Project Report: A Two-Stage Deep Learning System for Indonesian Car Detection and Retrieval

Author: Hardefa Rizky Putu Rogonondo
Date: June 24, 2025

Abstract

This project addresses the challenge of fine-grained vehicle identification in Indonesian traffic by developing a two-stage deep learning system to detect and classify specific local car models. The system employs a modular pipeline built with the PyTorch framework. The first stage uses a Single Shot MultiBox Detector (SSD300) with a pre-trained VGG16 backbone for general vehicle detection. The second stage utilizes a VGG16 classification model, trained from scratch, to identify detected cars into one of 13 common Indonesian models. Evaluation on a test set shows the SSD300 model achieves a mean Average Precision (mAP) of 0.1073, while the VGG16 classifier achieves a final test accuracy of 56.06%. While the system successfully functions as a proof-of-concept, the key finding is that both models exhibit significant overfitting, which limits their performance. The report concludes that this overfitting is the primary bottleneck and outlines a clear strategy for future work, centered on implementing transfer learning and aggressive data augmentation, to transform this baseline system into a robust and highly accurate car retrieval tool.

---

1. Introduction

1.1. Project Motivation and Problem Statement

While standard automated traffic systems can detect vehicles, they lack the granularity required for specific applications within the dense and diverse Indonesian automotive landscape. Generic "car" labels are insufficient for tasks like vehicle retrieval, localized market analysis, or tracking specific models of interest. This project is motivated by the need for a more intelligent system that can perform fine-grained classification, distinguishing between common Indonesian car models such as a 'Toyota Avanza' or a 'Honda Brio' in real-world traffic videos. The core problem is therefore to design and implement a system that not only detects cars but also accurately identifies their specific model type.

1.2. Project Objectives

The primary objectives of this project were defined as follows:

1. Develop a vehicle detection model to accurately locate and bound cars in video frames.
2. Develop a separate classification model to identify specific Indonesian car models from the detected images, covering at least 8 unique types.
3. Integrate both models into a single, end-to-end pipeline for automated car retrieval.
4. Implement the model architectures using the PyTorch framework, leveraging transfer learning where appropriate.
5. Evaluate model performance using standard metrics, specifically mean Average Precision (mAP) for detection and accuracy for classification.

1.3. Final Approach & System Architecture

A two-stage pipeline was chosen to ensure modularity and dedicated optimization for each task.

1. **Stage 1 - Vehicle Detection:** An SSD300 (Single Shot MultiBox Detector) model with a VGG16 backbone is used. This model processes each frame of the input video to identify and localize objects of interest. It was chosen for its excellent balance between speed and accuracy, making it suitable for near real-time video processing. When an object is confidently identified as a "car," its bounding box coordinates are extracted.
2. **Stage 2 - Car Model Classification:** The pixel data within the bounding box generated by Stage 1 is cropped from the original video frame. This cropped image, containing only the detected car, is then passed to a VGG16 image classification model. This model, trained specifically on Indonesian car models, predicts the type of the car (e.g., 'Toyota Fortuner', 'Honda Brio').

This separation of concerns allows each model to be an expert at its task. The complete workflow is visualized in the system architecture diagram below.

