Technical Report on Indonesian Car Retrieval System

Lhuqita Fazry
lhuqita.fazry@gmail.com

1 Executive Summary

This report details an automated vehicle detection and classification system specifically engineered for Indonesian traffic analysis. Our pipeline integrates a Faster R-CNN [Ren et al., 2015] detection model with an EfficientNet [Tan and Le, 2019] classifier, designed to categorize vehicles into eight distinct types: Bajaj, Box, Bus, Hatchback, MPV/SUV, Minibus, Pickup-Truck, and Sedan. Table 1 summarizes the vehicle taxonomy used in this project. We combined MPV and SUV into a single category, acknowledging the challenges of differentiating them in typical small-scale traffic light imagery. Despite being trained on limited datasets, the system demonstrates promising performance across these vehicle types.

Table 1: Vehicle Class Taxonomy with Index

No.	Class Name	Description
1	Bajaj	Three-wheeled auto-rickshaw common in Indonesian urban transport
2	Box	Enclosed commercial vehicles (e.g., delivery vans, box trucks)
3	Bus	Full-size transit buses with passenger capacity > 30
4	Hatchback	Compact cars with rear door that opens upwards (e.g., Toyota Agya)
5	MPV/SUV	Multi-purpose vehicles and sport utility vehicles
6	Minibus	Small buses with passenger capacity < 20 (e.g., Toyota Hiace)
7	Pickup-Truck	Light trucks with open cargo area (e.g., Mitsubishi Triton)
8	Sedan	Conventional passenger cars with separate trunk (e.g., Honda Civic)

Initial evaluations reveal strong detection capabilities for specific vehicle classes that typically represent physically larger vehicles. For instance, the detector achieved 53.97% Average Precision (AP) for MPV/SUVs and 42.71% AP for Buses, contributing to an overall 22.91% AP50 across all categories. Analyzing performance by object scale (based on pixel area), the system notably exhibits greater effectiveness in identifying medium-sized objects, achieving 23.40% AP, compared to 11.56% AP for objects categorized as "large" by pixel area. This suggests that while classes like MPV/SUVs and Buses are well-detected, many of their instances, along with other common vehicle types, frequently appear within the 'medium' pixel-size range in the captured imagery, indicating robust performance for the most commonly observed vehicle sizes in traffic.

The classification module attains an overall accuracy of 77%, demonstrating perfect recognition (100% precision and recall) for Bajaj, Box, Bus, and Pickup-Truck categories. While MPV/SUVs achieved a solid 68% precision with full recall, performance varied, with challenges observed for Hatchbacks and Sedans due to limited testing samples, resulting in 0% metrics for these classes. The system's weighted F1-score of 0.69 underscores its potential for real-world deployment, particularly for scenarios dominated by common vehicle types like Buses and MPV/SUVs. Addressing class imbalance will be crucial for achieving universal applicability. However, we omit this step for

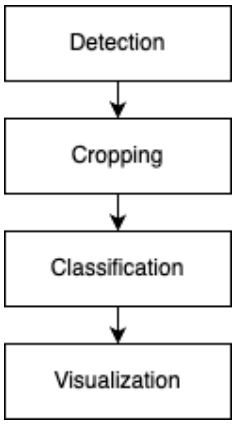


Figure 1: The pipeline of Indonesian Car Retrieval System.

simplicity. Table 2 summarizes the details of this project, including the GitHub link, output sample, and other relevant document links.

Table 2: Project Detail

Item	**Link**
Project's Github	https://github.com/lhfazry/icrs
Sample Video Output	Link
Detection Model Weight	Link
Detection Training Log	Link
Detection Evaluation Log	Link
Classification Model Weight	Link
Classification Training Log	Link
Classification Evaluation Log	Link

2 System Architecture

2.1 Overall Pipeline

The system adopts a modular four-stage pipeline optimized for heterogeneous traffic, which consists of some steps: detection, cropping, classification, and visualization. Figure 1 summarizes this pipeline.

1. Detection

- *Input*: Raw video stream (1080p @ 30FPS)
- *Process*: Frame-wise vehicle localization using Faster R-CNN
- *Output*: Bounding boxes with confidence scores

The system ingests a raw 1080p video stream (30 FPS) and processes each frame independently using a Faster R-CNN architecture implemented in Detectron2. The detector is previously trained on a custom Indonesian Car Dataset. The explanation about the dataset will be given later. The detector localizes vehicles with region proposals generated by a Region Proposal Network (RPN), followed by bounding box regression and classification. For each detected vehicle, the module outputs: pixel coordinates of bounding boxes (x_1, y_1, x_2, y_2) and confidence scores (0-1 range, thresholded at 0.5). Despite the model output classification, we only use the bounding box result because the classification for the detected vehicle will be classified by using a separate classification model.

2. Cropping

- *Input*: Detected bounding boxes

- *Process*:
  - 5% context expansion around boxes
  - Adding square padding (black [0,0,0]) at the bottom
  - Resize to 128×128 pixels
- *Output*: Normalized vehicle patches

The system processes detected bounding boxes through three normalization steps: First, each box is expanded by 5% to preserve contextual features (e.g., vehicle logos near edges). Next, black padding ([0,0,0]) is added exclusively at the bottom to create square aspect ratios while maintaining consistent feature positioning. Finally, patches are resized to 128×128 pixels using bilinear interpolation. The purpose of the resize is to match the classification model's input, which commonly expects a square size.

