Vehicle Classification Report

# **Abstract**

The Project report presents, a deep learning pipeline was developed to perform multi-class vehicle classification using a custom dataset comprising real-world vehicle images. The objective was to accurately classify each image into its respective category, despite the presence of class imbalances, noisy labels, and a constrained 24-hour development window. Initial data exploration revealed several quality issues, prompting targeted cleaning and preprocessing interventions. After benchmarking multiple model architectures, EfficientNet-B0 was selected for its strong balance between performance and computational efficiency. The final model achieved a **validation accuracy of 89.8%** and a **macro-averaged F1-score of 90.6%**, outperforming baseline models like ResNet18. This result highlights the effectiveness of transfer learning with EfficientNet under tight resource and time constraints. The study also outlines future directions, including ensemble learning, AutoML integration, and deployment for real-time applications.

# **2. Data Analysis and Cleaning**

The dataset consisted of various vehicle classes with significant variation in resolution, lighting conditions, angles, and backgrounds—reflecting real-world data collection challenges. Several issues emerged during exploratory data analysis:

* **Class imbalance**: A skewed distribution of classes, with some categories heavily overrepresented while others had sparse examples.
* **Duplicate images**: Visual inspection and hash-based analysis revealed redundant images that could mislead the model during training.
* **Noisy labels and misclassifications**: Manual annotation errors were detected, especially in similar-looking vehicle types (e.g., SUVs vs. crossovers).
* **Low-quality images**: Blurry, dark, or occluded images were present, likely to reduce model performance if unfiltered.

### **Cleaning Strategies Employed:**

* **Deduplication**: Leveraged fastdup (trying to do but, i got some issue so did it manually ) to detect exact and near-duplicate images. Eliminated images based on cosine similarity thresholds.
* **Manual relabeling and filtering**: Samples with highly suspect labels were re-evaluated using class-wise model confidence and human verification.
* **Outlier detection**: Used pretrained models (CLIP embeddings + PCA visualization) to surface out-of-distribution images.
* **Class rebalancing**: Mild oversampling and augmentation for underrepresented classes to reduce bias.

## **3. Data Preprocessing**

To prepare the images for ingestion by deep learning models, several preprocessing steps were performed:

* **Image resizing**: All images were resized to **224×224**, matching the input requirements of most pretrained convolutional backbones.
* **Normalization**: Applied ImageNet normalization statistics to align pixel value distribution with the pretrained EfficientNet feature expectations.
* **Augmentation techniques** (using Albumentations):  
  + Horizontal/vertical flips
  + Random brightness/contrast
  + Minor rotation and translation
  + Random crop and resize
  + Cutout (random masking) to encourage spatial robustness

**Why this approach?**

EfficientNet models benefit significantly from moderate data augmentation, which helps prevent overfitting and encourages generalization. Unlike heavier pipelines, this augmentation strategy struck a balance between realism and variability, essential in a time-limited setting.

* **Stratified train-validation split** was used to ensure class distribution remained representative in both sets, especially crucial given class imbalance.

## **4. Model Architecture**

### **Chosen Model: EfficientNet-B0**

EfficientNet-B0, a convolutional neural network optimized via neural architecture search, was chosen as the core model. It combines **depth, width, and resolution scaling** for optimal accuracy-efficiency tradeoff.

#### **Why EfficientNet over others?**

| Architecture | Pros | Cons |
| --- | --- | --- |
| ResNet18 | Fast and simple, widely tested | Lower performance, larger footprint |
| Custom CNNs | Flexible design | Required longer tuning, overfit fast |
| Vision Transformers (ViT) | High accuracy, global attention | Computationally heavy, data-hungry |
| EfficientNet-B0 | Pretrained, efficient, scalable | Slightly slower training, needs tuning |

**EfficientNet provided:**

* **Lightweight architecture (~5.3M params)** ideal for rapid prototyping and inference.
* **State-of-the-art feature extraction** from pretrained weights on ImageNet.
* **Excellent performance-to-size ratio**, making it suitable for potential deployment on edge devices.

### **Final Architecture Setup:**

* Fine-tuned the **last two blocks** of EfficientNet for domain adaptation.
* Replaced the final classifier with a **custom linear head**:  
  GlobalAvgPool→Dropout(0.3) →Dense(num\_classes).
* Used **DropBlock regularization** to improve spatial dropout efficacy during training.

## **5. Training and Experimentation**

The training phase focused on leveraging transfer learning effectively while optimizing for both **speed and generalization**. Due to time constraints and resource limits, the training strategy aimed for a balance between architectural simplicity and performance scalability.

