CAR DAMAGE DETECTOR

Automated Car Damage Classification from Images

Problem Statement

- Assessing car damage from images is challenging due to variations in lighting, angles, and vehicle positions, making manual inspection subjective and inconsistent.
- Traditional visual assessments by inspectors are time-consuming, prone to human error, and lack standardization, which can delay claims processing or resale evaluations.
- There is a critical need for an automated, objective system that can classify vehicle damage reliably and consistently across diverse car images.
- A deep learning-based approach enables scalable, fast, and repeatable assessments, reducing processing times and improving accuracy in estimating repair costs or validating insurance claims.

Project Objectives

- Automatically classify car damage into one of six categories: Front Normal, Front Breakage, Front Crushed, Rear Normal, Rear Breakage, Rear Crushed.
- Achieve at least 75% accuracy on a validation dataset of labeled car images.
- Provide an interactive Streamlit web app allowing users to upload an image and instantly view the predicted damage class.
- Support fair and transparent claim settlements by providing objective, evidence-based damage assessments directly from uploaded images.

Dataset Overview

- The dataset consists of approximately 2300 labeled images of vehicles, carefully collected to capture various types of front and rear damage scenarios.

- Images are distributed across six predefined damage categories to ensure balanced learning:
    - Front Breakage (FB): 500 images
    - Front Crushed (FC): 400 images
    - Front Normal (FN): 500 images
    - Rear Breakage (RB): 300 images
    - Rear Crushed (RC): 300 images
    - Rear Normal (RN): 300 images
- The dataset includes images taken under different lighting conditions, angles, and backgrounds, with a focus on third-quarter front or rear views to mimic real-world scenarios encountered during insurance inspections or resale evaluations.

DATA PREPROCESSING

- **Image Transformations (Data Augmentation & Standardization)**
    - RandomHorizontalFlip(): Introduces horizontal flips to improve model robustness and generalization to different car orientations.
    - RandomRotation(10): Rotates images randomly up to ±10 degrees to add rotational invariance and simulate real-world camera angles.
    - ColorJitter(brightness=0.2, contrast=0.2): Varies brightness and contrast to mimic diverse lighting conditions and enhance model resilience.
    - Resize((224, 224)): Rescales all images to a consistent 224×224 resolution required by the CNN architecture.
    - ToTensor(): Converts images into PyTorch tensors, scaling pixel values to the [0,1] range for numerical stability.
    - Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]): Standardizes pixel values using ImageNet statistics—critical when leveraging pre-trained CNNs.

Data Splitting

- Training Set: 75% of the dataset (1,725 images) used to optimize model weights during training.
- Validation Set: 25% of the dataset (575 images) reserved to evaluate model performance and detect overfitting.

Data Loaders

- Data Loader Objects: Created for both the training and validation datasets to efficiently load mini-batches (batch size 32).
- Features: Include shuffling of training data to improve model generalization and multi-process data loading for faster throughput.

Model Development & Optimization

- **Baseline CNN Model**
  - Started with a simple CNN architecture.
  - Achieved an initial accuracy of 61.22%, providing a performance benchmark for further improvements.

- **Regularization Techniques Applied**
  - Introduced Batch Normalization to stabilize and accelerate training.
  - Added Dropout Regularization to reduce overfitting.
  - Incorporated L2 Regularization to improve generalization.
  - Together, these boosted accuracy to 56%.

- **EfficientNet-B0 Implementation**
  - Transitioned to a more advanced EfficientNet-B0 model.
  - Leveraged compound scaling to optimize parameters efficiently.
  - Resulted in a notable performance improvement with 71.13% accuracy.

- **Adoption of ResNet50 Architecture**
    - Adopted the deeper, residual-based ResNet50 network.
    - Enabled better learning of complex damage patterns.
    - Significantly increased accuracy to 77.74%.

- **Hyperparameter Tuning with Optuna**
    - Performed extensive tuning of learning rate and dropout probability.
    - Used the Optuna framework for efficient hyperparameter search.
    - Final ResNet50 model achieved 77.74% accuracy, representing the best-performing solution.