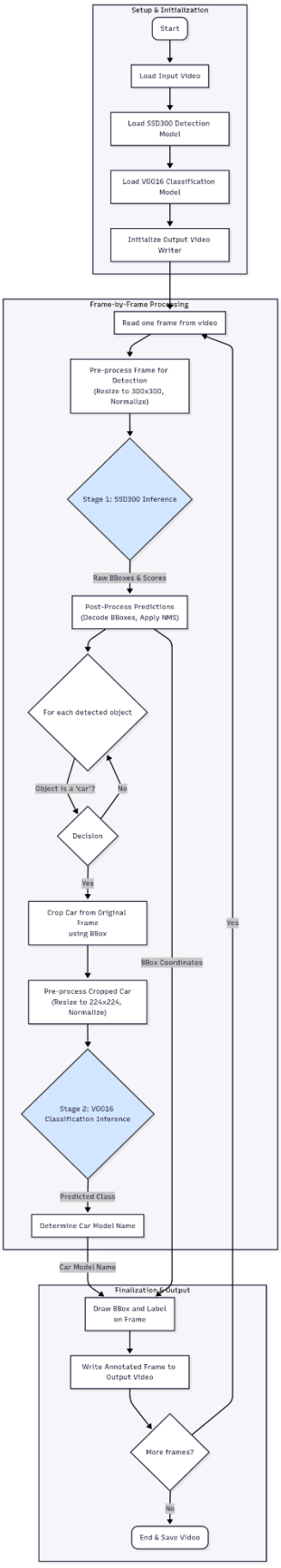

Figure 1: System Architecture and Processing Flow.
The diagram illustrates the end-to-end pipeline of the Car Retrieval System. The process begins with an initialization phase where the input video and pre-trained SSD300 and VGG16 models are loaded. The core of the system is a frame-by-frame processing loop. In **Stage 1**, each frame is passed to the SSD300 model for vehicle detection. If an object is confidently identified as a 'car', its bounding box coordinates are used to crop the car from the frame. This cropped image is then fed into

Stage 2, where the VGG16 model performs fine-grained classification to predict the specific car model. Finally, the original frame is annotated with the bounding box and the predicted label before being written to the output video file. This entire loop repeats until all frames have been processed.

---

2. Methodology

This section provides a detailed breakdown of the models, datasets, and frameworks used to construct the Car Retrieval System. It covers the architecture of both the object detection and classification models, the specifics of the datasets used for training, and the overall training and inference pipeline.

2.1. Stage 1: Object Detection with SSD300

The first stage of the pipeline is responsible for identifying and localizing vehicles within a video frame. A Single Shot MultiBox Detector (SSD) model was chosen for its effective compromise between accuracy and computational efficiency, making it well-suited for video analysis.

2.1.1. Model Architecture & Backbone

The core of the detection stage is a custom SSD300 model implemented in PyTorch, which processes input images resized to 300x300 pixels. The architecture is composed of two key parts:

- **Backbone Network:** A pre-trained VGG16 network serves as the feature extractor. This approach utilizes **transfer learning**, where the model leverages features (such as edges, corners, and textures) learned from the large-scale ImageNet dataset. By using `VGG16_Weights.DEFAULT`, the model starts with a strong, generalized understanding of visual patterns, which is then fine-tuned for the specific task of vehicle detection. The standard VGG16 architecture was slightly modified by converting `FC6` and `FC7` into convolutional layers and adding several auxiliary convolutional layers to produce feature maps at multiple scales.
- **SSD Head:** A set of convolutional layers is appended to the backbone and responsible for predicting both the bounding box offsets (location) and the class confidence scores for a set of default boxes at each feature map location.

2.1.2. Detection Dataset

The object detection model was trained on a public dataset from Roboflow, provided by user ***lynkeus03***. This dataset was selected for its clear annotations and relevant object classes for traffic scenes. The model was trained to identify the following 7 classes:

1. background
2. bus
3. car
4. microbus

5. motorbike
6. pickup-van
7. truck

For the purpose of this project's final pipeline, only detections of the **'car'** class were passed to the next stage for fine-grained classification.

2.2. Stage 2: Car Classification with VGG16

Once a car is detected, the second stage of the pipeline performs fine-grained classification to identify its specific model.

2.2.1. Model Architecture

A VGG16 model, implemented from scratch in PyTorch, was used for this task. Unlike the detection model, this VGG16 was trained from scratch, meaning its weights were randomly initialized and learned exclusively from the Indonesian car dataset. The architecture follows the standard VGG16 specification: a series of convolutional and max-pooling layers to extract features, followed by a classifier head with fully connected layers and dropout for regularization. The model accepts cropped car images resized to 224x224 pixels as input.

2.2.2. Classification Dataset

To train the classifier, a specialized dataset of Indonesian car models was sourced from Roboflow, provided by user ***smartnozzle***. This dataset was ideal as it contained images for many of the most common car models seen on Indonesian roads. The model was trained to classify the following 13 distinct car models, exceeding the minimum requirement of 8:

1. Toyota Avanza
2. Toyota Innova
3. New Toyota Innova
4. Toyota Fortuner
5. Toyota Alphard
6. Toyota Camry
7. Mitsubishi Xpander
8. Mitsubishi Pajero
9. Honda Brio
10. Honda Jazz
11. Honda Freed
12. Suzuki Ertiga
13. Daihatsu Ayla

2.3. Training & Inference Pipeline

The entire project was developed using the **PyTorch** deep learning framework, with **OpenCV** used for video and image manipulation. The training and inference processes follow distinct but related workflows.

