“Damage Car Detector”

“An CNN based model segregates the damaged car from not damaged car”

Dann Berlin
3rd Year
B.tech/Artificial Intelligence & Data Science

Meresh
3rd Year
B.tech/Artificial Intelligence & Data Science

Steve Aaron
3rd Year
B.tech/Artificial Intelligence & Data Science

Thomas Allwin
3rd Year
B.tech/Artificial Intelligence & Data Science

Abstract

In the insurance and automotive industries, efficient and accurate damage detection from car images is a critical step in streamlining repair estimations, claim settlements, and quality control. This project proposes an automated image preprocessing and classification system that identifies whether a car is damaged or not using deep learning. Utilizing computer vision techniques with OpenCV and TensorFlow, our system classifies uploaded car images into 'Damaged' or 'Not Damaged' categories, enabling faster decision-making processes.

Problem Statement

Manual inspection of car damages is labor-intensive, subjective, and prone to error. Insurance companies often face delays and inconsistencies when verifying damage through images submitted by users. There is a need for an automated, consistent, and scalable solution to classify damaged car images quickly and reliably.

Existing System

Existing solutions typically involve:

- Manual inspection of images by human agents
- Traditional machine learning models requiring feature engineering
- Limited use of real-time web-based or AI-powered apps for classification
- Limitations include human error, lack of scalability, delayed claim processing, and high operational costs.

Proposed System

We propose a deep learning-based solution using Convolutional Neural Networks (CNNs) trained on car image datasets to classify images as 'Damaged' or 'Not Damaged'. The system is integrated with a web-based front end using Streamlit, allowing users to upload images and receive instant classification results. The use of OpenCV ensures high-quality preprocessing and normalization of images before prediction.

Methodology

1. **Dataset Preparation**: Car images are collected and labeled as 'Damaged' or 'Not Damaged'. Images are resized to 256x256 pixels and normalized. Downloaded using "icrawler" module.

2. **Model Architecture**:

- Input Layer: 256x256x3
- Convolutional layers with ReLU activation and MaxPooling
- Flatten layer
- Dense layers with Dropout for regularization
- Output layer with Sigmoid activation for binary classification

3. **Training and Validation**: The model is trained with binary cross-entropy loss using Adam optimizer.
4. **Integration**: The trained model is integrated with a Streamlit-based web application.

System Implementation

- **Backend**: TensorFlow and Keras for CNN model implementation.
- **Image Processing**: OpenCV for resizing and normalization.
- **Frontend**: Streamlit for an interactive web interface.
- **Deployment**: Local or server-based deployment with options for future cloud deployment.

Architecture

User -> Streamlit Web Interface -> Preprocessing (OpenCV) -> Trained CNN Model (TensorFlow) -> Classification Result

Mathematical approach:

1. Convolution layer:

The core operation in CNN is convolution. For a given input image II and kernel KK:

$$S(i, j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} W(i - m, j - n) \cdot X(m, n)$$

Where:

- S(i, j) is the output.
- N is the size of the kernel.\
- X(m,n) is Imput Image
- W(i-m, j-n) is kernel flipping over the image

2. ReLU activation function:

$$ReLU(x) = \max(0, x)$$

Purpose: Adds non-linearity to the model and helps it learn complex patterns.

3. Pooling Layer:

Max pooling downsamples the feature map:

$$P(i, j) = \max_{m,n \in \text{window}} F(i + m, j + n)$$

Where:

P(i, j) is the pooled output,

F is the input feature map.

4. Flattening layer:

This step reshapes the 3D tensor output of the last pooling/convolutional layer into a 1D vector:

$$\text{Flatten}(x) = x_1, x_2, ..., x_n$$

Used before passing into Dense (Fully Connected) layers.

5. Dense layer (Fully Connected layer):

In this layer, every neuron from the flattening layer summed to a single

value (Z) after that softmax (f(z)) will be applied to the resultant output (Z)

$$z = \sum W_i \cdot x_i + b$$

$$a = f(z)$$

$$s(x) = \frac{1}{1 + e^{-\lambda x}}$$

Where:

- x is the input vector,
- W is the weight matrix,
- b is the bias vector,
- f is the activation function (ReLU or Sigmoid).

Used for binary classification output between 0 and 1 ("Not Damaged" or "Damaged").

6. Dropout:

Dropout randomly disables neurons during training to prevent overfitting:

$$x = \begin{cases} 0 & \text{with probability } p \\ \frac{x}{1-p} & \text{with probability } 1-p \end{cases}$$

Where p is the dropout rate.

Results

- Achieved over 90% accuracy on a test set of car images
- Real-time prediction through the web interface
- Demonstrated significant time savings in damage identification
- The Test image uploaded in the Web interface will be analysed by the CNN model.

Conclusion

The Damage Car Detector system automates the process of identifying vehicle damage using deep learning and computer vision. It enhances the speed and accuracy of insurance claim processing and quality control. With further training on diverse datasets and integration with real-time claim systems, this solution can greatly benefit the automotive and insurance sectors.

Output