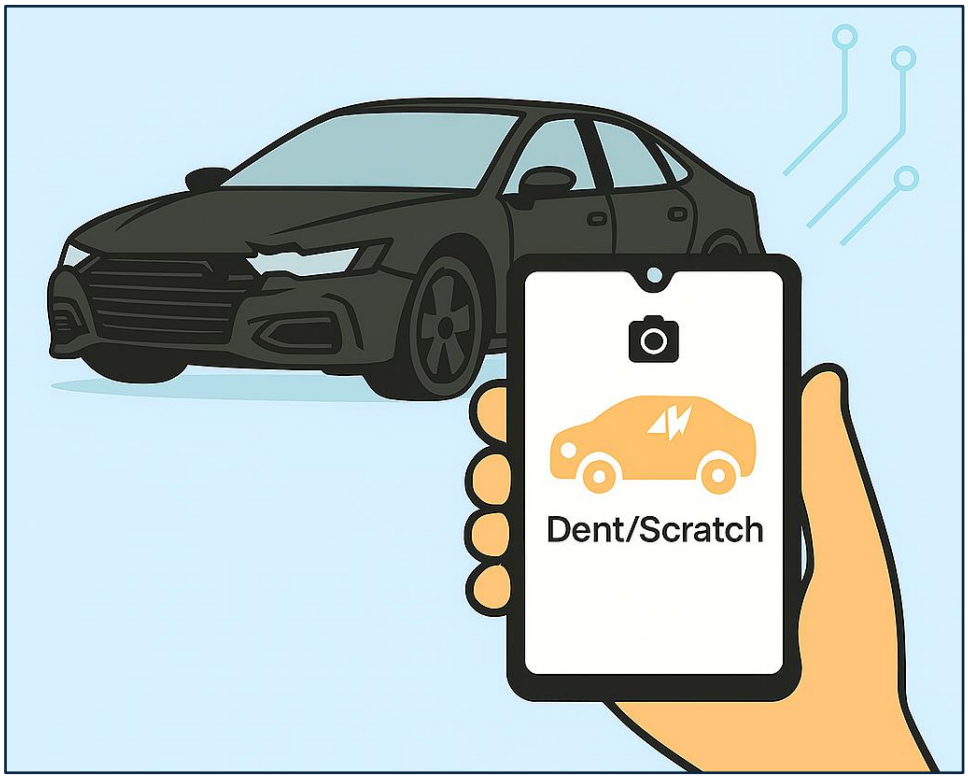
King Saud University
College of Computer and Information Sciences
Department of Information Technology

كلية علوم الحاسب والمعلومات
قسم تقنية المعلومات

Semester-1 1447H



Car Damage Classification

IT461 Project - Final Report

Prepared by

Section 72597	
Name	**ID**
Ahlam Alqahtani	444200835
Walaa Saif Aleslam	444200088
Rama Alomair	444200662
Danyh Alotaibi	444201030

Supervised by
Dr. Luluah Alhusain

Table of Contents

Table of Figure

Table of table

Introduction

In Riyadh, one of the fastest-growing metropolitan cities in the world, the number of vehicles on the road has increased tremendously in recent years. With rapid urban expansion and heavy daily traffic, the city faces **high congestion levels and frequent road accidents** [1][2]. According to local reports, traffic accidents in Saudi Arabia remain one of the leading causes of injuries and economic losses, placing a significant burden on both individuals and institutions [3]. Even minor accidents often lead to traffic delays and considerable financial and administrative costs for insurance companies, workshops, and vehicle owners [4].

A major challenge in this context is the **manual assessment of vehicle damage**, which usually requires trained inspectors. This process is **time-consuming, costly, and prone to human error** [5]. Such limitations can delay insurance claim approvals, increase operational expenses, and cause dissatisfaction among customers.

Our motivation to address this problem stems from the urgent need for **automated, accurate, and efficient solutions** in high-demand environments such as Riyadh. Developing a computer vision–based model for vehicle damage classification can directly support industries by accelerating insurance claim approvals, reducing disputes, and improving customer satisfaction [6][7]. Moreover, in alignment with **Saudi Vision 2030**, this project contributes to building intelligent transportation and service systems that can scale with the growing population and traffic demands [2].

The intended task of this project is straightforward: **given an image of a damaged vehicle, the model predicts the type of damage**. The **input** is an RGB image of the car showing external damage, and the **output** is a class label among different categories such as door_scratch, bumper_scratch, door_dent, bumper_dent, glass _shatter, head_lamp and tail_lamp [6][7][8].

This process is summarized in [Figure 1-Input and output below:

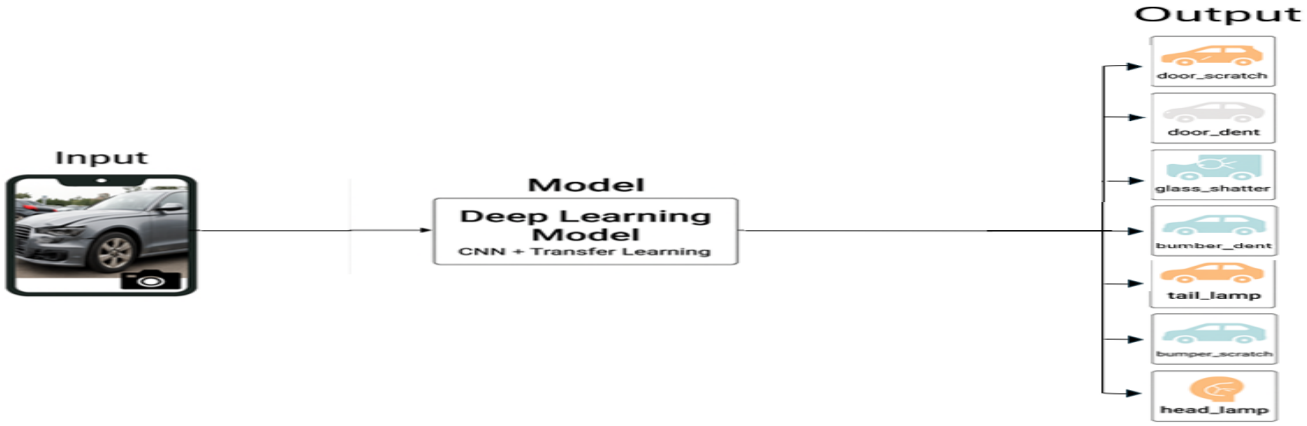


Figure 1-Input and output

By automating this task, our project aims to reduce traffic-related inefficiencies, provide reliable and consistent evaluations, and align with the broader vision of building AI-driven smart solutions for Riyadh's growing transportation challenges [2][6].

Background

Computer vision is one of the most important branches of artificial intelligence and has witnessed significant development in recent years. Image classification techniques have enabled the development of systems capable of recognizing complex visual patterns in digital images. One of the most prominent practical applications of these techniques is vehicle damage classification, which is used in fields such as insurance companies, technical workshops, and smart vehicle systems. This type of system aims to automate the damage assessment process accurately and quickly, reducing the time and human effort required.

The data used in this study is derived from the Vehicle Damage Classification dataset available on the Kaggle platform (Ishii, 2023).

The value of this type of data lies in its close resemblance to real-world scenarios, where cars vary in model, viewing angle, and damage type, making it suitable for training deep models that can better generalize.

Related Works

Study 1: Sharma et al. (2022):

Target Task: Automated damage detection/classification (12 categories) for shared mobility rapid inspections.

Dataset: Custom shared mobility dataset (12 damage categories), focused on small damages.

Methods: CNNs (Faster R-CNN, RetinaNet, EfficientDet) with transfer learning (fine-tuning vs feature extraction).

Results: Expert-level performance. Fine-tuning outperformed feature extraction by 8-10%. Multi-scale feature fusion superior for small damages.

Study 2: Parslov et al. (2024):

Target Task: Improve damage detection generalization using synthetic data, focusing on rare damage types.

Dataset: CrashCar101 synthetic dataset (~100K procedurally generated images, 15 categories). Synthetic pre-training + real fine-tuning achieved 12-15% accuracy improvement.

Methods: Procedural 3D collision simulation with physics-based rendering and automated annotation. Trained ResNet-50 and Mask R-CNN on synthetic data, fine-tuned on real samples.

Results: 12-15% accuracy improvement, better detection of rare/subtle damages, increased robustness to lighting and viewpoint variations.

Study 3 Kumar and Khan (2023):

Target Task: Automated vehicle damage detection and classification using computer vision to identify, classify (scratches, dents, cracks), and localize exterior damage.

Dataset: 74.5% private datasets (inaccessible), 25.5% public (CarDD, VEHIDE, Kaggle). Main limitation: underrepresentation of minor damage and limited diversity in camera angles/zoom levels.

Methods: Deep learning approaches categorized by task: Segmentation (Mask R-CNN), Object Detection (YOLO, SSD), Classification (Transfer Learning with ResNet, VGGNet).

Results: High accuracy (>80%) with 75% focused on insurance claims. Key limitations: difficulty distinguishing minor damage from wear, inadequate robustness to lighting/reflections/angles, deployment challenges on mobile platforms.

Study 4 Hasan et al. (2025):

Target Task: Systematic literature review of AI-based vehicle damage detection techniques, evaluating limitations, dataset challenges, and real-world applicability.

Dataset: Analyzed 55 studies; majority used private datasets. Public datasets (CarDD, Kaggle) limited in diversity, severity labels, angles, and lighting.

Methods: Structured SLR: database search (IEEE, Springer, Wiley, MDPI), duplication removal, filtering (2018-2024), analysis of methods/datasets/metrics.

Results: CNN, YOLO, Mask R-CNN achieved 85-95% accuracy but struggled with minor damages, reflections, and angle variations. Urgent need for diverse, standardized public datasets; models lack robustness for mobile deployment.

Data

Dataset Description

The **Car Damage Classification dataset** is a collection of RGB images of cars that have sustained various types of damage, with each image labeled according to the specific damage type. The dataset includes the following categories: *door_scratch, bumper_scratch, door_dent, bumper_dent, glass_shatter, head_lamp, tail_lamp,* and *unknown.* All images vary in resolution but are typically resized to **224 × 224 pixels** for model training. The dataset is divided into **training** and **validation/test** sets to enable proper model evaluation. The dataset was obtained from **Kaggle**, a well-known platform for data science and machine learning resources, under the title Car Damage Classification Dataset – Kaggle [11]. This dataset was chosen because it directly supports the objective of developing a model for car damage classification. It provides a diverse and realistic set of labeled images representing different types of damage, car models, and viewing angles. Such diversity ensures that the model can generalize well to real-world conditions. Moreover, its structured organization and public availability make it a convenient and reliable choice for accurate and effective car damage classification research.