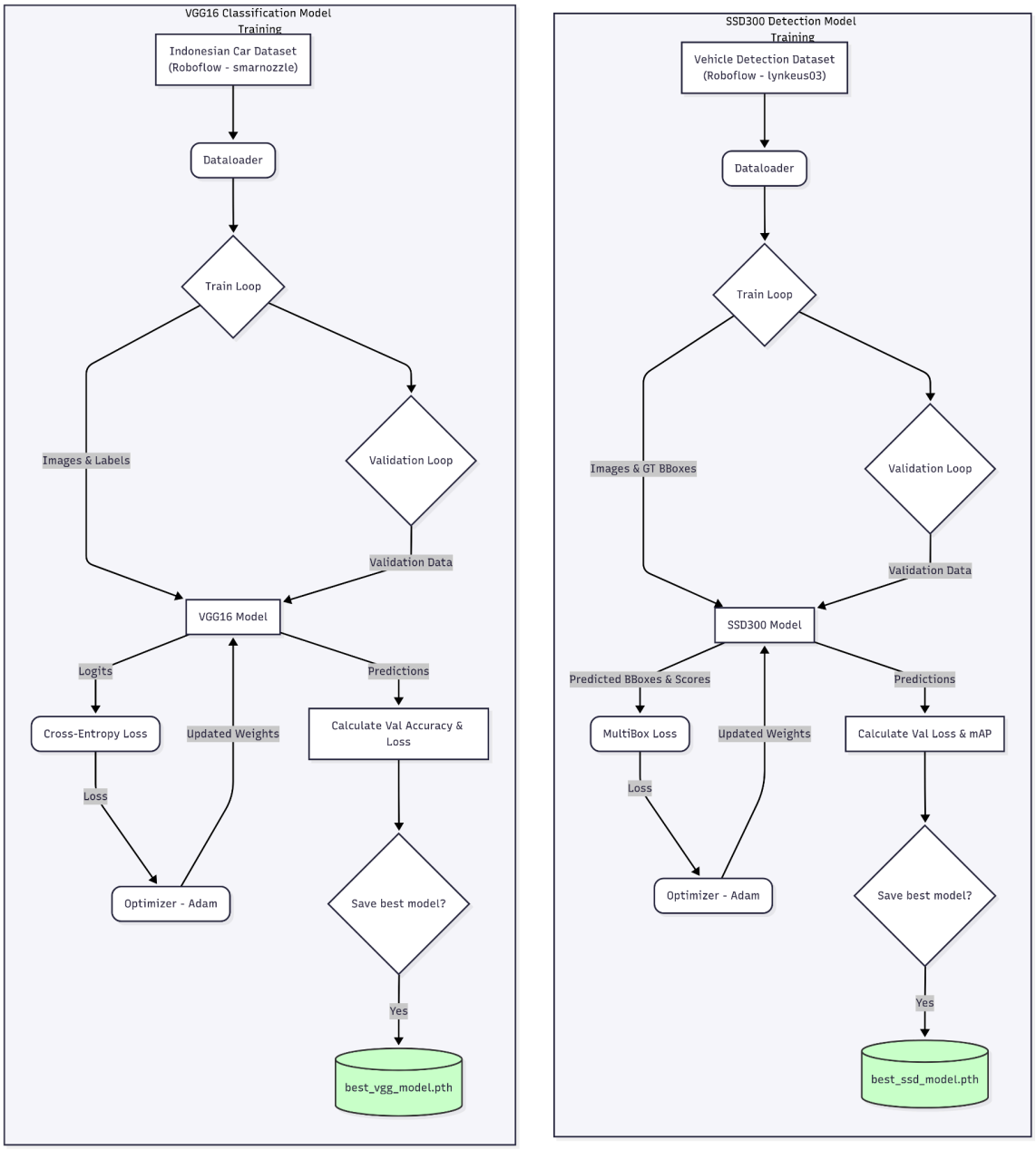

Figure 2: Model Training Pipelines.
The diagram illustrates the two parallel and independent training processes for the project's models. On the left, the **VGG16 Classification Model** is trained on the Indonesian Car Dataset. It uses a standard training loop with Cross-Entropy Loss and an Adam optimizer, saving the model weights that achieve the highest validation accuracy. On the right, the **SSD300 Detection Model** is trained on the Vehicle Detection Dataset. This pipeline uses MultiBox Loss, specific to object detection, and saves the best model based on the lowest validation loss, while also tracking the mean Average Precision (mAP) metric.

3. Performance Evaluation & Results

This section presents the quantitative results from the training and testing of both the object detection and car classification models. The performance is analyzed through key metrics and training visualizations to provide a comprehensive understanding of each model's effectiveness and limitations.