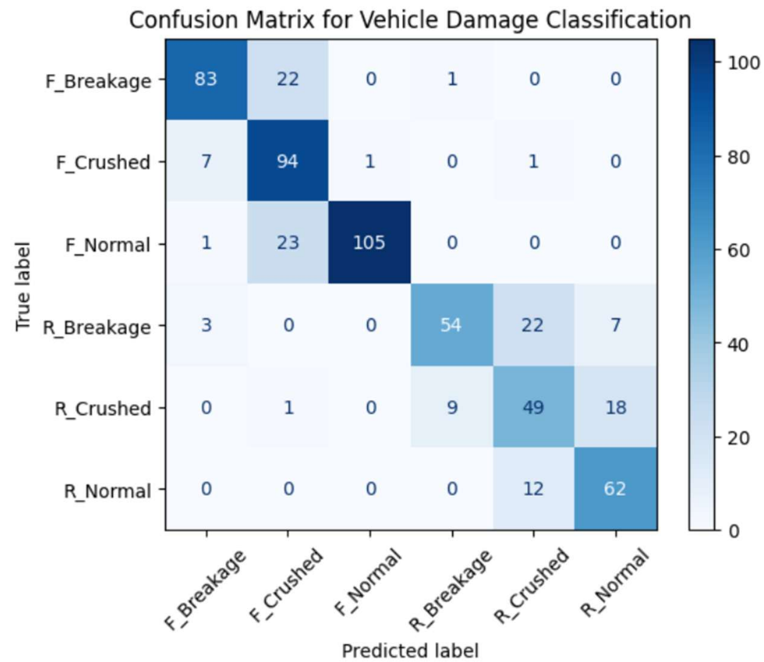
MODEL EVALUATION

- **Classification Report**

    - Generated precision, recall, and F1-score for each damage class.
    - Report included overall accuracy > 75% after hyperparameter tuning, meeting the project objective.
    - Demonstrated substantial improvement over initial baseline models.

	precision	recall	f1-score	support
0	0.88	0.78	0.83	106
1	0.67	0.91	0.77	103
2	0.99	0.81	0.89	129
3	0.84	0.63	0.72	86
4	0.58	0.64	0.61	77
5	0.71	0.84	0.77	74
accuracy			0.78	575
macro avg	0.78	0.77	0.77	575
weighted avg	0.80	0.78	0.78	575

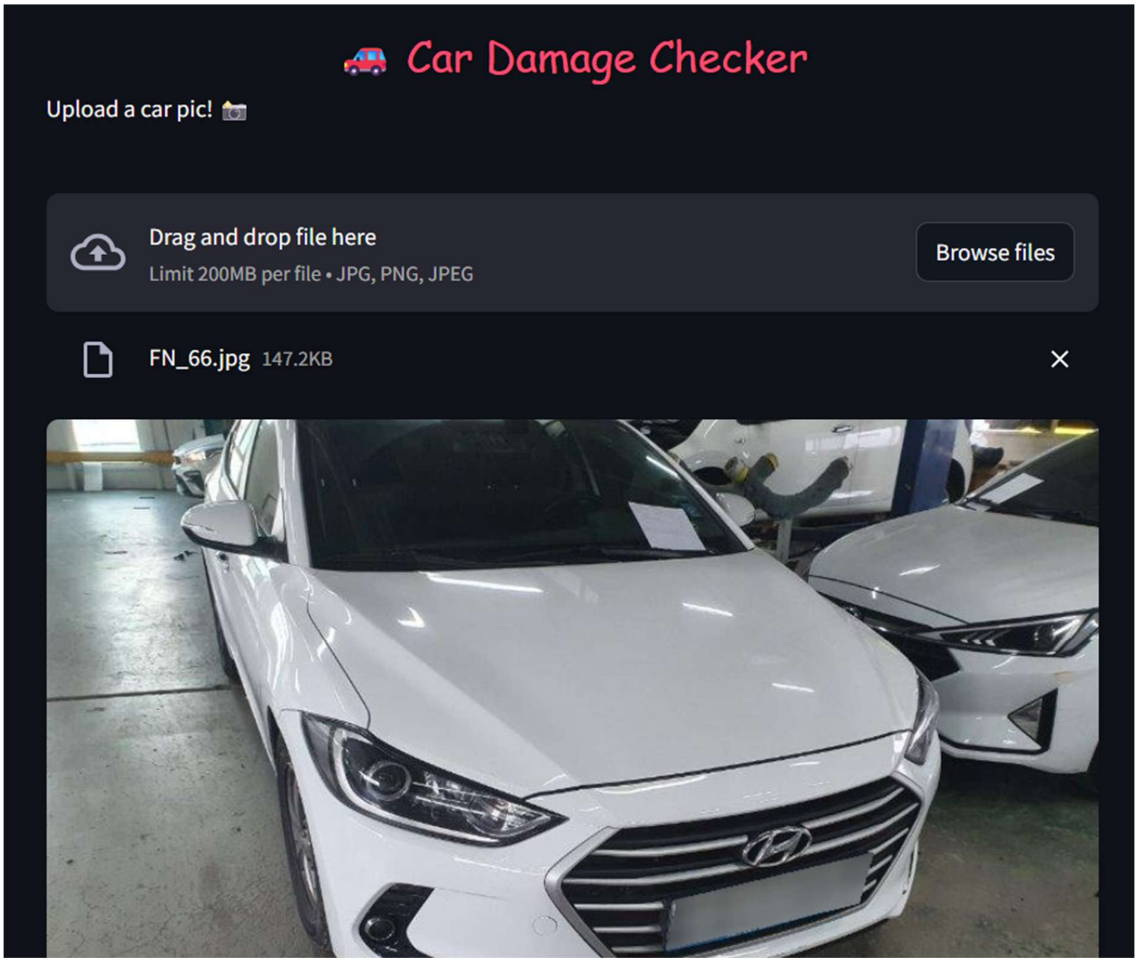
- **Confusion Matrix**
  - Visualized actual vs. predicted labels across six classes.
  - Identified patterns of misclassification between similar damage classes.