3. Classification

- *Input*: Normalized patches
- *Process*: Per-patch inference via EfficientNet-B0
- *Output*: Vehicle category + classification confidence

Each normalized 128×128 patch undergoes inference through an EfficientNet-B0 model optimized for vehicle recognition. The lightweight architecture processes patches individually, extracting hierarchical features through its compound scaling-based structure. The step outputs a specific vehicle category and classification confidence.

4. Visualization

- Overlays bounding boxes and class labels
- Filters detections (< 0.5 confidence)

The step generates annotated output by overlaying detection and classification results on the original video frames. Bounding boxes are drawn in a rectangle, and formatted labels display both vehicle category and confidence score (e.g., "Bus 0.77"). For simplicity, we use the same color for all classes, but it is a straightforward step to make the color class-specific. The pipeline automatically filters out all detections with confidence scores below 0.5 to ensure only reliable predictions are displayed.

2.2 Components

2.2.1 Detection Module (Faster R-CNN via Detectron2)

We employ Faster R-CNN for the detection module with implementation through a popular Detetron2 framework [Merz et al., 2023]. We fine-tune the pre-trained model previously trained on the Coco dataset [Lin et al., 2015] using a custom Indonesian Car dataset. The Coco dataset is a popular collection of object detection datasets that contains 91 classes, including car, bus, and truck. We use Faster R-CNN because it's a robust detection model.

Faster R-CNN (Region-based Convolutional Neural Network) is a seminal two-stage object detection framework that significantly improved upon its predecessors, R-CNN and Fast R-CNN, by integrating the region proposal generation directly into the neural network. This innovation, introduced by [Ren et al., 2015], addressed the computational bottleneck of external region proposal algorithms (like Selective Search), making the entire detection pipeline end-to-end trainable and much faster. At its core, Faster R-CNN consists of two main modules: a Region Proposal Network (RPN) and a Fast R-CNN detector. Figure 2 illustrates the Region Proposal Network (RPN) of Faster R-CNN.

The first stage, the Region Proposal Network (RPN), takes the feature map generated by a backbone Convolutional Neural Network (CNN) (e.g., VGG, ResNet [Simonyan and Zisserman, 2015]) as input. The RPN then slides a small neural network over this feature map to simultaneously predict object objectness scores and refine bounding box coordinates for a set of predefined anchor boxes. Anchor boxes are a set of bounding boxes with various scales and aspect ratios, designed to cover objects of different sizes and shapes in the image. For each anchor box k at a sliding-window location (x, y), the RPN outputs two values: an objectness score p_k (probability of containing an object) and

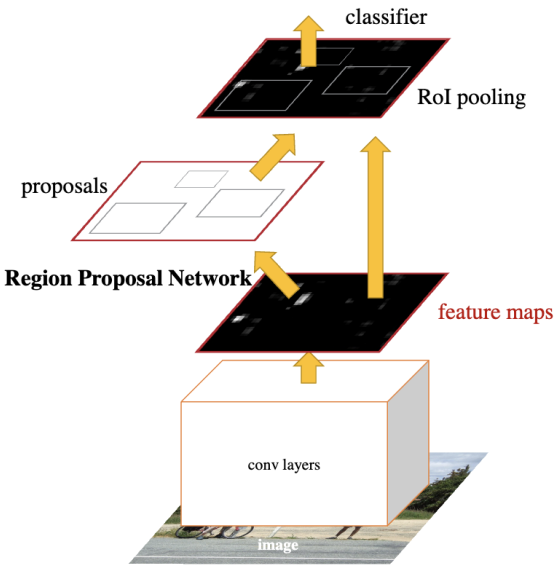


Figure 2: Regional Proposed Network (RPN) of Faster RCN ([Ren et al., 2015]).

four regression parameters $t_k = (t_x, t_y, t_w, t_h)$ that represent the predicted offsets from the anchor box to the true object's bounding box. The overall loss function for the RPN combines a classification loss (binary cross-entropy for object/non-object) and a regression loss (Smooth L1 loss for bounding box refinement).

The region proposals generated by the RPN are then fed into the second stage, which is essentially a Fast R-CNN detector. This detector utilizes a Region of Interest (RoI) Pooling layer to extract fixed-size feature maps from the shared convolutional features for each proposed region. These fixed-size features are then passed through fully connected layers, followed by two sibling output layers: one for classifying the object within the proposal into specific categories (e.g., car, bus, pedestrian) using a softmax function, and another for further refining the bounding box coordinates using regression. The entire Faster R-CNN network is trained end-to-end using a multi-task loss function that jointly optimizes both the RPN and the Fast R-CNN detection network, enabling shared feature learning and significantly boosting detection speed and accuracy

2.2.2 Classification Module (EfficientNet-B0)

For the classification model, we use the EfficientNet model, which not only has a good performance but is also efficient. We use the official PyTorch implementation of this model. We fine-tune the pre-trained model of EfficientNet-B0, which was previously trained on ImageNet [Deng et al., 2009].

EfficientNet, introduced by [Tan and Le, 2019], is a family of convolutional neural networks designed to achieve superior accuracy and efficiency compared to prior models. The core innovation of EfficientNet lies in its compound scaling method. Traditional approaches to scaling neural networks typically increase one dimension at a time—either depth (number of layers), width (number of channels), or resolution (input image size). EfficientNet, however, empirically observed that it is crucial to uniformly scale all three dimensions in a balanced manner to maximize performance. This is achieved by using a compound coefficient ϕ, which proportionally scales depth (α^ϕ), width (β^ϕ), and resolution (γ^ϕ), where α, β, γ are constant scaling factors determined through a small grid search on a baseline model.