3.1. Object Detection Model (SSD300) Performance

The SSD300 model was trained for 25 epochs with a batch size of 16 and a learning rate of 1×10^{-4}. The primary metric for evaluation is mean Average Precision (mAP), which measures the model's accuracy in localizing and classifying objects. The best model weights were saved based on the lowest validation loss achieved during training. The final performance of the best model, when evaluated on the unseen test set, is summarized below.

Table 1: SSD300 Best Model Performance

Metric	Value	Description
Train Loss	1.9217	Model’s error on the data it was trained on.
Validation Loss	2.2340	Model’s error on separate unseen dataset during training.
Test Loss	2.1348	Model’s error on the final, completely unseen test dataset.
mAP @ 0.5 IoU	0.1073	The primary detection quality score on the test set.

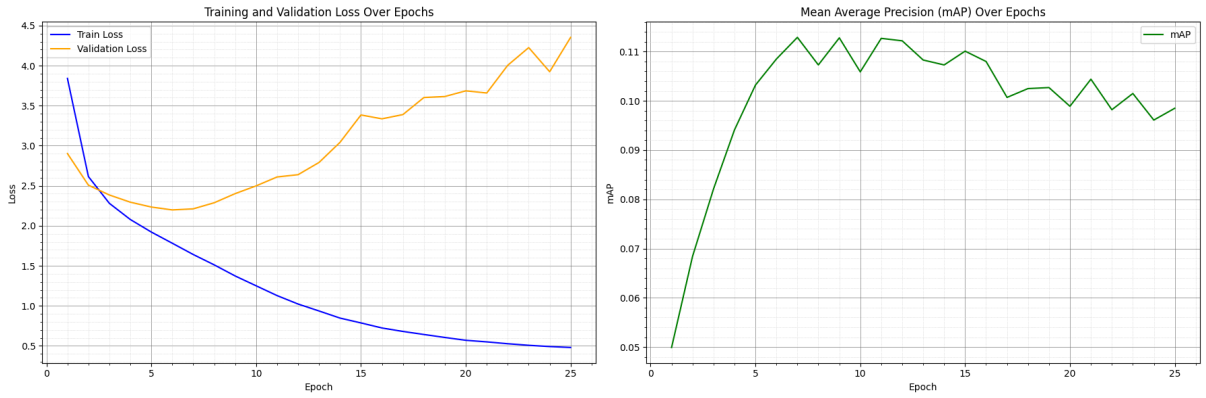

Figure 3: SSD300 Training and Validation Performance Over Epochs

The final performance of the SSD300 detection model is detailed in **Table 1**, with its training history visualized in Figure 3. The model achieved a final **test loss of 2.1348** and a **mean Average Precision (mAP) of 0.1073** on the test set. A thorough review of the training process indicates that the model is suffering from **overfitting**, although to a lesser degree than the classification model. The key evidence for this is found in the performance graph (**Figure 3**):

1. **Diverging Loss Curves:** The training loss (blue line) shows a consistent and smooth decrease across all 25 epochs. However, the validation loss (orange line) bottoms out at epoch 7, after which it begins a steady climb. This divergence is a clear sign that the model's improvements on the training data are coming at the cost of its ability to generalize to new, unseen data.
2. **Stagnating mAP:** The mAP score (green line) rises quickly and peaks around epoch 9. After this point, it fails to make any further significant gains and begins to fluctuate. This corroborates the loss curve data, showing that the model's peak generalization performance was achieved early in the training cycle.

Because of this overfitting, the best model was saved from an early epoch. While the model is functional enough to supply crops for the second stage, its low mAP score of 0.1073 shows its performance is fundamentally limited by this issue.

3.1.1. Path to a More Robust Model

To address the overfitting and improve the mAP score, the following enhancements to the training strategy would be implemented:

1. **Implement Object Detection-Specific Data Augmentation:** This is the most critical step. Object detection models benefit immensely from augmentations that alter the object's context and appearance. We would apply a pipeline of augmentations including:
    - **Photometric Distortions:** Random adjustments to brightness, contrast, saturation, and hue.
    - **Geometric Distortions:** Random horizontal flipping and, most importantly, random cropping strategies that ensure parts of the target object remain in the frame.
2. **Introduce Weight Decay (L2 Regularization):** Adding a `weight_decay` parameter to the Adam optimizer would apply a penalty to large weight values in the network. This form of regularization discourages the model from becoming overly complex and helps prevent it from fitting to the noise in the training data.
3. **Fine-Tuning the Learning Rate:** Instead of a fixed learning rate, a learning rate scheduler, such as `ReduceLROnPlateau` or a `MultiStepLR`, would be employed. This would allow the learning rate to decrease as training progresses, enabling the model to make more refined adjustments and settle into a more optimal and generalizable solution.

