2×2 average pooling, reducing the spatial dimensions and the number of feature maps [3].

DenseNet-121's innovative architecture achieves a balance between computational efficiency and representational power, making it a powerful choice for image classification tasks. Compared to ResNet, DenseNet-121 has fewer parameters while delivering competitive performance on large-scale datasets like ImageNet [3].

3.4 Model Complexity

YOLO11n's streamlined architecture ensures rapid inference, essential for real-time detection scenarios. DenseNet121's design promotes efficient feature utilization, leading to high classification accuracy without excessive computational demands.

3.5 Pretrained Models

Leveraging pretrained models is crucial for tasks where data availability and task complexity creates significant challenges. Approximately 12,000 images were available for windshield detection, and 10,000 images for seatbelt classification

Seatbelt Detection: This task is exceptionally challenging due to the non-trivial nature of the conditions under which seatbelt usage must be identified. The complexities include:

- Varying Lighting Conditions
- Obstructions and Reflections
- Camera Angles
- Driver Postures and Clothing
- Weather Conditions

To mitigate these challenges, DenseNet121, pretrained on large-scale datasets, was fine-tuned for binary classification. Its dense connectivity and efficient feature reuse allowed the model to handle these complexities effectively.

Windshield Detection: YOLO11n, a lightweight object detection model, pretrained on a robust object detection dataset, was fine-tuned to detect windshields. The use of a pretrained model enabled the detection to be efficient and reliable, even with variations in the dataset.

4 Training Process

4.1 Training Strategy

The training process utilized a transfer learning strategy by leveraging a pre-trained DenseNet-121 model initialized with the `DenseNet121_Weights.IMAGENET1K_V1` weights. The final classifier layer was modified to output predictions for two classes. This approach benefits from the pre-trained model's capability to extract rich and generalized features, significantly reducing the required training time and data while maintaining high accuracy.

Additionally, a custom YOLO model (`yolo11n.pt`) was trained using the same data, demonstrating the integration of object detection with image classification tasks. The DenseNet-121 model was trained for 50 epochs, while YOLO was trained for 120 epochs with a batch size of 8 and an image size of 640. The training strategy also incorporated learning rate scheduling to optimize the convergence process.

4.2 Loss Function and Optimization

The **loss function** used for the DenseNet-121 model was the **cross-entropy loss**, implemented using PyTorch's `torch.nn.CrossEntropyLoss` The formula is given as:

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{\exp(z_{i,y_i})}{\sum_{c=1}^{C}\exp(z_{i,c})}\right),$$

where:

- N is the total number of samples,
- C is the number of classes,
- $z_{i,c}$ is the unnormalized logit (raw score) for class c of sample i,
- y_i is the ground truth class index for sample i.

Additional Details:

- PyTorch's `CrossEntropyLoss` expects the input logits $z_{i,c}$ rather than probabilities, as it computes the softmax internally for numerical stability.
- The target y_i is provided as class indices (not one-hot encoded vectors).

For optimization, **Stochastic Gradient Descent (SGD)** was employed with momentum, updating the parameters ϕ_t and momentum m_t at each step as:

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\sum_{i\in\mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1},$$

where:

- β is the momentum coefficient (set to 0.7),
- α is the learning rate,
- $\mathcal{B}_t$ is the mini-batch at iteration t,
- $\ell_i[\phi_t]$ is the loss for sample i in the mini-batch,
- $\frac{\partial \ell_i[\phi_t]}{\partial \phi}$ is the gradient of the loss with respect to the parameters.

Weight decay (L2 regularization) was applied to prevent overfitting by penalizing large weight values. The regularized loss function is defined as:

$$\mathcal{L}_{reg} = \mathcal{L} + \lambda \|\phi\|_2^2,$$

where $\lambda = 0.001$ is the regularization coefficient.