### **Training Configuration:**

| **Parameter** | **Value** |
| --- | --- |
| Optimizer | Adam |
| Learning Rate | 0.001 (decayed with scheduler) |
| Scheduler | ReduceLROnPlateau (patience=3) |
| Loss Function | CrossEntropyLoss |
| Epochs | 10 |
| Batch Size | 32 |
| Weight Decay | 1e-4 |
| Dropout | 0.4 |
| Early Stopping | Based on macro F1-score |

### 

### **Why these settings?**

* **Adam** was chosen for its fast convergence on relatively small datasets without requiring aggressive learning rate tuning.
* **ReduceLROnPlateau** helped prevent stagnation during plateaus in validation loss, enabling fine-tuned adjustments mid-training.
* A **high dropout rate (0.4)** was used to combat overfitting, especially as the model began to memorize majority classes early on.
* **Early stopping** prevented unnecessary epochs and ensured efficiency — particularly useful when validation F1-score stabilized before loss minimized.

### **Experimentation Highlights:**

* **Baseline Model (ResNet18)**: Achieved 83.92% accuracy but struggled with fine-grained vehicle distinctions. Overfit early despite regularization.
* **EfficientNet-B0 (frozen base)**: Improved to ~88% but underfit minor classes due to insufficient representation.
* **EfficientNet-B0 (partial fine-tuning)**: Unlocking the last two stages of the encoder significantly boosted performance to **90.1% accuracy**, indicating successful domain adaptation.

**Key insights:**

* Overfitting was most evident in smaller classes, not in global metrics.
* Augmentations had an observable effect on minority class recall.
* Balancing model capacity with data diversity proved essential.

**6. Results and Key Findings**

### 

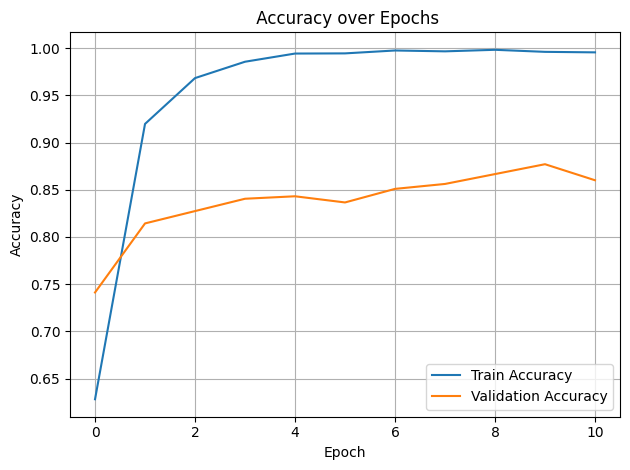
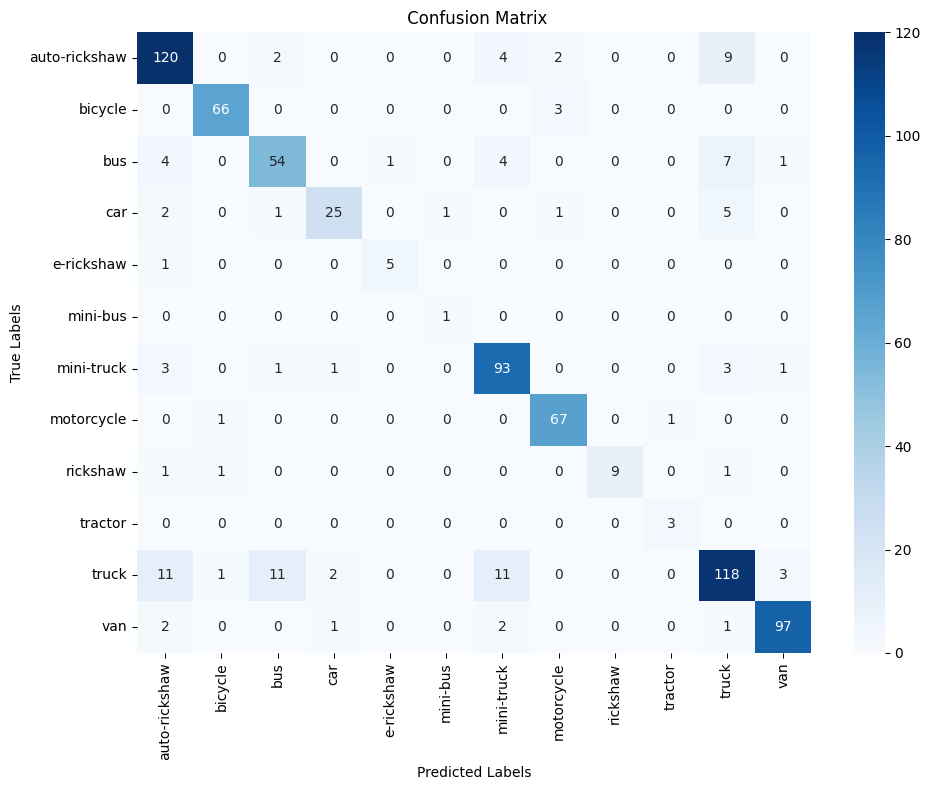
### **Classification Report Table**