The baseline model for the EfficientNet family is EfficientNet-B0. This base architecture was discovered through Neural Architecture Search (NAS) and is built upon Mobile Inverted Bottleneck Convolution (MBConv) blocks, a key component first seen in MobileNetV2 [Sandler et al., 2018]. MBConv blocks are characterized by an "inverted residual" structure where the input channels are first expanded to a higher dimension with a 1x1 convolution, then processed by a depthwise separable convolution (which applies a single filter per input channel), and finally compressed back to fewer

channels with another 1x1 convolution. This design significantly reduces computational cost while preserving representational capacity. Each MBConv block also incorporates a Squeeze-and-Excitation (SE) module, which adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels.

EfficientNet-B0 serves as the foundational model, offering a good balance between accuracy and computational cost. By applying the compound scaling factors (α, β, γ) derived from the initial grid search, the original authors systematically scaled EfficientNet-B0 to create a series of larger models, from EfficientNet-B1 to B7. Each subsequent model in the series represents a larger, more complex version with increased depth, width, and resolution, allowing for better performance at the cost of higher computational requirements. This principled scaling approach ensures that each variant is optimally configured for its target computational budget, making EfficientNet a highly adaptable and efficient architecture for various computer vision tasks, including image classification.

3 Methodology

3.1 Dataset Preparation

Effective model training in vehicle detection and classification hinges on a meticulously prepared dataset that accurately reflects the target environment. This section details the creation and preprocessing steps for both our detection and classification datasets, crucial for optimizing performance within the Indonesian traffic context.

3.1.1 Detection Dataset

A custom dataset focused on Indonesian vehicles was curated for the detection task. The images were sourced from publicly accessible CCTV feeds within Jakarta, specifically from the Bali Tower network (e.g., footage from https://cctv.balitower.co.id/Tomang-004-702108_2/embed.html). These raw images possess an original resolution of 1920x1080 pixels, capturing a wide field of view common in urban surveillance.

The annotation process was conducted using Roboflow, a specialized platform for dataset management and labeling. During this process, bounding boxes were drawn around target vehicles, and each instance was assigned to one of eight predefined categories: Bajaj, Box, Bus, Hatchback, MPV_SUV, Minibus, Pickup-Truck, and Sedan. The original high-resolution images were then preprocessed by Roboflow, including a resizing operation to 576x324 pixels. This downscaling was strategically chosen to accelerate subsequent processing steps during model training while preserving the original aspect ratio of the scene. Figure 3 illustrates the labelling process on the Roboflow platform. Figure 4 shows some sample images from the detection dataset along with the bounding box.

The dataset was subsequently partitioned into training, validation, and test sets. The final distribution is as follows: training set 117 images, validation set 17 images, and test set 18 images. An analysis of the class distribution within the detection dataset (based on instances, not images) reveals significant imbalances, which is a common characteristic of real-world traffic data. The distribution of vehicle instances across classes is detailed in Table 3. Figure 5 illustrates the distribution of vehicle instances in the detection dataset.

Table 3: Class Distribution of Vehicle Instances in Detection Dataset

Vehicle Class	Number of Instances
MPV_SUV	161
Box	52
Minibus	52
Bus	38
Pickup-Truck	27
Hatchback	8
Bajaj	7
Sedan	3

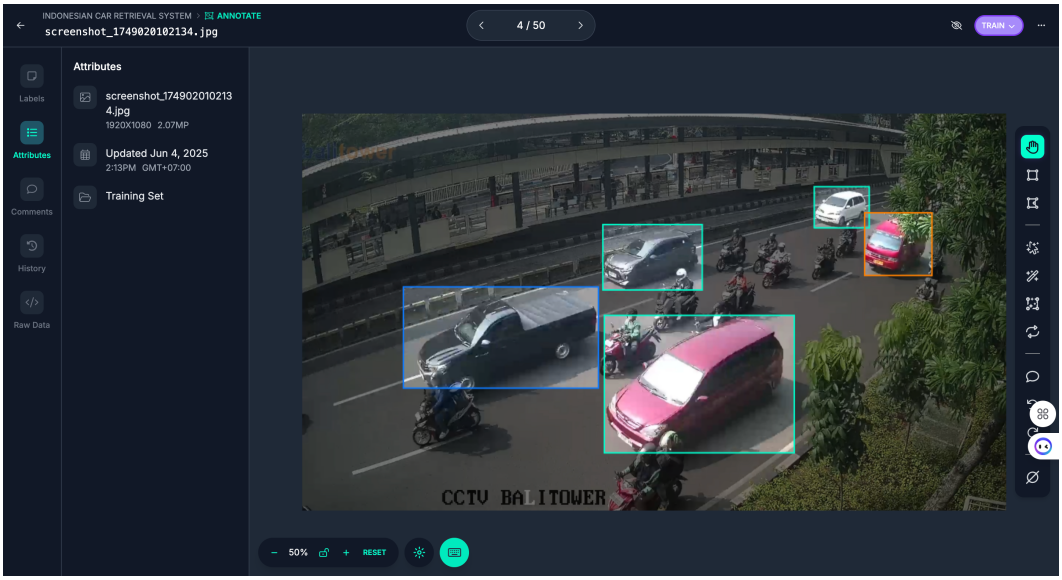


Figure 3: A screenshot of labelling processing on the Roboflow platform

Figure 4: Some of the sample images in Detection Dataset

3.1.2 Classification Dataset

The classification dataset was derived directly from the bounding box annotations of the detection dataset, ensuring consistency between the two tasks. For each annotated bounding box, a sub-image representing the individual vehicle instance was extracted using a custom bounding box cropping algorithm.