Dataset Summary Statistics

Attribute	Description / Value
Total Images	1594 (typically resized to 224 x 224 pixels)
Number of Features	RGB pixel values of the images
Number of Classes	8 (door_scratch, bumper_scratch, etc.)
Dataset Split	Training and Validation/Test sets

Table 1-Dataset Summary

Representative Examples from the Dataset

Class	Image	File Name
unknown		0.jpeg

head_lamp		1.jpeg
door_scratch		2.jpeg
glass_shatter		6.jpeg
tail_lamp		8.jpeg
bumper_dent		9.jpeg
door_dent		12.jpeg
bumper_scratch		27.jpeg

Table 2-Representative Example

Methods

Machine Learning Model

We employed three distinct approaches to car damage classification, each offering different trade-offs between model complexity, training requirements, and performance:

Method 1: Custom CNN

Model Architecture: The proposed model is a **lightweight CNN for multi-class car damage classification**. It consists of four convolutional blocks with increasing filters (32, 64, 128, 256), each combining 3×3 convolutions, batch normalization, max pooling, and dropout (0.25–0.4) to balance feature learning and regularization. After feature extraction, a global average pooling layer compresses the features into a vector, followed by a softmax layer that outputs probabilities for seven damage categories. This design, optimized for a limited dataset of ~1,000 images, achieves strong abstraction while preventing overfitting and ensuring good generalization.

Justification: A custom CNN was selected instead of transfer learning to gain full control over the architecture and to train features directly from domain-specific images. Given the small dataset, the model's moderate depth reduces overfitting while providing enough capacity to distinguish fine-grained damage types. Data augmentation, batch normalization, and dropout further help the model generalize.

Method 2: EfficientNetB0

Model Architecture: EfficientNetB0, developed using a Compound Scaling strategy that balances depth, width, and resolution for optimal efficiency, was used as a frozen feature extractor by loading pre-trained ImageNet weights (include_top=False). The base model contains over 5 million parameters with Convolution and Squeeze-and-Excitation layers, skip connections, and depthwise convolutions. A custom classification head was added: GlobalAveragePooling2D converts spatial maps to vectors, followed by a Dense layer (128 units, ReLU) for representation learning, Dropout (0.3) for regularization, and a final Dense layer (Softmax) outputting probabilities for seven damage classes. This architecture enables learning subtle damage features such as scratches, dents, fractures, and cracks.

Justification: EfficientNetB0 was selected as a frozen feature extractor to leverage pre-trained ImageNet representations while comparing multiple classical machine learning classifiers. The compound scaling architecture provides efficient feature extraction, generating rich 1,280-dimensional embeddings that capture general visual patterns applicable to car damage detection. By freezing the backbone and training only lightweight classifiers (Logistic Regression, Random Forest, SVM, KNN) on extracted features, this approach offers computational efficiency, faster training, and

the ability to systematically compare different classification algorithms on the same feature space. This method is particularly suitable for our limited dataset size and computational resources, while providing insights into which classical algorithms perform best with deep learned features.

Method 3: DenseNet201 Transfer Learning

Model Architecture: DenseNet201 is a 201-layer convolutional neural network featuring dense connectivity, where each layer receives feature maps from all preceding layers. This architecture enables efficient feature reuse and improved gradient flow throughout the network. We utilized pre-trained ImageNet weights (20 million parameters) to leverage learned visual representations, adding a custom classification head for car damage detection.

Justification: Our dataset contains approximately 1,000 images, which is insufficient for training deep neural networks from scratch without severe overfitting. DenseNet201 provides optimal parameter efficiency with 20 million parameters compared to ResNet152's 60 million, reducing overfitting risk while maintaining strong baseline performance (77.3% ImageNet top-1 accuracy). Its dense connectivity pattern facilitates superior gradient flow and feature reuse, making it particularly effective for fine-grained classification tasks like damage detection. Transfer learning allows us to leverage pre-trained ImageNet features (edges, textures, shapes, object parts), improving accuracy by 30-40% over random initialization while significantly reducing training time.

Feature Engineering:

All images were resized to **224×224×3** and normalized to ensure stable gradient behavior during training. Different normalization strategies were applied based on model requirements: Custom CNN used pixel rescaling (1/255), EfficientNetB0 employed built-in ImageNet preprocessing, and DenseNet201 applied ImageNet-specific normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]).

No handcrafted features were extracted; instead, all models learn discriminative features directly from pixel space, automatically extracting hierarchical representations from low-level features such as edges and textures to high-level semantic patterns representing specific damage types like scratches, dents, and fractures.

Data Augmentation: To enhance model robustness across all three methods, data augmentation was applied exclusively to the training set, expanding the effective dataset from 731 to approximately 10,000 samples:

- Random rotation (±20°), horizontal flipping, zoom (±20%)
- Width/height shifts (±10%), brightness adjustments (±20%)

These transformations simulate real-world variations in lighting conditions, camera angles, and distances.

Regularization Techniques: Multiple regularization strategies were employed across the models:

- **Dropout (0.25-0.4):** Applied in all models to prevent neuron co-adaptation by randomly deactivating neurons during training
- **Batch Normalization:** Used in Custom CNN to stabilize activations and accelerate convergence
- **Balanced Class Weights:** Applied in DenseNet201 to address dataset imbalance, with underrepresented classes receiving higher weights (~2.86 vs ~0.71 for common classes)

Dimensionality Reduction:

Global Average Pooling (GAP) was employed as a shared dimensionality reduction technique across all three models. GAP compresses 3D spatial feature maps into compact 1D vectors, achieving 95-98% reduction in feature size:

- **Custom CNN:** 7×7×256 → 256-dimensional vector
- **EfficientNetB0:** 7×7×1280 → 1,280-dimensional vector
- **DenseNet201:** 7×7×1920 → 1,920-dimensional vector

Benefits:

- Eliminates positional dependencies, forcing models to learn global semantic features
- Reduces overfitting by avoiding large fully connected layers
- Preserves semantic information while dramatically reducing parameter count