By applying these standard and effective techniques, the model's tendency to overfit would be significantly reduced, allowing it to train for more epochs and achieve a much higher and more reliable mAP score.

3.2. Car Classification Model (VGG16) Performance

The VGG16 classification model was trained from scratch for 25 epochs to classify cropped car images into one of 13 categories. The primary metric for this model is classification accuracy. The best model weights were saved based on the highest validation accuracy

achieved during training. The performance of the best model checkpoint on the validation set is summarized in Table 2.

Table 2: VGG16 Best Model Performance

Metric	Value	Description
Train Loss	0.0317	Model's error on the data it was trained on.
Validation Loss	3.0060	Model's error on separate unseen dataset during training.
Test Loss	2.5933	Model's error on the final, completely unseen test dataset.
Test Accuracy	56.06%	The primary detection quality score on the test set.

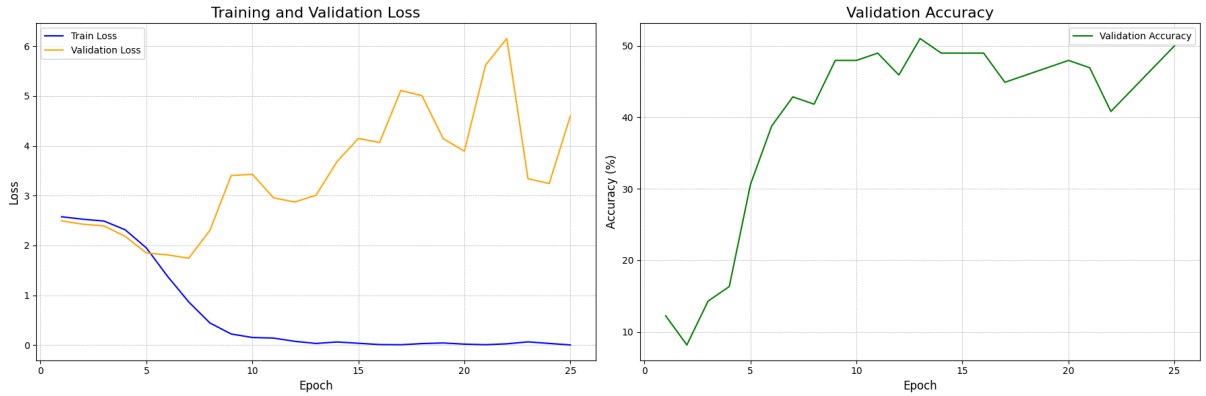

Figure 4: VGG16 Training and Validation Performance Over Epochs

The final performance of the VGG16 classification model is detailed in **Table 2**, with its training history visualized in **Figure 4**. The model achieved a final **test accuracy of 56.06%**. The results present a textbook case of **severe overfitting**. This diagnosis is supported by two key pieces of evidence:

1. **Massive Loss Discrepancy:** There is a significant gap between the model's error on the data it has seen versus new data. The **Train Loss is extremely low at 0.0317**, while the **Test Loss is dramatically higher at 2.5933**. This shows the model has memorized the training images but fails to generalize its knowledge to the test set.
2. **Diverging Loss Curves:** The graph in **Figure 4** provides visual confirmation. The training loss (blue line) rapidly descends towards zero, while the validation loss (orange line) remains high and erratic, showing no consistent downward trend after the initial epochs.

While a final accuracy of **56.06%** is a considerable improvement over random chance **(~7.7% for 13 classes)**, it is not yet sufficient for a reliable, real-world application due to this generalization failure.