To standardize the input for the classification model and mitigate issues arising from varying aspect ratios of cropped vehicles, a square padding methodology was applied. Each cropped vehicle image was resized and padded to a uniform 128x128 pixel resolution. This process involves scaling the longest dimension of the cropped image to 128 pixels, and then padding the shorter dimension with a neutral color (black in this case) to form a 128x128 square. This ensures that the original aspect ratio of the vehicle is maintained without distortion, while providing a consistent input size for the EfficientNet classification model. Figure 6 shows some image samples from the classification dataset.

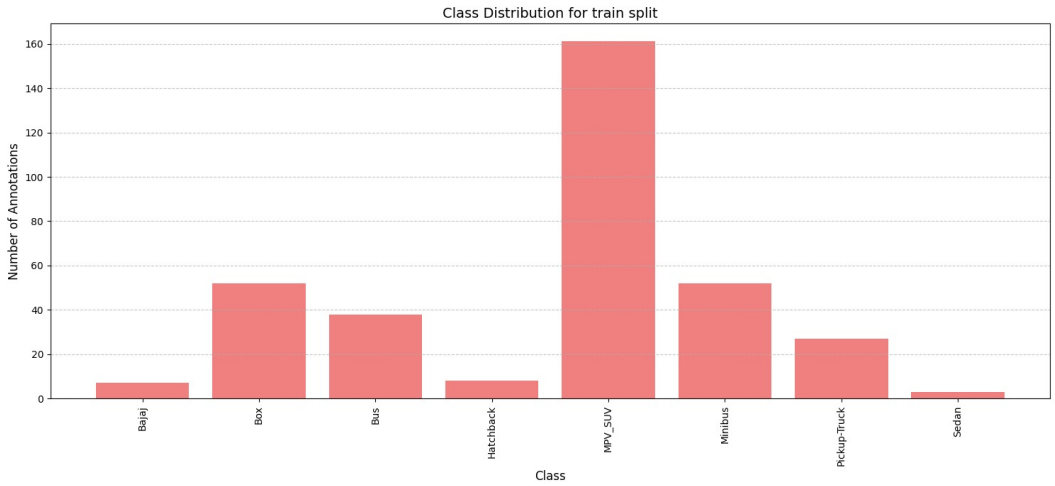


Figure 5: Class Distribution of Vehicle Instances in Detection Dataset

3.2 Detection Model

Choosing the right detection model is crucial for building a robust vehicle detection pipeline. We evaluated popular object detection architectures, focusing on their suitability for the unique challenges of Indonesian traffic.

3.2.1 Model Selection: Faster R-CNN vs. YOLO Analysis

Our model selection process involved a comparative analysis between two prominent object detection paradigms: Faster R-CNN (a two-stage detector) and YOLO (You Only Look Once) [Redmon et al., 2016] series (a one-stage detector). Faster R-CNN, known for its high accuracy, separates the object detection process into two phases: first, generating region proposals, and then classifying and refining these proposals (though we do not use the classification result through this exam). This sequential approach often yields superior localization accuracy, particularly for small or overlapping objects, which are prevalent in dense traffic scenarios. YOLO, on the other hand, performs detection in a single pass, predicting bounding boxes and class probabilities simultaneously. While generally faster, YOLO can sometimes struggle with precise localization and detecting small objects due to its grid-based approach. Given the primary requirement for accurate identification and precise bounding box generation of diverse vehicle types in varied conditions, we opted for Faster R-CNN [Ren et al., 2015]. Its established performance in handling complex scenes and diverse object scales made it the more appropriate choice for our specific application, where detection accuracy was prioritized over raw inference speed, especially with the later integration of a dedicated classification module [Tan and Le, 2019].

3.2.2 Training Process

The Faster R-CNN model was trained using the Detectron2 framework, leveraging its flexible and efficient implementation. The training process was meticulously configured to optimize performance on our custom Indonesian vehicle dataset. Figure 7a and 7b show the total loss plot and bounding box plot during training.

3.2.3 Hyperparameter Configuration

We initialized the training process by loading a pre-trained Faster R-CNN model with a ResNet-50-FPN backbone from the Detectron2 model zoo. This leverages knowledge learned from the large COCO dataset, providing a strong starting point for transfer learning. The key hyperparameter configurations are as follows:

- Configuration File: `cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))` - This command loads the base con-

