Car Damage Detection with Explainable AI

python 3.8+ PyTorch 2.0+ Streamlit 1.28+ License MIT

A comprehensive deep learning system for automated car damage detection using instance segmentation with explainable AI features. This project compares four different architectures and provides pixel-level damage localization with visual explanations.

Features

- **Multi-Architecture Comparison**: DenseNet121, EfficientNet-B3, InceptionV3, and Mask R-CNN
- **Instance Segmentation**: Pixel-precise damage localization using Mask R-CNN
- **Explainable AI**: Grad-CAM and Occlusion Sensitivity Analysis
- **Real-time Web Interface**: Interactive Streamlit dashboard
- **Comprehensive Evaluation**: Detailed performance metrics and visualizations
- **Production Ready**: Optimized for deployment with confidence thresholding

Installation

Prerequisites

- Python 3.8 or higher
- CUDA-compatible GPU (optional but recommended)
- Git

Step 1: Clone Repository

```
git clone https://github.com/Astrasv/Car-Damage-Detection-MaskRCNN
cd car-damage-detection
```

Step 2: Create Virtual Environment

```
# Using conda (recommended)
conda create -n car-damage python=3.11
conda activate car-damage

# Using venv
python -m venv car-damage-env
source Car_Damage_Detection_MASKRCNN/bin/activate  # On Windows: car-damage-env\Scripts\activate
```

Step 3: Install Dependencies

```
# Install PyTorch (check https://pytorch.org for your CUDA version)
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

# Install other requirements
pip install -r requirements.txt
```

Requirements File Contents

```
torch>=2.0.0
torchvision>=0.15.0
torchaudio>=2.0.0
streamlit>=1.28.0
opencv-python>=4.8.0
Pillow>=10.0.0
numpy>=1.24.0
matplotlib>=3.7.0
albumentations>=1.3.0
pycocotools>=2.0.7
scikit-image>=0.21.0
tqdm>=4.65.0
```

📁 Dataset Setup

Dataset Structure

Organize your dataset in the following structure:

```
dataset/
├── train/
│   ├── images/
│   │   ├── image1.jpg
│   │   ├── image2.jpg
│   │   └── ...
│   └── via_region_data.json
├── val/
│   ├── images/
│   │   ├── val_image1.jpg
│   │   ├── val_image2.jpg
│   │   └── ...
│   └── via_region_data.json
└── test/
    ├── images/
    │   ├── test_image1.jpg
    │   ├── test_image2.jpg
    │   └── ...
    └── via_region_data.json
```

Creating Annotations

1. **Using VIA (VGG Image Annotator)**:

   ```
   # Download VIA from: https://www.robots.ox.ac.uk/~vgg/software/via/
   # Use polygon tool to annotate damage regions
   # Export annotations as JSON
   ```

2. **Annotation Guidelines**:
   - Use polygon annotations for precise damage boundaries
   - Label damage types consistently
   - Ensure all images have corresponding annotations
   - Validate annotations using our provided scripts

Validate Dataset

```
python src/dataset.py --validate --data_dir dataset/train
python src/dataset.py --validate --data_dir dataset/val
```

🚀 Quick Start

Option 1: Using Pre-trained Model

```
# Download pre-trained model (if available)
wget https://your-model-url/best_model.pth -O models/best_model.pth

# Run the web application
streamlit run app.py
```

Option 2: Training from Scratch

```
# Test dataset and training setup
python test/test_training.py

# Start training
python test/train_robust.py

# Run the application
streamlit run app.py
```

□ Training Models

Basic Training

```
# Train Mask R-CNN (recommended)
python src/train.py

# Train with custom parameters
python train_robust.py --epochs 50 --batch_size 4 --lr 0.001
```

Advanced Training Options

```
# Simplified training (fewer epochs for testing)
python train_simple.py

# Robust training with error handling
python train_robust.py --num_epochs 25 --batch_size 2
```