The learning rate was dynamically adjusted using the **OneCycleLR** scheduler, allowing the learning rate to vary as:

$$\eta(t) = \{ \eta_{start} + (\eta_{max} - \eta_{start}) \cdot f(t), if t \leq t_{mid}, \eta_{max} + (\eta_{end} - \eta_{max}) \cdot f(t - t_{mid}), if t > t_{mid},$$

For YOLO, the built-in multi-task loss function combines:

$$\mathcal{L}_{YOLO} = \mathcal{L}_{box} + \mathcal{L}_{obj} + \mathcal{L}_{cls},$$

where:

- $\mathcal{L}_{box}$ penalizes errors in bounding box predictions,
- $\mathcal{L}_{obj}$ penalizes incorrect objectness scores,
- $\mathcal{L}_{cls}$ penalizes classification errors.

4.3 Hyperparameter Tuning

The hyperparameters tuned included:

- **Learning rate** (η): Fixed at 0.0001 for DenseNet-121 and dynamically adjusted by the OneCycleLR scheduler. For YOLO, the framework's default learning rate was used.
- **Batch size**: Set to 8 for both models to balance memory usage and gradient stability.
- **Number of epochs**: Set to 50 for DenseNet-121 and 120 for YOLO.
- **Momentum** (μ): Set to 0.7 for SGD in DenseNet-121 to stabilize updates.
- **Weight decay** (λ): Set to 0.001 to prevent overfitting by penalizing large weights.

While no explicit grid or random search was performed, these values were chosen based on prior best practices.

4.4 Regularization

To prevent overfitting, the following regularization techniques were applied:

- **Weight Decay (L2 Regularization)**: Added a term $\lambda \|\theta\|_2^2$ to the loss function, ensuring smaller weights and better generalization.
- **Pre-trained Weights**: DenseNet-121 was initialized with ImageNet pre-trained weights, enabling robust feature extraction and reducing overfitting.
- **Learning Rate Scheduling**: The OneCycleLR scheduler prevented over-adjustments in weights by controlling the learning rate dynamically.
- **Validation Monitoring**: Validation loss was monitored at every epoch, and the best model was saved based on the lowest validation loss, ensuring the preservation of the optimal model.

5 Evaluation and Analysis

5.1 Evaluation Metrics

5.1.1 yolov11:

For the evaluation of the YOLO11 model, several key metrics were used:

- **Precision and Recall**: Precision measures the percentage of true positive detections among all positive detections, while recall evaluates the percentage of true positives among all actual positive instances.
- **F1-Score**: Combines precision and recall into a single score by taking their harmonic mean.
- **Mean Average Precision (mAP)**: Evaluates the model's ability to make accurate predictions across various confidence thresholds.
- **Loss Metrics**: Box loss, classification loss, and objectness loss were tracked for both training and validation phases.

5.1.2 DenseNet:

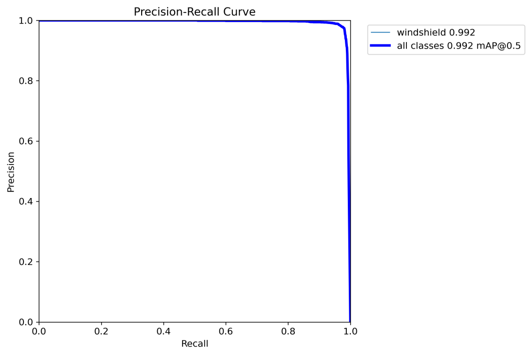
For evaluating the DenseNet model used to classify seatbelt violations, the following metrics were employed:

- **Accuracy**: Represents the proportion of correctly classified instances among the total number of instances.
- **Precision**: Evaluates the ratio of true positive predictions to the total positive predictions made by the model.
- **Recall**: Measures the proportion of true positives that were correctly identified by the model. This is critical for minimizing false negatives.
- **F1-Score**: Combines precision and recall into a harmonic mean, balancing these two metrics.