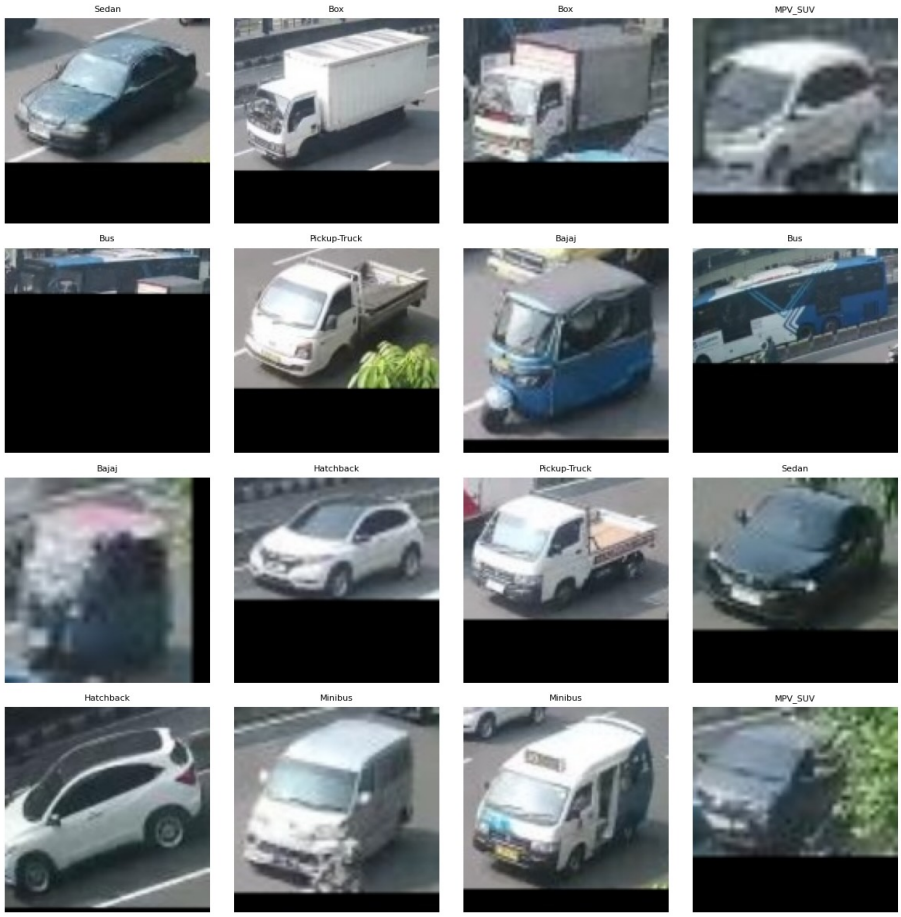
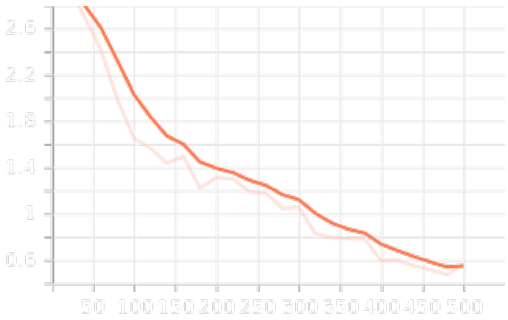
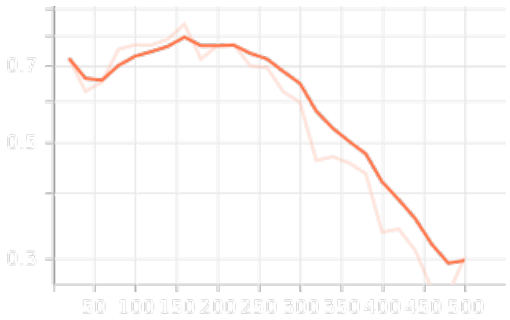


Figure 6: Sample of images from classification dataset



(a) Total loss plot of detection model



(b) Bounding box loss plot of detection model

figuration for a Faster R-CNN model with a ResNet-50 backbone and Feature Pyramid Network (FPN), pre-trained for a 3x schedule on COCO.

- Data Loader Workers: `cfg.DATALOADER.NUM_WORKERS = 2` - Specifies the number of subprocesses to use for data loading, balancing between CPU utilization and memory consumption.
- Model Weights: `cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/ faster_rcnn_R_50_FPN_3x.yaml")` - Sets the initial weights of the model to the pre-trained COCO weights, facilitating a transfer learning approach.

- Images Per Batch: `cfg.SOLVER.IMS_PER_BATCH = 4` - Determines the number of images processed in each training batch. A smaller batch size can sometimes help with generalization, especially with limited data.
- Base Learning Rate: `cfg.SOLVER.BASE_LR = 0.00025` - Defines the initial learning rate for the optimizer, a critical parameter influencing training convergence.
- Maximum Iterations: `cfg.SOLVER.MAX_ITER = 500` - Sets the total number of training iterations. This was chosen after preliminary experiments to ensure sufficient convergence while preventing overfitting on our relatively small dataset.
- ROI Head Batch Size: `cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128` - Specifies the number of RoIs (Regions of Interest) sampled for each image during training of the ROI heads.
- Number of Classes: `cfg.MODEL.ROI_HEADS.NUM_CLASSES = 8` - Configures the final classification layer of the model to predict probabilities for our eight specific vehicle categories.

3.2.4 Transfer Learning Approach

A transfer learning approach was fundamental to the success of our detection model, especially given the limited size of our custom dataset. Instead of training the Faster R-CNN model from scratch, we initialized its weights with a model pre-trained on the large and diverse COCO dataset [Lin et al., 2015]. This allowed us to leverage the rich, general-purpose feature representations learned from millions of images, which are highly beneficial for object detection tasks. By fine-tuning this pre-trained model on our specific Indonesian vehicle dataset, we could adapt these generic features to the nuances of our target domain, accelerating convergence, improving generalization, and mitigating overfitting risks that often arise with smaller datasets. This strategy enabled the model to achieve strong performance despite the constraints of our data.

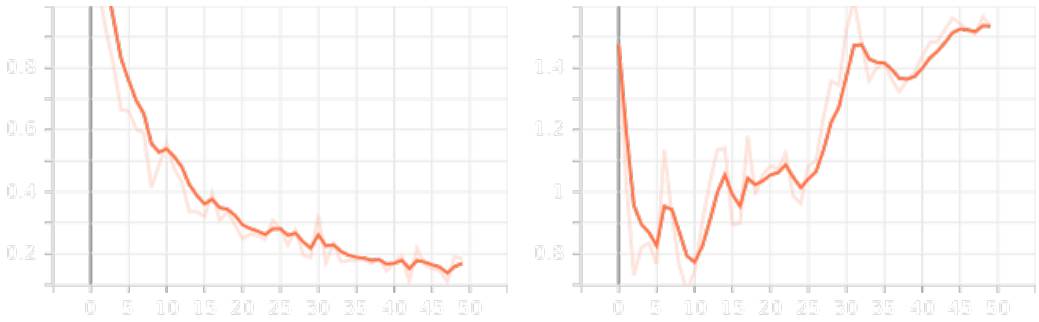
3.3 Classification Model

The classification module is a critical component of our pipeline, responsible for accurately categorizing detected vehicles. This section details the selection of the classification architecture, the strategies employed for its training, and the image processing techniques used to prepare the data.