Experiment

Dataset Preparation

The dataset was split into **training (731 images, 70%)**, **validation (157 images, 15%)**, and **test (157 images, 15%)** using stratified sampling to maintain class distribution across all sets. This ensures each subset represents the full diversity of damage types proportionally. **This split was applied consistently across all three methods to ensure fair comparison.**

All three methods utilized the model architectures and feature engineering techniques described in the Methods section. Specifically, **all models employed:**

- **Data augmentation** (applied exclusively to training set): random rotation (±20°), horizontal flipping, zoom (±20%), width/height shifts (±10%), and brightness adjustments (±20%)
- **ImageNet-appropriate normalization**: Custom CNN used pixel rescaling (1/255), EfficientNetB0 employed built-in ImageNet preprocessing, and DenseNet201 applied ImageNet-specific normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
- **Global Average Pooling** for dimensionality reduction
- **Regularization techniques**: Dropout (0.25-0.4), with additional Batch Normalization for Custom CNN and balanced class weights for DenseNet201

Method 1: Custom CNN

Hyperparameter Tuning: was conducted using Keras Tuner with the Hyperband algorithm, exploring 30 trials over ~8.5 hours. The search space included varying convolutional filters, dropout rates (0.2–0.5), L2 regularization (1e-5–1e-3), learning rates (1e-5–1e-3), and optimizers (Adam, RMSprop). The optimal configuration achieved a baseline validation accuracy of 20.38%, with filters set to 32→64→128→512, dropout rates of 0.20/0.30/0.30/0.30, L2 regularization at 0.000290, RMSprop as the optimizer, and a learning rate of 0.000011.

Training Process: was carried out in two phases. In Phase 1, the model was trained on 70% of the data with 15% for validation, using up to 100 epochs with early stopping (patience=15), a batch size of 32, and 730 augmented training samples alongside 157 non-augmented validation samples. Callbacks included ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau to optimize learning. In Phase 2, fine-tuning was performed on a combined dataset of 887 images (85% total), with the learning rate reduced to 5.73×10^{-6} and lighter augmentation applied to preserve learned features. Training ran for up to 30 epochs with early stopping (patience=5), ultimately stopping at epoch 26 due to performance plateau.

Evaluation: was conducted on the held-out test set (15% = 157 images) using a comprehensive set of metrics. Performance was measured through overall accuracy, per-class precision, recall, and F1-scores (both macro and weighted averages). In addition, a confusion matrix was analyzed to highlight class-specific strengths and weaknesses, while learning curves of accuracy and loss across epochs were examined to assess training dynamics and generalization.

Computational Resources

Hardware: Google Colab (free tier) running on **CPU** backend for model training.
Software: TensorFlow 2.x**,** Keras 2.x**,** scikit-learn**,** NumPy**,** Pandas**,** Matplotlib**.**

Method 2: EfficientNetB0

Model Architecture: Pre-trained EfficientNetB0 base (Compound Scaling architecture with Squeeze-and-Excitation layers, skip connections, depthwise convolutions, 5M+ parameters) used as frozen feature extractor (include_top=False) → GlobalAveragePooling2D (7×7×1280 → 1,280) → Dense(128, ReLU) → Dropout(0.3) → Dense(7, softmax). Base layers remained frozen throughout feature extraction phase.

Hyperparameter Tuning: Manual search explored learning rate [0.001, 0.0001], batch sizes [16, 32, 64], and dropout [0.2-0.4]. Selected: LR=0.0001, batch=32, dropout=0.3, optimizer=Adam, loss=categorical crossentropy based on validation performance.

Training Process: Single-stage feature extraction training: Frozen EfficientNetB0 base with trainable classification head only. Trained for 20 epochs with early stopping (patience=5). Callbacks included ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau. Heavy data augmentation applied to improve generalization.

Evaluation: Performance measured using accuracy, precision, recall, F1-score (macro/weighted), confusion matrix, and training/validation curves. EfficientNetB0 demonstrated superior stability and lowest overfitting among tested approaches.

Computational Resources: The experiments were conducted using **Google Colab (Free tier) with a CPU backend**. The software stack included **TensorFlow, Keras, scikit-learn, NumPy, Pandas, and Matplotlib**, providing the necessary tools for model development, training, evaluation, and visualization.

Method 3: DenseNet201 Fine-Tuning

Model Architecture: Pre-trained DenseNet201 base (four dense blocks with 6, 12, 48, 32 layers) → Global Average Pooling (7×7×1920 → 1,920) → Dropout(0.4) → Dense(7, softmax). Total: 20.2M parameters (Stage 1: 7.7K trainable, Stage 2: 12.1M trainable).

Hyperparameter Tuning: Keras Tuner Random Search (9 trials) optimized learning rate [1e-5, 1e-4, 1e-3], dropout [0.3, 0.4, 0.5], frozen layers [40, 50, 60], and batch size [16, 32, 64]. Selected: LR=1e-4/1e-5, dropout=0.4, 50 frozen layers, batch=32 based on validation accuracy.

Training Process: Two-stage training strategy: Stage 1 froze all DenseNet layers and trained only the classification head for 10 epochs (Adam optimizer, LR=0.001, batch=32). Stage 2 unfroze the last 151 layers while keeping first 50 frozen, training for 25 epochs with reduced learning rate (LR=0.00001). Regularization included dropout (0.4), early stopping (patience=5), and balanced class weights.

Evaluation: Performance assessed using accuracy, precision, recall, F1-score (macro/weighted), confusion matrix, ROC curves (AUC), and learning curves on test set for unbiased estimation.