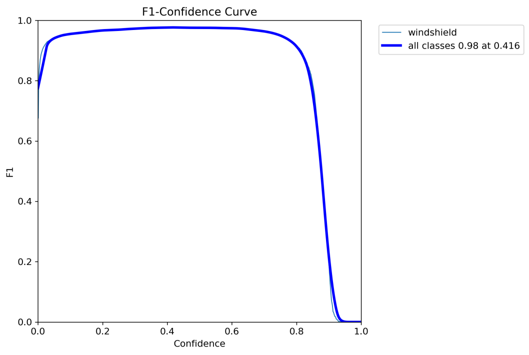
5.2 Validation and Test Results

subsubsectionYOLOv11: The validation and test results, as depicted in the figures, demonstrated the following:

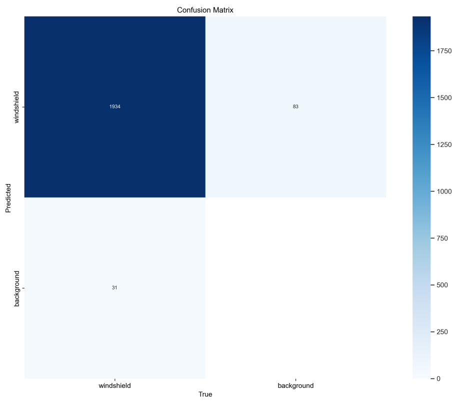
- **Validation Set:**
  - The model achieved a precision of 0.992 and a recall of 0.99 for the "windshield" class, as shown in the Precision-Recall curve (Figure 2a).
  - The F1-confidence curve (Figure 2b) peaked at 0.98, confirming an optimal balance between precision and recall at a confidence threshold of 0.416.
  - Confusion matrices (Figures 2c and 2d) revealed strong performance with low misclassification rates.
  - Box loss and classification loss steadily decreased during training and validation, as observed in Figure 5.