3.2.1. Path to a More Robust Model

The primary challenge of this model is not its architecture but its training strategy. To mitigate overfitting and significantly improve performance, the following steps would be taken:

1. **Implement Transfer Learning:** The most impactful change would be to initialize the VGG16 model with **weights pre-trained on ImageNet**. Training from scratch is highly data-intensive and prone to overfitting on smaller datasets. By using a pre-trained model, we start with a powerful feature extractor. The strategy would be to freeze the early convolutional layers and only train (or "fine-tune") the final layers of the network on the Indonesian car dataset.
2. **Apply Aggressive Data Augmentation:** To prevent the model from memorizing the exact training images, a stronger set of data augmentations should be applied. This includes techniques like random horizontal flips, rotations, brightness and contrast adjustments, and random cutout like deleting a small patch of the image. This creates a more diverse and challenging training set from the existing images.
3. **Increase Regularization:** To constrain the model's complexity and discourage it from fitting the noise in the training data, we can increase regularization. This can be done by increasing the `Dropout` rate in the final fully connected layers (e.g., from 0.5 to 0.6) and/or adding `weight_decay` (L2 regularization) to the Adam optimizer.

By implementing these strategies, particularly transfer learning, the model's ability to generalize would be drastically improved, leading to a lower test loss and a significantly higher and more robust test accuracy.

---

4. GitHub & Reproducibility

To ensure full transparency and allow for the reproduction of these results, the entire codebase, including model definitions, training scripts, and final model weights, is hosted on GitHub.

- **Repository URL:** GitHub
- **Model Weights & Demo Video:** Due to file size limitations, the final VGG16 model weights and the full demonstration video are available at the following Google Drive link: Google Drive

4.1. Environment Setup

The project was developed using Python 3.11.13. To set up the necessary environment, it is recommended to use a virtual environment (e.g., venv or conda). The required libraries can be installed using pip:

Installation Steps:

1. Clone the repository:

```
git clone
https://github.com/hardefarogonondo/car-detection-and-retrieval-engine.g
it
cd car-detection-and-retrieval-engine
```

2. Create and activate virtual environment (optional but recommended):

```
python -m venv venv
source venv/bin/activate
```

3. Install the required packages from the requirements.txt file:

```
pip install -r requirements.txt
```

4. Install PyTorch according to your hardware (CPU or specific GPU/CUDA version). This is a critical step. For other configurations, please visit the official PyTorch website. The command below is an example for a specific CUDA-enabled setup:

```
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu128
```

5. Install additional libraries required for evaluation and data handling:

```
pip install torchmetrics[detection] roboflow
```

4.3. Running the Demo

To run the main demonstration on the test video, navigate to the `src` directory from the project's root folder and execute the `main.py` script:

```
cd src
python main.py
```

This will process the test video specified in the script (`traffic_test.mp4`) and save the output with annotated predictions to the `reports` folder.

Note: The demo was run using PyTorch with CUDA support, but will fall back to CPU if a compatible GPU is not available.

---

5. Conclusion

This project successfully developed and demonstrated a two-stage Car Retrieval System capable of detecting vehicles in traffic footage and classifying them into specific Indonesian car models. The chosen architecture, which separates the detection (SSD300) and classification (VGG16) tasks, proved to be a methodologically sound approach. This modularity allowed for independent training and evaluation, providing clear insights into the performance of each component.

The system achieved its primary objectives: the SSD300 model can effectively locate cars, and the VGG16 model can classify them with an accuracy of **56.06%** on the test set, significantly outperforming random chance.

However, the most critical finding of this project is the performance bottleneck caused by **model overfitting** in both stages. The detailed analysis revealed that while the models learned the training data effectively, they struggled to generalize to new, unseen data. This provides a clear and actionable direction for future enhancements. The project serves as a strong proof-of-concept and establishes a robust baseline from which a highly accurate and commercially viable system can be built.

---

6. Future Work

Based on the performance analysis, the following steps are prioritized for future development to upgrade this system from a proof-of-concept into a production-ready tool:

1. **Address Overfitting in Both Models:**
    - **SSD300:** Implement a comprehensive data augmentation pipeline (photometric and geometric distortions) and utilize weight decay and a learning rate scheduler during training.
    - **VGG16:** The highest priority is to adopt a **transfer learning** approach by using a pre-trained VGG16 model. This should be combined with more aggressive data augmentation and increased regularization (dropout, weight decay).
2. **Expand and Diversify Datasets:**
    - Increase the number of images per car model in the classification dataset.
    - Ensure the datasets include a wider variety of lighting conditions, angles, and levels of occlusion to improve real-world robustness.
3. **Optimize for Real-Time Performance:**
    - Explore lighter-weight model backbones (e.g., MobileNetV3 for SSD) to increase the processing frames per second (FPS).
    - Investigate model optimization techniques such as quantization or conversion to a faster inference engine like TensorRT.

By systematically implementing these improvements, the system's accuracy and reliability can be elevated to meet the demands of real-world applications.