3.3.1 Architecture Comparison

We considered two prominent convolutional neural network architectures for the classification task: ResNet50 [He et al., 2015] and EfficientNet-B0 [Tan and Le, 2019]. ResNet50 is a well-established architecture known for its deep residual learning framework, which helps mitigate vanishing gradients in very deep networks. It offers strong performance and is widely used. EfficientNet-B0, on the other hand, is the baseline model of the EfficientNet family, characterized by its compound scaling method and efficient Mobile Inverted Bottleneck Convolution (MBConv) blocks. EfficientNet architectures are generally known for achieving high accuracy with significantly fewer parameters and computations compared to ResNets of similar performance. For our application, where both accuracy and computational efficiency (especially for potential edge deployment) are important, EfficientNet-B0 was chosen. Its superior parameter efficiency allowed us to achieve competitive classification accuracy with a smaller model footprint, making it more suitable for integration into a real-time pipeline.

Both ResNet50 and EfficientNet-B0 act as powerful feature extractors. Their initial layers learn low-level features like edges and textures, while deeper layers progressively extract more abstract, high-level features crucial for distinguishing between different vehicle types. In EfficientNet-B0, the MBConv blocks, combined with Squeeze-and-Excitation (SE) modules [Hu et al., 2019], enhance the quality of these extracted features by dynamically recalibrating channel-wise dependencies. This sophisticated feature extraction capability is vital for handling the subtle visual differences between closely related vehicle classes in our dataset (e.g., distinguishing between different car body types).



(a) Total train loss plot of classification model

(b) Total validation loss plot of classification model

3.3.2 Training Strategy

The EfficientNet-B0 model was fine-tuned on our custom classification dataset with a specific training strategy to adapt it to our vehicle categories. To adapt the pre-trained EfficientNet-B0 model for our specific classification task, we replaced its original classification head with a custom one tailored for our 8 vehicle classes. The process involved identifying the number of input features to the original classifier and then constructing a new linear layer. This ensures that the final layer correctly outputs predictions for our eight categories, allowing the model to be fine-tuned on the new task while leveraging the powerful feature representations learned by the backbone. Figure 8a and 8b show the training and validation loss. The plots indicate that the model is overfit due to the increasing validation loss as the iterations increase. We can employ a strategy such as early stop to mitigate this issue or add more datasets.

```
num_ftrs = model.classifier[1].in_features
model.classifier[1] = nn.Linear(num_ftrs, 8) # Assuming 8 classes
```

For simplicity in this implementation, no explicit class imbalance handling techniques (such as weighted sampling, focal loss, or oversampling/undersampling) were applied during the training of the classification model. We acknowledge that the uneven distribution of instances across vehicle classes could potentially bias the model towards the majority classes.

The training ran for 50 epochs. We utilized the AdamW [Loshchilov and Hutter, 2019] optimizer with an initial learning rate of 0.001. To effectively guide the optimization process and improve convergence, a cosine annealing learning rate scheduler was employed [Loshchilov and Hutter, 2017]. This scheduler dynamically adjusts the learning rate throughout training, gradually decreasing it from its initial value to a minimum, following a cosine curve. The `T_max` parameter was set to 50, meaning the learning rate completes one half-cycle of the cosine function over the entire 50 epochs. This helps the model explore the loss landscape more thoroughly at higher learning rates initially and fine-tune weights precisely with smaller rates towards the end of training.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50)
```

3.3.3 Image Processing

A series of data augmentation techniques was applied during training to enhance the model's robustness and generalization capabilities. These transformations introduce variability into the training data, helping the model become invariant to minor variations in illumination, scale, orientation, and position.

```
train_transform = transforms.Compose([
  transforms.RandomResizedCrop(224),
  transforms.RandomHorizontalFlip(),
  transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
  transforms.RandomRotation(15),
  transforms.ToTensor(),
```

```
])
```

Following all augmentations, images were normalized using standard ImageNet mean and standard deviation values. This standardization is critical because the EfficientNet-B0 model was pre-trained on ImageNet [Deng et al., 2009], and applying the same normalization scheme ensures consistency in the input data distribution.

```
transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

These values represent the mean and standard deviation of pixel intensities across the ImageNet dataset's three color channels (Red, Green, Blue). Normalizing the input data helps stabilize training and improve model performance.