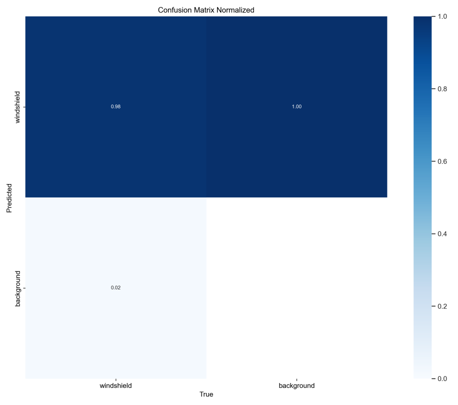
(a) Precision-Recall Curve



(b) F1-Confidence Curve



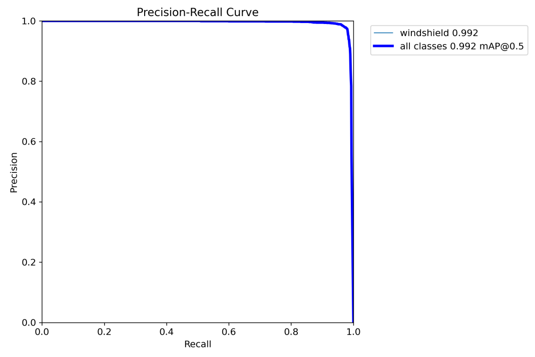
(c) Confusion Matrix



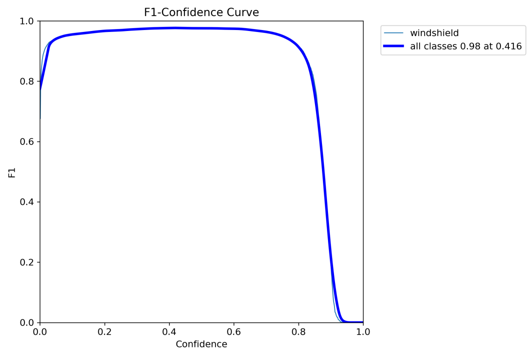
(d) Normalized Confusion Matrix

Figure 2: Validation Figures for YOLOv11

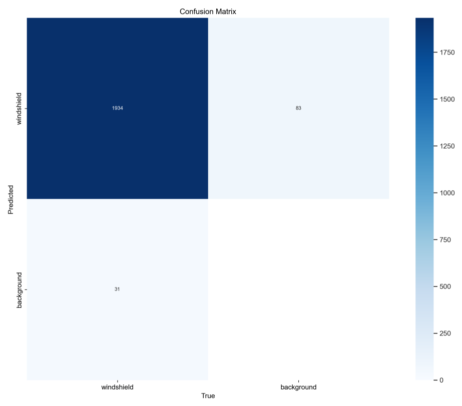
- **Test Set:**
  - The model maintained a high performance with a precision of 0.992 and a recall of 0.99, as indicated in the Precision-Recall curve (Figure 3a).
  - The F1-confidence curve (Figure 3b) peaked at 0.98, confirming robustness across datasets.
  - Confusion matrices (Figures 3c and 3d) for the test set demonstrated similar trends to the validation set, with high true positive rates and minimal misclassification.
  - Visual predictions on test samples are illustrated in Figure 4, showcasing effective object detection and classification of windshields.