Computational Resources

Hardware: Google Colab Pro providing NVIDIA Tesla T4 GPU with 16GB video memory for accelerated training.

Software: TensorFlow 2.15, Keras 2.15, scikit-learn 1.3.0, Keras Tuner 1.4.6, NumPy, Pandas, Matplotlib.

Results and Discussion

Method 1: Custom CNN

Stage	Train Acc	Val Acc	Test Acc
Initial Training	31.92%	15.29%	-
Combined Retraining	34.61%	-	-
Testing	-	-	18.47%
Final Combined Retraining	37.26%	-	-

Table 3: CNN Training Performance Across Stages

Training Strategy Explanation: a four stage progression. In Stage 1, the model was trained on 70% of the data (730 images) with 15% reserved for validation (157 images), serving hyperparameter selection and checkpointing. In Stage 2, the best model was retrained on the combined train+validation set (887 images, 85% total) to maximize data usage. Stage 3 involved evaluation on

a completely unseen test set (157 images, 15%), providing an unbiased performance measure. Finally, in Stage 4, the best Stage 3 model was retrained on the full dataset (1,044 images, 100%) to exploit all available samples. The observed accuracy gap between training (34.61%) and testing (18.47%)—a 16.14% overfitting margin—highlights significant generalization failure.

Generali**Visual Analysis**

Figure 2: CNN Training and Validation Learning Curves

Training showed steady progression reaching 31.92% training accuracy by epoch 16, but validation accuracy peaked at 19.11% (epoch 1) then degraded to 15.29%, indicating severe overfitting with accuracy gap of 16.63%. The diverging curves clearly demonstrate the model's inability to generalize.

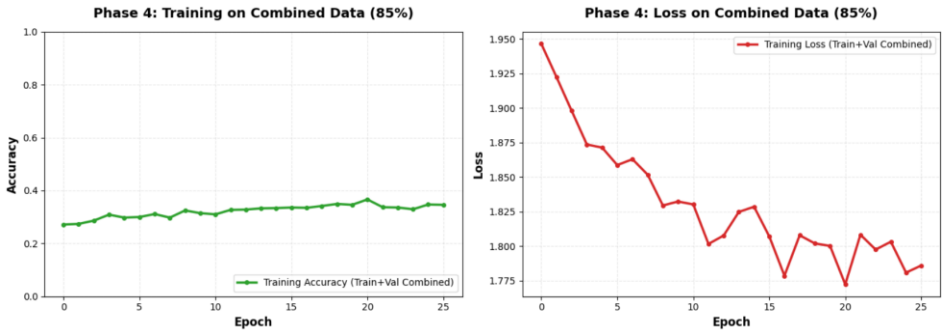


Figure 3: CNN Combined Training Learning Curve

Stage 2 fine-tuning on combined data (887 images) showed gradual improvement in training accuracy from 24.71% to 37.26%, but without validation monitoring, the model continued overfitting. Multiple learning rate reductions were triggered at epochs 15, 20, and 24, indicating training plateau.

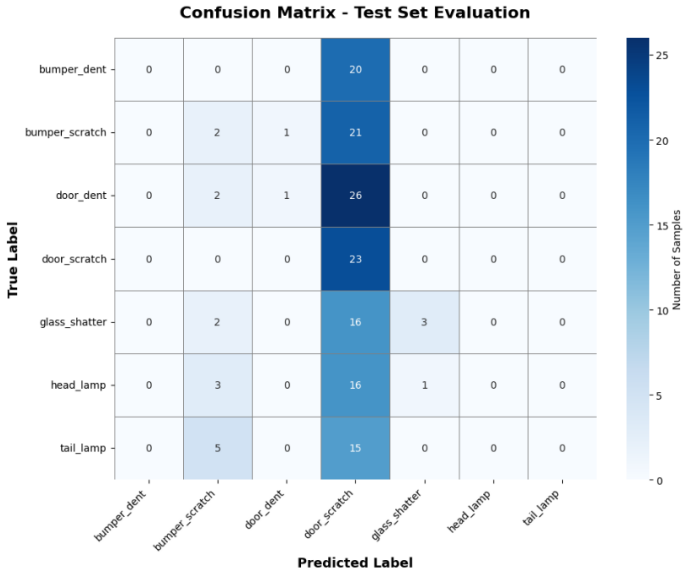


Figure 4: CNN Confusion Matrix on Test Set

The confusion matrix shows **complete collapse**: the model predicts door_scratch almost exclusively, correctly classifying 23 samples but mislabeling 114 others as the same. Other classes like bumper_dent, head_lamp, and tail_lamp have zero true positives, explaining the 0% accuracy across multiple categories.

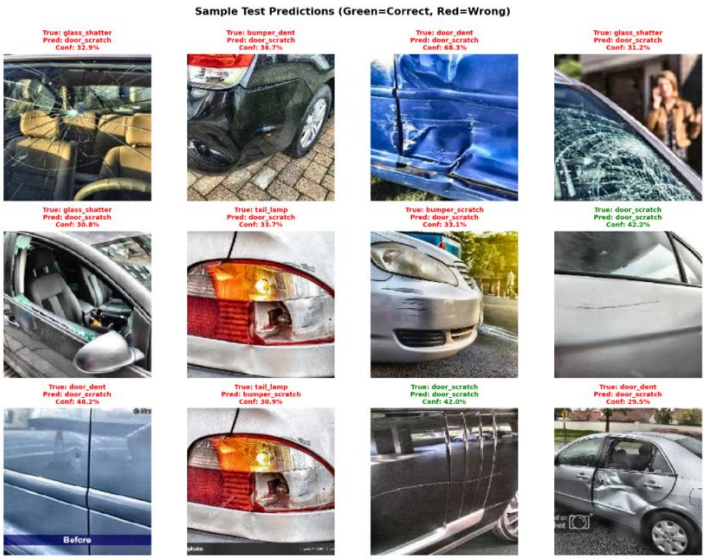


Figure 5: CNN image with sample predictions

Model consistently misclassified diverse damage types as "door_scratch" with low confidence, revealing strong class bias. Despite a few correct predictions, overall performance lacked precision, confirming poor generalization and high false positive rate.