For validation and testing, a deterministic sequence of transformations was applied to ensure consistent input to the model without introducing stochasticity. This process ensures that evaluation is performed on consistently preprocessed images, allowing for fair and reproducible measurement of model performance.

```
val_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

4 Experimental Results

This section presents a comprehensive evaluation of the developed vehicle detection and classification pipeline. We assess performance using a combination of quantitative metrics for both detection and classification tasks, complemented by qualitative analysis to provide deeper insights into the system's behavior in real-world scenarios.

4.1 Detection Metrics

The detection performance of the Faster R-CNN model was evaluated using standard COCO metrics, primarily focusing on Average Precision (AP) across various Intersection over Union (IoU) thresholds and object scales. Table 4 summarizes overall detection results.

Table 4: Overall Detection Performance Metrics

Metric	**Value (%)**
AP (Average Precision @ IoU=[0.50:0.95])	17.13
AP50 (Average Precision @ IoU=0.50)	22.91
AP75 (Average Precision @ IoU=0.75)	21.07

These results indicate that while the model shows a reasonable performance at a lower IoU threshold (AP50), its performance drops when higher localization accuracy is required (AP75). The overall AP, averaged across various IoU thresholds, reflects the challenging nature of the dataset and the limited training data.

We also provide the detection based on object-scale. Table 5 summarizes the performance result across different object scales. Please note that, N/A in APs likely due to an insufficient number or absence of defined "small" objects in the test set after resizing, or the detection model's difficulty with very small instances. This breakdown highlights that the detector performs comparatively better on medium-sized vehicles, which likely constitute a significant portion of visible vehicles in the resized (576x324) CCTV footage. Performance on large objects is considerably lower.

Per-class Average Precision (AP) provides a more granular view of the detector's capabilities as shown in Table 6. The model demonstrates strong detection performance for dominant classes like MPV_SUV, Bus, and Box, achieving over 40% AP. However, it struggles significantly with classes

Table 5: Detection Performance by Object Scale

Metric	Value (%)
APm (Average Precision for medium objects)	23.40
APl (Average Precision for large objects)	11.56
APs (Average Precision for small objects)	N/A

having very few instances in the dataset (Bajaj, Hatchback, Minibus, Pickup-Truck, Sedan), resulting in 0% AP for these categories. This indicates that the detection module either fails to propose regions for these classes or misclassifies proposals during the detection phase due to severe class imbalance.

Table 6: Per-Class Average Precision (AP) for Detection

Vehicle Class	AP (%)
MPV_SUV	53.97
Bus	42.71
Box	40.40
Bajaj	0.00
Hatchback	0.00
Minibus	0.00
Pickup-Truck	0.00
Sedan	0.00

4.2 Classification Metrics

The classification module's performance was evaluated using standard classification metrics, including accuracy, precision, recall, and F1-score, with a focus on per-class analysis. Table 7 summarizes the overall classification performance.

Table 7: Overall Classification Performance Metrics

Metric	Value (%)
Overall Accuracy	77
Macro Average Precision	71
Macro Average Recall	64
Macro Average F1-score	63
Weighted Average Precision	74
Weighted Average Recall	77
Weighted Average F1-score	69

The overall accuracy of 77% indicates that the classifier correctly identifies the vehicle type for a significant portion of detected instances. The difference between macro and weighted averages highlights the impact of class imbalance: the weighted average, which accounts for the number of instances per class, is generally higher due to strong performance on majority classes. Table 8 shows a detailed breakdown of precision, recall, and F1-score for each class reveals varying performance.

Table 8: Per-Class Classification Performance Metrics