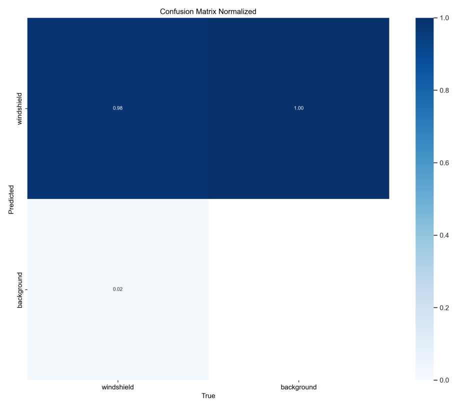
(a) Precision-Recall Curve



(b) F1-Confidence Curve



(c) Confusion Matrix



(d) Normalized Confusion Matrix

Figure 3: Test Figures for YOLOv11

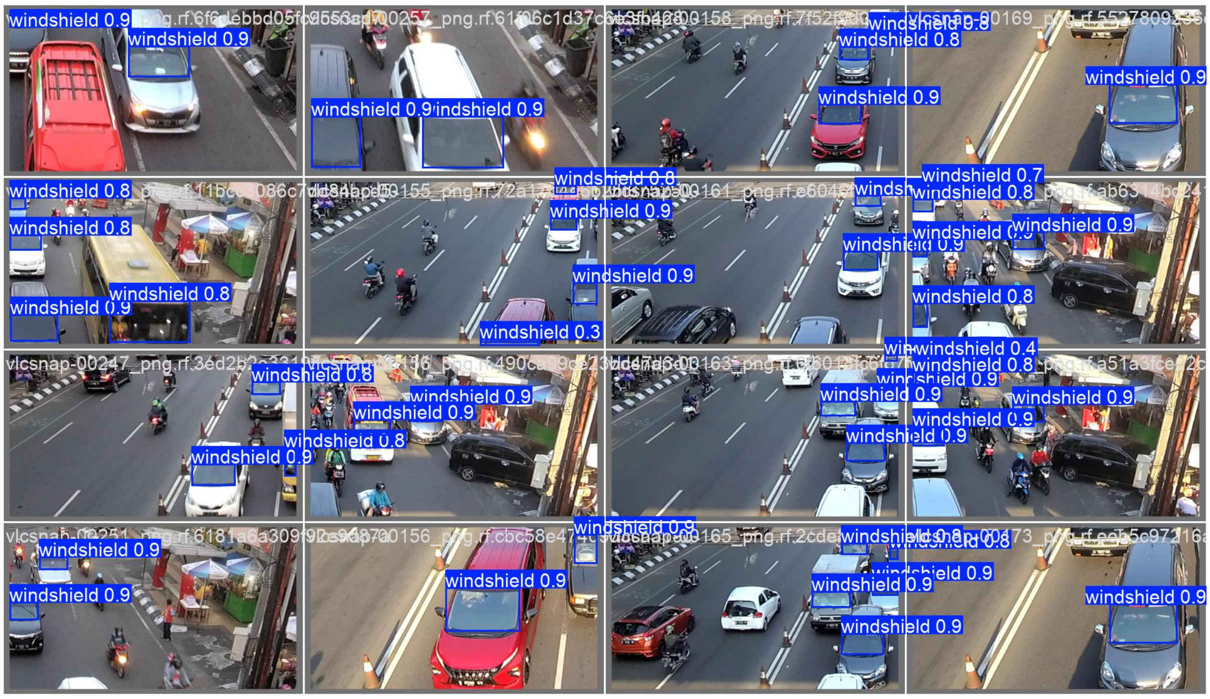


Figure 4: Visualization on test samples

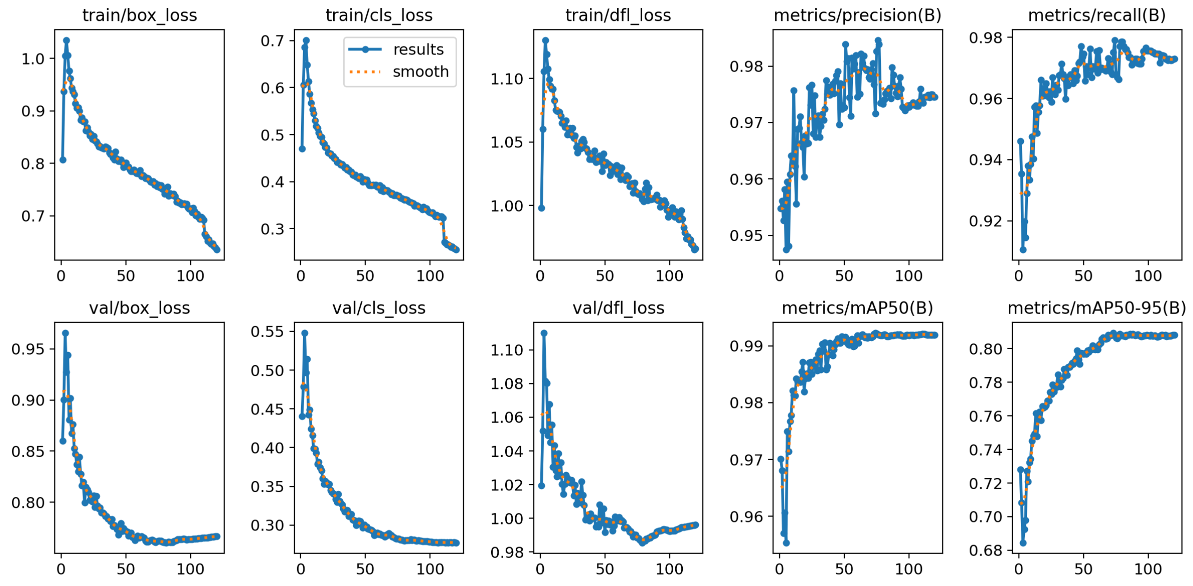


Figure 5: Training and Validation Loss Metrics for YOLOv11

5.2.1 DenseNet:

The DenseNet model achieved the following results:

- **Training Metrics:**
  - Accuracy: 0.9556
  - Precision: 0.9556
  - Recall: 0.9556
  - F1-Score: 0.9556
- **Validation Metrics:**
  - Accuracy: 0.9558
  - Precision: 0.9558
  - Recall: 0.9558
  - F1-Score: 0.9558

The evaluation across different thresholds revealed the following insights (Figure 6):

- Increasing the threshold leads to higher precision but reduces recall, indicating the model becomes more conservative in its predictions.
- At a threshold of 0.5, the F1-score reached its peak (0.9585), balancing precision (0.9590) and recall (0.9580).
- Lowering the threshold to 0.3 results in a recall of 0.9834 while maintaining a relatively high F1-score of 0.9605. This is crucial for minimizing false negatives, as a false negative (classifying a violation as non-violation) could lead to detrimental outcomes.

5.2.2 Justification and Recommendations

Given the critical nature of minimizing false negatives in detecting seatbelt violations (i.e., ensuring violations are not missed), recall should be prioritized. A slightly lower precision is acceptable as false positives (e.g., flagging a compliant driver as violating) can be manually verified and dismissed without major consequences.