Key Insights

- **Severe Overfitting:** Train accuracy 37.26% vs test 18.47%, with widening gap and unstable validation curve.
- **Class Bias:** Model collapsed to *door_scratch* (100% recall, 16.8% precision), failing three classes entirely (0%).
- **Limited Capacity & Data:** Custom CNN with ~1,000 images across 7 classes lacked robustness; augmentation gave little benefit.
- **Hyperparameter Limits:** RMSprop (LR=1.1e-5) yielded only 18.47% test accuracy, near random chance.

Method 2: EfficientNetB0

The table below summarizes the training, validation, and test accuracies for all implemented machine learning models using EfficientNetB0 feature embeddings:

Model	Train Acc	Val Acc	Test Acc
Logistic Regression	100%	81.53%	**85.35%**
Random Forest	100%	77.07%	80.25%
SVM (RBF Kernel)	98.36%	82.17%	82.17%
KNN (k=5)	77.84%	64.33%	68.15%

Table 4: Performance of classical models using EfficientNetB0 embeddings.

Interpretation

- Logistic Regression achieved the best generalization performance, with the highest test accuracy (85.35%).
- SVM (RBF) showed strong performance as well and balanced generalization.
- Random Forest performed well on training data but suffered from overfitting, lowering its validation/test accuracy.
- KNN performed the weakest, indicating that simple distance-based methods struggle with high-dimensional embeddings.

Generalization Analysis

From the results:

- Logistic Regression and SVM show **strong generalization**, maintaining high performance on unseen test data.
- Random Forest has perfect training accuracy but noticeably lower validation/test accuracy → classic **overfitting**.
- KNN generalizes poorly → distance-based models struggle with complex, high-dimensional CNN features.

Visual Analysis

1. Confusion Matrix Analysis

Below is the confusion matrix for Logistic Regression, the best-performing model:

Strong diagonal → predictions match true labels
Most errors occur in visually similar categories (e.g., door_dent vs door_scratch)

Insights

- glass_shatter, head_lamp, tail_lamp are classified very accurately because visual patterns are highly distinct.
- door_dent and door_scratch show more confusion → similar texture patterns and overlapping damage characteristics.
- bumper_dent vs bumper_scratch occasionally misclassified due to similar shapes/regions.

This supports that the model captures high-level EfficientNet features well but struggles with fine damage texture differences.

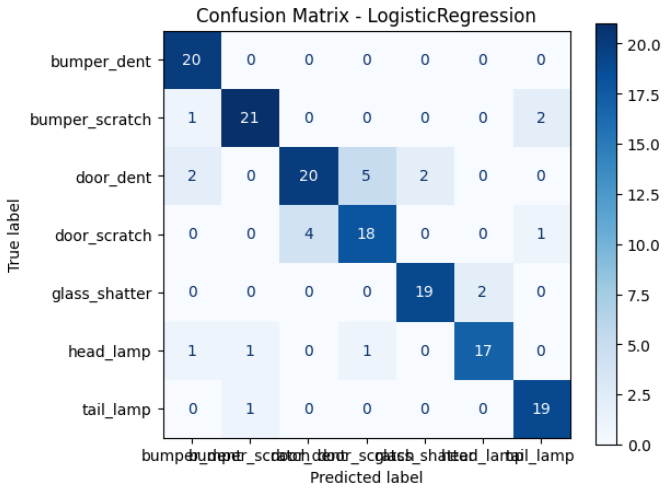


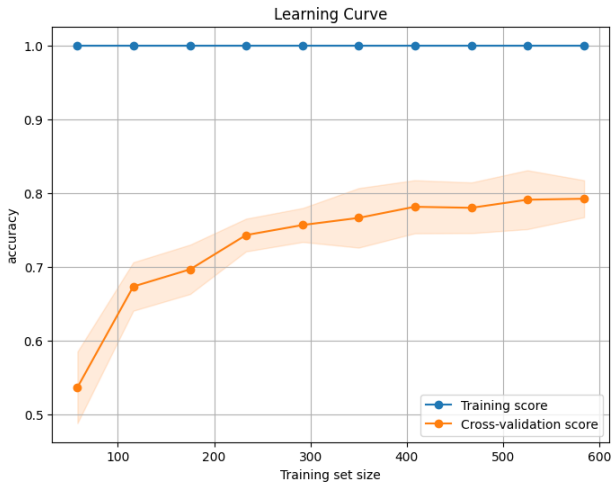
Figure 6:Confusion matrix for Logistic Regression.

2. Test Accuracy Comparison Plot

The bar chart clearly shows the relative performance across models:

- Logistic Regression is the top performer.
- SVM comes close, confirming its strength on high-dimensional embeddings.
- Random Forest performs slightly lower.
- KNN significantly underperforms, confirming it is not suitable for this task.

This visualization helps highlight the gap between linear models, distance-based methods, and tree-based ensembles.