Class	Precision	Recall	F1-Score	Support
Bajaj	1.00	1.00	1.00	1
Box	1.00	1.00	1.00	4
Bus	1.00	1.00	1.00	6
Hatchback	0.00	0.00	0.00	4
MPV_SUV	0.68	1.00	0.81	23
Minibus	1.00	0.14	0.25	7
Pickup-Truck	1.00	1.00	1.00	1
Sedan	0.00	0.00	0.00	1

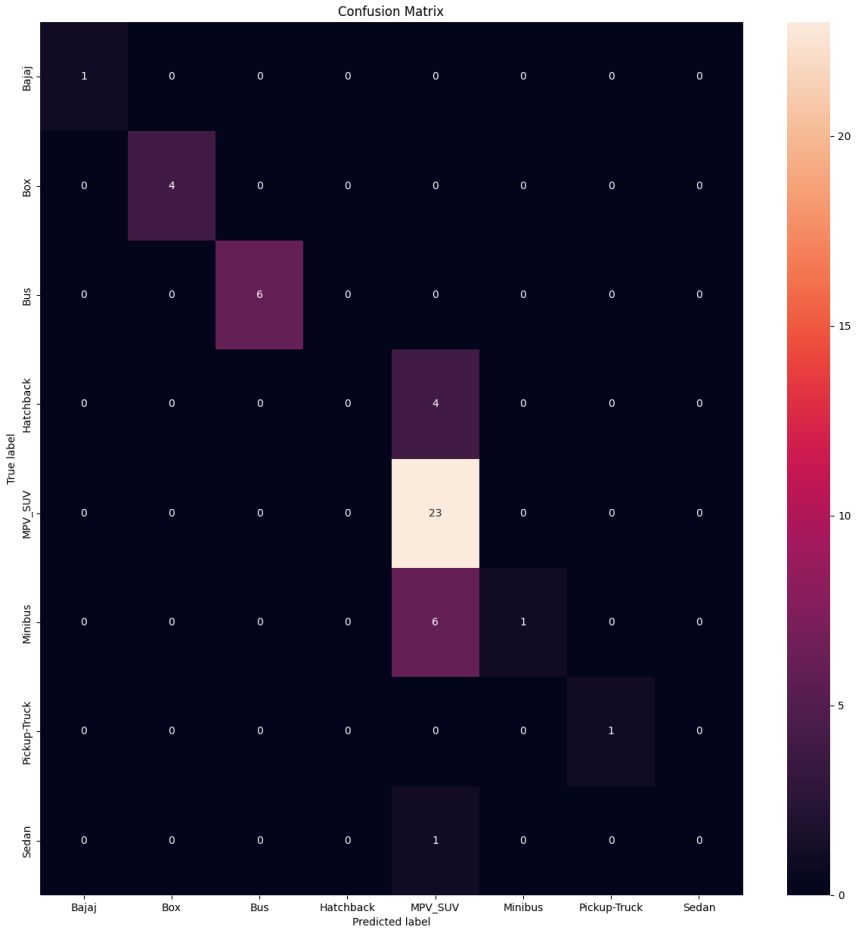
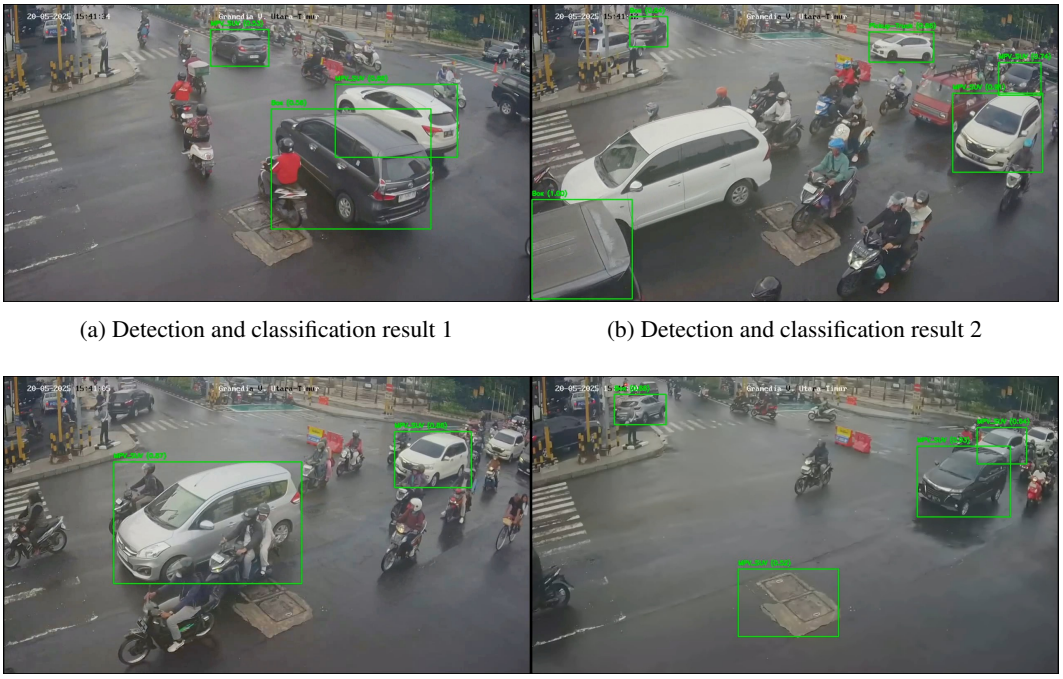


Figure 9: A heatmap of classification confusion matrix

The confusion matrix provides crucial insights into specific misclassifications made by the classifier as shown in Figure 9. Confusion matrix reveals that the model performs exceptionally well for Bajaj, Box, Bus, MPV_SUV, Pickup-Truck, and Sedan, with perfect precision and recall (or very high for MPV_SUV). This indicates robust classification for these categories, especially for MPV_SUV which has the highest support.

On the other hand, despite has 4 support samples, the model achieved 0% precision, recall, and F1-score on Hatchback. This suggests that all actual Hatchback instances were misclassified, or no instances were correctly identified as Hatchback. Similar to Hatchback, with only 1 support sample, the model failed to correctly classify any Sedan instances.

In addition, Minibus has low recall and F1-score. While the precision for this class is 1.00 (all its predictions were correct if it predicted Minibus), its recall is only 0.14 and F1-score 0.25. The confusion matrix indicates that 1 instance that was actually a Minibus was misclassified as a Hatchback. This particular misclassification highlights a potential confusion or similarity in features between these two vehicle types from the model's perspective, or simply the very limited data for these classes.



(a) Detection and classification result 1

(b) Detection and classification result 2

(c) Detection and classification result 3

(d) Detection and classification result 4

Figure 10: Overall Figure Caption

These results underscore the significant impact of class imbalance on the classification performance, particularly for classes with very few samples (e.g., Bajaj, Pickup-Truck, Sedan, Hatchback, Minibus). While the model excels at classifying more frequently observed vehicle types, rare classes present a considerable challenge.

4.3 Qualitative Results

Beyond quantitative metrics, qualitative analysis provides valuable insights into the system's practical performance and identifies specific scenarios where the model excels or struggles. Figure 10 shows some detection and classification result on the provided video test. As suggested by per-class classification result, we can see that the model is excel at detecting and classifying the MPV_SUV, but the model can miss-detect and miss-classify other categories.

References

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL `https://arxiv.org/abs/1512.03385`.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019. URL `https://arxiv.org/abs/1709.01507`.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL `https://arxiv.org/abs/1405.0312`.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL `https://arxiv.org/abs/1608.03983`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Grant Merz, Yichen Liu, Colin J Burke, Patrick D Aleo, Xin Liu, Matias Carrasco Kind, Volodymyr Kindratenko, and Yufeng Liu. Detection, instance segmentation, and classification for astronomical surveys with deep learning (<scp>deepdisc</scp>): <scp>detectron2</scp> implementation and demonstration with hyper suprime-cam data. *Monthly Notices of the Royal Astronomical Society*, 526(1):1122–1137, September 2023. ISSN 1365-2966. doi: 10.1093/mnras/stad2785. URL http://dx.doi.org/10.1093/mnras/stad2785.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL https://arxiv.org/abs/1506.02640.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015. URL https://api.semanticscholar.org/CorpusID:10328909.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. URL https://api.semanticscholar.org/CorpusID:4555207.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. URL https://arxiv.org/abs/1409.1556.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019. URL https://api.semanticscholar.org/CorpusID:167217261.