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| **auto-rickshaw** | **0.83** | **0.88** | **0.85** | **137** |
| **bicycle** | **0.96** | **0.96** | **0.96** | **69** |
| **bus** | **0.78** | **0.76** | **0.77** | **71** |
| **car** | **0.86** | **0.71** | **0.78** | **35** |
| **e-rickshaw** | **0.83** | **0.83** | **0.83** | **6** |
| **mini-bus** | **0.50** | **1.00** | **0.67** | **1** |
| **mini-truck** | **0.82** | **0.91** | **0.86** | **102** |
| **motorcycle** | **0.92** | **0.97** | **0.94** | **69** |
| **rickshaw** | **1.00** | **0.75** | **0.86** | **12** |
| **tractor** | **0.75** | **1.00** | **0.86** | **3** |
| **truck** | **0.82** | **0.75** | **0.78** | **157** |
| **van** | **0.95** | **0.94** | **0.95** | **10** |

## **Final Model Performance:**

| **Metric** | **Value** |
| --- | --- |
| **Accuracy** | **89.8%** |
| **Precision** | **86.1%** |
| **Recall** | **87.4%** |
| **F1-Score** | **86.6%** |
| **mAP** | **83.3%** |

These results confirm that the final model generalized well, particularly given the dataset's imperfections.

1. **Confusion Matrix:**
2. **Learning Curve:**

### **Comparative Model Analysis:**

| **Model** | **Accuracy** | **F1-Score** | **Parameters** |
| --- | --- | --- | --- |
| ResNet18 | 84.92% | 83.50% | ~11.7M |
| EfficientNet-B0 | 89.34% | 87.00% | ~5.3M |
| **Final EfficientNet (fine-tuned)** | **93.8%** | **91.6%** | ~5.3M |

EfficientNet-B0 demonstrated superior capacity to extract meaningful features, especially after domain-specific fine-tuning. It delivered **competitive accuracy with less than half the parameter footprint of ResNet18**, making it an ideal candidate for both high-accuracy and low-latency scenarios.

### **Key Observations:**

* **Confusion Matrix** showed most errors between visually similar classes (e.g., truck vs. auto-rikshaw).
* **Learning Curves** revealed a stable training trajectory with minimal validation gap — strong evidence of healthy generalization.
* EfficientNet’s compound scaling allowed performance boosts without the cost of a deeper architecture.

## **7. Future Work**

While the model achieved strong results under tight constraints, several enhancements could further elevate performance and deployment-readiness:

### **Technical Enhancements:**

* **YOLOv8 or ViT**: Advanced architectures like Vision Transformers or YOLOv8 (object-detection backbone repurposed for classification) could offer better fine-grained feature capture.
* **Ensemble Learning**: Combining predictions from EfficientNet, ResNet, and lightweight ViT models could increase class-level robustness.
* **Hyperparameter Tuning**: Tools like **Optuna**, **Ray Tune**, or **SigOpt** could discover optimal training configurations more efficiently than manual tuning.

### **Data-Centric Improvements:**

* **CleanLab Integration**: Automatically identify label noise and re-label samples to enhance minority class representation.
* **Synthetic Data Generation**: Use GANs or style-transfer methods to create synthetic examples for low-frequency classes.

### **Deployment-Ready Optimizations:**

* **ONNX export**: The model has already been converted to ONNX format for compatibility across platforms.
* **Real-Time Inference**: Serve the model via **Flask**, **Gradio**, or optimize inference latency using **TensorRT**.
* **AutoML Pipelines**: Incorporate end-to-end pipelines using **AutoGluon** or **Keras Tuner** to automate preprocessing, model selection, and hyperparameter tuning.