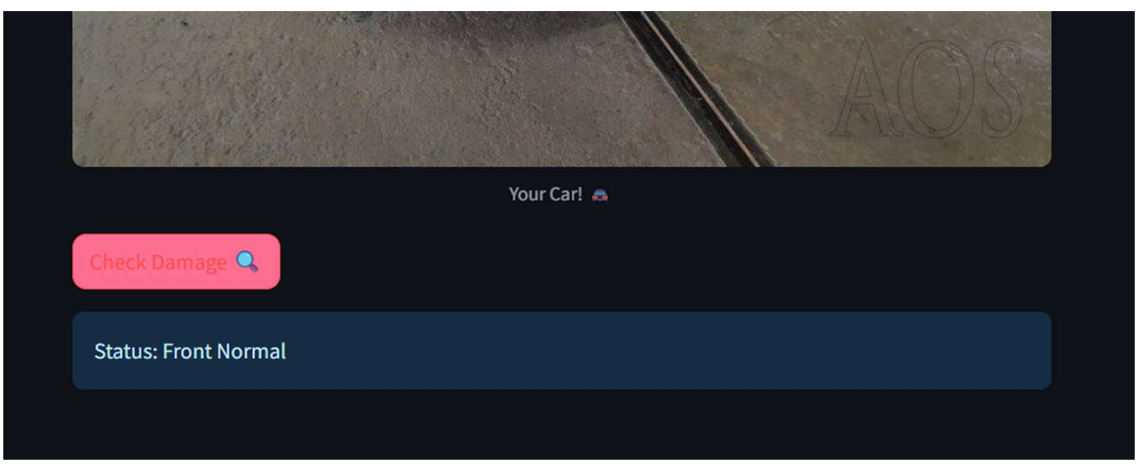


Streamlit App Integration

- Developed an interactive web application using Streamlit for real-time car damage classification.
- Integrated the trained ResNet50 model and preprocessing for seamless predictions.
- Allows users to upload a car image (preferably third-quarter front or rear view) directly through the app's intuitive interface.
- On clicking "Predict Damage Class," the app provides:
  - Predicted damage category (e.g., Front Breakage, Rear Normal).
  - Visual display of the uploaded image alongside the prediction.
- Handles all preprocessing within the app (resizing, normalization) to ensure consistent and reliable model inference.
- Deployed the app on Streamlit Cloud (streamlit.io) for public access.
- Designed for business and non-technical users (e.g., insurers, fleet managers) to enable quick, consistent, and objective car assessments.

User Interaction Preview





Project Summary

- Built a deep learning system to detect and classify car damage from images into six categories: Front/Rear – Normal, Breakage, Crushed.
- Prepared a dataset of 2300 images with augmentations for better generalization.
- Split dataset into 75% training and 25% validation sets, ensuring balanced class representation for accurate evaluation.
- Trained baseline CNN, EfficientNet-B0, and ResNet50 with batch norm, dropout, and L2; tuned ResNet50 with Optuna, achieving 77.74% final accuracy.
- Evaluated results using classification reports and confusion matrices, identifying key misclassification patterns.
- Created a Streamlit app for real-time car damage classification from uploaded images.
- Live App: https://rakesh-project-car-damage-detector.streamlit.app/
- GitHub: https://github.com/rakeshkapilavayi/Car-Damage-Detector