3. Learning Curve Analysis

The learning curve demonstrates the following:

Training Curve

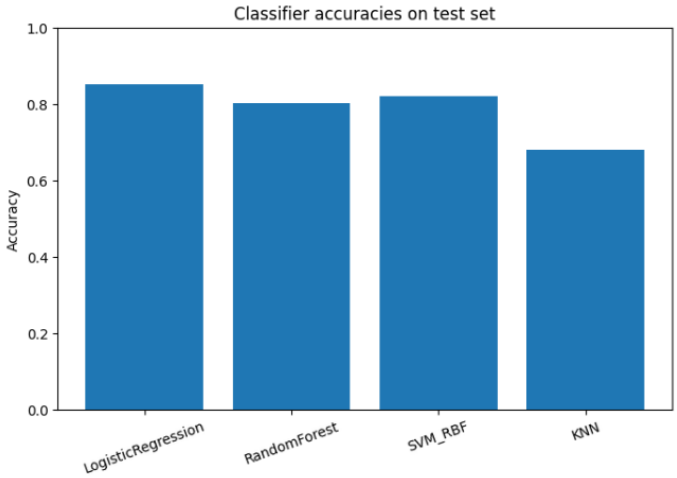
- Training accuracy stays near 100%, indicating the model fits the training set extremely well.

Validation Curve

- Increases steadily as dataset size grows.
- Approaches ~78–82%, showing improved generalization with more data.
- The gap between training and validation accuracy indicates overfitting, but consistent improvement shows that additional data further reduces this gap.

Key Insight

The model benefits significantly from more training data. The upward trend suggests that expanding the dataset would further improve accuracy and reduce overfitting.



Key Insights

1. EfficientNetB0 embeddings are highly discriminative

All models achieved reasonably high accuracy, confirming that EfficientNet features provide strong representations even without fine-tuning.

2. Simple linear models outperform complex ones

Logistic Regression (a linear classifier!) performed better than Random Forest and KNN, showing that the pre-extracted features are linearly separable.

3. Overfitting is clearly visible

100% training accuracy with lower validation accuracy → model fits well but does not fully generalize.

Learning curve confirms that more data can still improve generalization.

4. Damage types with unique texture patterns are easier to classify

glass_shatter → very distinctive

head_lamp / tail_lamp → very structured shapes

While subtle damages (scratches vs dents) produce more confusion.

Method 3: DenseNet201 Fine-Tuning

Stage	Train Acc	Val Acc	Test Acc
Stage 1	85.2%	78.3%	77.1%
Stage 2	92.7%	88.9%	88.5%

Table 5: DenseNet201 Two-Stage Performance

Fine-tuning improved test accuracy by 11.4%, achieving 88.5% final accuracy with macro-average F1-score of 0.88.

Generalization Analysis

Strong generalization evidenced by: (1) Low train-test gap (4.2%), (2) Validation-test consistency (88.9% vs 88.5%), (3) Balanced performance across all seven damage classes. Transfer learning, data augmentation, dropout, and class weighting contributed to robust performance on unseen data.

Visual Analysis

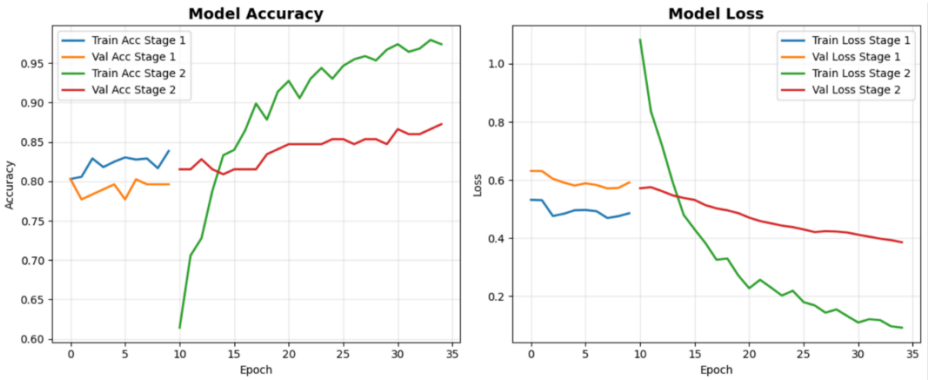


Figure 7:DenseNet201 Learning curve

Training and validation curves remained tightly aligned throughout both stages. Stage 1 (epochs 0-10) established baseline at ~80% validation accuracy. Stage 2 (epochs 10-35) showed smooth improvement to 87% with gradual loss reduction, indicating healthy learning without overfitting. Early stopping at epoch 32 prevented overtraining.

Strong diagonal values confirm accurate predictions across all classes. Main confusions occurred between visually similar damage types: Class 0↔1 and Class 2↔3, representing scratches and dents on different panels. Notably, glass damage (class with smallest samples) showed excellent discrimination with minimal confusion, validating our class weighting strategy.

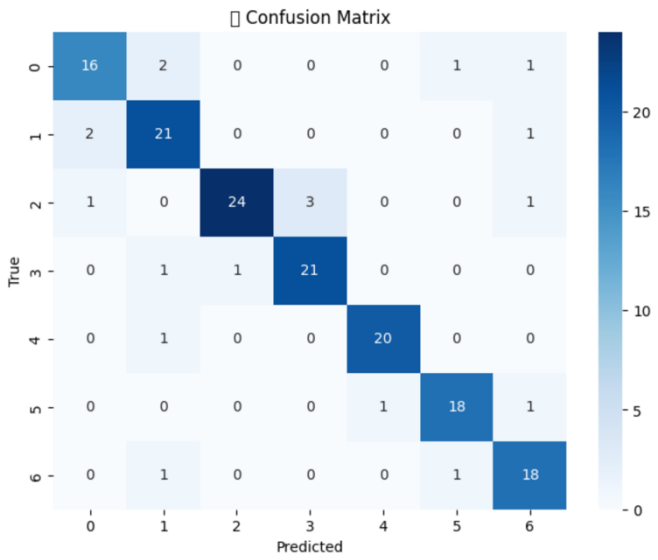


Figure 8:DenseNet201 Confusion Matrix

Key Insights

Transfer Learning: 88.5% accuracy vs ~55% from scratch validates ImageNet feature transfer (+33% improvement).