To optimize for recall, a threshold of 0.3 is recommended. At this threshold:

- Recall is maximized at 0.9834, significantly reducing false negatives.
- The F1-score remains high at 0.9605, ensuring balanced performance.
- Precision of 0.9385 is acceptable for this application, given the ease of manual verification for false positives.

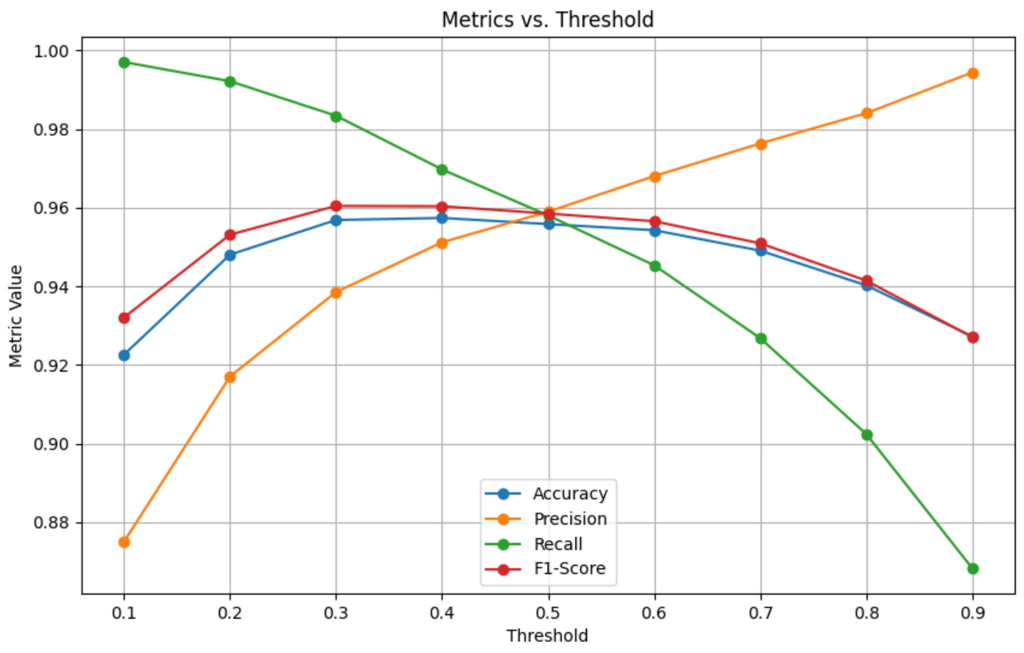


Figure 6: Metrics vs. Threshold for DenseNet Model

5.3 Error Analysis:

The YOLO model struggles to detect windshields in scenarios involving extreme darkness or excessive windshield glare. Similarly, even when the windshield is successfully localized, cropped, and fed into the DenseNet model, it may encounter the same issues, leading to classification errors. Another typical error would be the case if we get different angles of the cars and nearby building the windows would be thought of windshield, most of the time this isn't an issue since a camera would be needed to be setup in a fixed position to capture the car at a certain angle every time so it would be rare to run into this problem.

5.4 Extra: Computation Speed

Initially, we did not expect the model to process real-time video efficiently due to the computational demands of DenseNet121, a relatively large convolutional neural network (CNN). Optimizations such as quantization and model pruning were anticipated to be necessary to improve inference speed while maintaining accuracy [4]. For real-time performance at 30 FPS, the processing time per frame must not exceed 0.0333 seconds, and for 60 FPS, it must stay below 0.0167 seconds.

Surprisingly, our pipeline achieved an average processing time of 0.0149 seconds per frame, corresponding to approximately 67 FPS. This exceeds the requirements for both 30 FPS and 60 FPS video, demonstrating that DenseNet121 can meet real-time performance demands without immediate optimization.