Training Parameters

Parameter	Default	Description
`--epochs`	10	Number of training epochs
`--batch_size`	2	Batch size for training
`--lr`	0.005	Learning rate
`--data_dir`	'dataset'	Path to dataset directory

Monitor Training

```
# View training progress
tensorboard --logdir runs/

# Check training curves
ls *.png  # training_curves.png will be generated
```

□ Running the Application

Start Web Interface

```
streamlit run app.py
```

The application will be available at `http://localhost:8501`

Web Interface Features

1. □ **Damage Detection Tab**:

- Upload car images (JPG, PNG)
- Adjust confidence threshold
- View detection results with bounding boxes and masks
- Download annotated results

2. 🔍 **Explainable AI Tab**:

    - Generate Grad-CAM heatmaps
    - Perform occlusion sensitivity analysis
    - Interactive parameter adjustment
    - Visual explanation downloads

Configuration Options

- **Model Path**: Specify custom model checkpoint
- **Confidence Threshold**: Adjust detection sensitivity (0.01-1.0)
- **Visualization**: Toggle bounding boxes, masks, transparency
- **Debug Mode**: View detailed prediction information

🔍 Explainable AI Features

Grad-CAM Analysis

```
from src.explainable_ai import create_simple_explanation
from src.inference import CarDamagePredictor

# Load model
predictor = CarDamagePredictor('models/best_model.pth')

# Generate Grad-CAM explanation
result = create_simple_explanation(
    predictor,
    image,
    method='gradcam',
    alpha=0.4
)

# Save explanation
result['explanation_image'].save('gradcam_result.png')
```

Occlusion Analysis

```
# Generate occlusion sensitivity map
result = create_simple_explanation(
    predictor,
    image,
    method='occlusion',
    patch_size=50,
    stride=25,
    alpha=0.5
)
```

Testing Explainable AI

```
# Test explainable AI implementation
python test_explainable_ai.py

# Test with specific images
python test_gradcam_fix.py

# Simple functionality test
python working_gradcam_test.py
```

📊 Model Evaluation

Comprehensive Evaluation

```
# Run full evaluation (if evaluation.py exists)
python src/evaluation.py --model_path models/best_model.pth --data_dir dataset/test
```

Performance Metrics

The system evaluates models using:

- **Detection Metrics**: mAP@0.5, mAP@0.75, mAP@0.5:0.95
- **Segmentation Metrics**: Mask IoU, Boundary F1-Score
- **Classification Metrics**: Accuracy, Precision, Recall, F1-Score

Custom Evaluation

```
from src.inference import CarDamagePredictor
import os

# Load model
predictor = CarDamagePredictor('models/best_model.pth')

# Evaluate on test images
test_dir = 'dataset/test/images'
for image_file in os.listdir(test_dir):
    image_path = os.path.join(test_dir, image_file)
    predictions = predictor.predict(image_path, debug=True)
    print(f"{image_file}: {predictions['num_detections']} detections")
```

� Directory Structure

```
car-damage-detection/
├── src/                          # Source code
│   ├── __init__.py
│   ├── model.py                 # Model definitions
│   ├── dataset.py               # Dataset handling
│   ├── train.py                 # Training script
│   ├── inference.py             # Inference pipeline
│   └── explainable_ai.py        # XAI implementation
├── models/                       # Trained models
│   ├── best_model.pth
│   └── latest_model.pth
├── dataset/                      # Dataset directory
│   ├── train/
│   ├── val/
│   └── test/
├── tests/                        # Test scripts
│   ├── test_training.py
│   ├── test_explainable_ai.py
│   └── working_gradcam_test.py
├── app.py                        # Streamlit web app
├── requirements.txt              # Dependencies
└── README.md                    # This file
```

- PyTorch and torchvision teams for the deep learning framework
- Streamlit team for the web application framework
- VIA team for the annotation tool
- Open source computer vision community
- Contributors and testers

Happy Coding! ��

For more information, visit our project page or check out the demo video.