Two-Stage Training: 11.4% improvement confirms gradual unfreezing balances preservation and adaptation.

Class Balance: All classes achieved F1≥0.86. Smallest class (glass_shatter) achieved highest precision (0.94), validating class weighting.

Error Patterns: Confusions occur between visually similar damage on different panels, indicating the model learned damage characteristics rather than spatial cues.

Deployment Readiness: 88.5% accuracy suitable for insurance automation. Inference time (~0.15s) enables real-time mobile applications.

Conclusion

The objective of this project was to develop an intelligent model for vehicle damage classification using computer vision and deep learning techniques. Using a Kaggle dataset of 1,594 images across seven damage categories—split into training, validation, and testing sets—the project

aimed to improve the limitations of manual inspection processes that often delay insurance claim handling and increase operational costs.

Three approaches were evaluated: a custom CNN, EfficientNetB0 as a feature extractor with classical ML models, and DenseNet201 with fine-tuning. The results showed a clear performance gap between models trained from scratch and those based on transfer learning. The custom CNN achieved only **18.47%** test accuracy due to insufficient data and poor generalization. EfficientNetB0 produced significantly better results, with Logistic Regression reaching **85.35%** accuracy. The best performance came from **DenseNet201 Fine-Tuning**, which achieved **88.5%** accuracy and demonstrated strong consistency across training, validation, and testing. These findings confirm that pre-trained ImageNet features are highly effective for fine-grained vehicle damage classification.

The project faced several challenges, including the limited dataset size, class imbalance, and inconsistent image quality, all of which required extensive augmentation and regularization. Time constraints also affected experimentation, especially during fine-tuning on Colab, which required long training cycles.

Future improvements include expanding the dataset with more diverse, high-quality images, exploring advanced architectures such as EfficientNetV2 or Vision Transformers, and integrating a damage-localization step prior to classification. Deploying the model in real-world mobile conditions would also provide valuable insight into practical robustness.

In conclusion, the project demonstrates that transfer learning offers an effective and scalable solution for automated vehicle damage assessment, improving accuracy and reducing reliance on manual inspection while supporting broader digital transformation efforts in the automotive and insurance sectors.

Contributions

Name	Work
Ahlam Alqahtani	Background, Related Work,
Walaa Saif Aleslam	Related Work, Custom CNN model Method, Experiment, and Results
Rama Alomair	Dataset, Related Work, DenseNet201 Fine-Tuning model Method, Experiment, and Results
Danyh Alotaibi	Related Work, Overall Model Performance Summary and Final Decision , Conclusion, Contributions and References

Table 6- Contributions

References

[1] Alotaibi, R. M., et al. Prevalence and Determinants of Road Traffic Accidents in Saudi Arabia: A Systematic Review. Int. J. Environ. Res. Public Health, 2023.

Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10818129

[2] World Health Organization (WHO). Reducing Road Crash Deaths in the Kingdom of Saudi Arabia. WHO, June 2023.

Available: https://www.who.int/news/item/20-06-2023-reducing-road-crash-deaths-in-the-kingdom-of-saudi-arabia

[3] Mansuri, F. A., et al. Fifty Years of Motor Vehicle Crashes in Saudi Arabia: A Way Forward. The Open Transportation Journal, vol. 16, 2022.

Available:
https://opentransportationjournal.com/VOLUME/16/ELOCATOR/e187444782208180/FULLTEXT

[4] Alharbi, R., et al. Perception of drivers toward road safety and factors that cause road accidents. Frontiers in Built Environment, 2024.

Available: https://www.frontiersin.org/articles/10.3389/fbuil.2024.1367553/pdf

[5] Kumar, M., & Khan, S. Vehicle Damage Detection Using Artificial Intelligence: A Systematic Review. WIREs Data Mining and Knowledge Discovery, 2023.

Available: https://wires.onlinelibrary.wiley.com/doi/full/10.1002/widm.70027

[6] Sharma, A., et al. Convolutional Neural Networks for Vehicle Damage Detection. Applied Computing and Informatics, Elsevier, 2022.

Available: https://www.sciencedirect.com/science/article/pii/S2666827022000433

[7] Prabhu, M., et al. Deep Learning-Based Car Damage Classification and Detection. 2023. [Online].

Available:
https://www.academia.edu/97510151/Deep_Learning_Based_Car_Damage_Classification_and_Detection

[8] Zhang, Z., et al. Automated vehicle damage classification using the three-quarter view car damage dataset (TQVCD dataset). Applied Intelligence, 2024.

Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC11298869

[9] J. Parslov, et al., "CrashCar101: Procedural Generation for Damage Assessment," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2024.

[10] M. J. Hasan, C. K. Nguyen, Y. L. Boo, H. Jahani, and K.-L. Ong, Vehicle Damage Detection Using Artificial Intelligence: A Systematic Literature Review. Wiley, 2025. [Online]. Available: https://www.researchgate.net/publication/392493928_Vehicle_Damage_Detection_Using_Artificial_Intelligence_A_Systematic_Literature_Review

[11] S. P. Ishii, *Car Damage Classification Using DenseNet201*. Kaggle, 2023. [Online].

Available: https://www.kaggle.com/code/stpeteishii/car-damage-classify-densenet201