The model's current speed is sufficient for real-time applications, contrary to initial expectations. While further optimizations like quantization or pruning [4] could enhance scalability for higher frame rates or resource-constrained environments, they are not strictly necessary for the current implementation.

6 Insights and Observations

Throughout the development and evaluation of the seatbelt detection system, several key insights and patterns emerged:

6.1 Key Findings

- **Computation Speed:** The system processed frames at an average of 67 FPS, surpassing the real-time requirements of 30 FPS and 60 FPS without the need for additional optimizations like pruning or quantization.
- **Error Patterns:** Common errors stemmed from extreme lighting conditions, such as glare or darkness, and occasional misclassification of building windows as windshields, highlighting the importance of consistent camera positioning.

7 Implications for Problem-Solving

- **Safety Enhancement:** Automating seatbelt detection ensures better compliance with traffic laws, contributing to reduced fatalities and improved road safety.
- **Scalability and Deployment:** Integrating the system into smart city infrastructure and testing its scalability across different urban environments and camera setups would provide valuable insights for large-scale adoption.

8 Conclusion and Future Directions

8.1 Summary of Contributions

This project successfully developed an automated seatbelt detection system leveraging YOLOv11 for object detection and DenseNet121 for classification. Key contributions include:

- **High-Performing Models:** YOLOv11 achieved a precision of 0.992 and a recall of 0.99 for windshield detection, while DenseNet121 demonstrated an F1-score of 0.9605 for seatbelt classification.
- **Real-Time Efficiency:** The pipeline achieved an average processing speed of 67 FPS, exceeding the requirements for real-time applications without requiring optimization techniques like quantization or pruning.
- The project highlighted the importance of fixed camera setups and robust data preprocessing to mitigate common errors like glare, extreme darkness, and misclassification of building windows.

8.2 Limitations

Despite its success, the project faced several limitations:

- **Limited Dataset Size:** The datasets used were relatively small compared to the scale required for detecting nuanced variations in real-world scenarios. Such tasks often rely on datasets curated by government or large organizations with access to massive amounts of labeled data.
- **Lighting and Reflection Challenges:** Extreme glare, darkness, and reflections occasionally led to errors in both detection and classification, highlighting the need for a more diverse dataset and advanced preprocessing techniques.
- **Generalization Across Environments:** The fixed camera assumption limits the system's ability to generalize across varying setups, such as different angles or dynamic camera positions.

8.3 Future Work

To enhance the project, several realistic improvements and alternative methods are proposed:

- **Dataset Expansion:** Collaborations with government agencies or organizations to access larger, more diverse datasets would significantly improve model generalization and robustness.
- **Unified Model Design:** Instead of a two-step pipeline, a singular, larger YOLO model could be trained to perform both windshield detection and seatbelt classification, simplifying the workflow and potentially increasing performance.
- **Lightweight Classification Models:** Exploring smaller, more efficient models for classification, such as MobileNet or EfficientNet, could reduce computational overhead while maintaining accuracy.
- **Advanced Preprocessing Techniques:** Implementing glare reduction, low-light enhancement, or image segmentation methods could address lighting challenges and improve overall performance.

The results affirm the feasibility of using advanced deep learning models like YOLO and DenseNet for real-world safety-critical applications, offering both efficiency and scalability for future deployment. Addressing the discussed limitations and incorporating the suggested improvements will enhance the system's robustness and impact, contributing to safer roads and better compliance with traffic regulations.

References

[1] Qiu, L.; Rao, J.; Zhao, X. Seatbelt Detection Algorithm Improved with Lightweight Approach and Attention Mechanism. *Applied Sciences*, **2024**, *14*, 3346. https://doi.org/10.3390/app14083346

[2] Khanam, R.; Hussain, M. *YOLOv11: An Overview of the Key Architectural Enhancements*. Huddersfield University, 2024. arXiv preprint arXiv:2410.17725v1.

[3] Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K. Q. Densely Connected Convolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. arXiv preprint arXiv:1608.06993v5.

[4] Han, S.; Mao, H.; Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016. arXiv preprint arXiv:1510.00149v5.