# Car Damage and Repair Cost Estimation Using Image Segmentation

Alexandre Baptista, nr fc64506 ; Carolina Rodrigues, nr fc62910

## 1. Problem Statement and motivation:

This project aims to develop a system that detects and classifies vehicle damage based on images as well as estimating the corresponding repair costs.

Given one or more images of the same damaged vehicle as input, the system will:

- Detect and identify the types of damage (such as scratches, dents, broken glass);
- Estimate the total repair cost.

This is particularly interesting for insurance and automotive industries, where this process is often done with manual inspections which are time-consuming and prone to subjectivity. With this project we intend to improve the accuracy and efficiency of damage detection and cost prediction.

## 2. Research and background

We will review previous academic papers on car damage detection (such as R.E. van Ruitenbeek, S. Bhulai, *Convolutional Neural Networks for vehicle damage detection* [1]. This study develops a model to locate vehicle damages and classify them into twelve categories using multiple deep learning algorithms.
And car industry case study on how AI and image recognition technology can improve insurance claims estimation processes. [2]

## 3. Dataset

For this project, we are using the **"Damaged Vehicle Images"** dataset available on Roboflow [3], which contains over 5,000 labeled images of damaged vehicles. The dataset provided was exported in **COCO JSON format**, including bounding boxes and class labels for various types of damage, such as **scratches, dents, and broken windshield**.

Additionally, we will explore complementary datasets, such as the Electric Vehicle Dataset (1997–2024) from Kaggle [4], this information will be used to support the **cost estimation** component of the project, as repair costs can vary significantly depending on these characteristics.

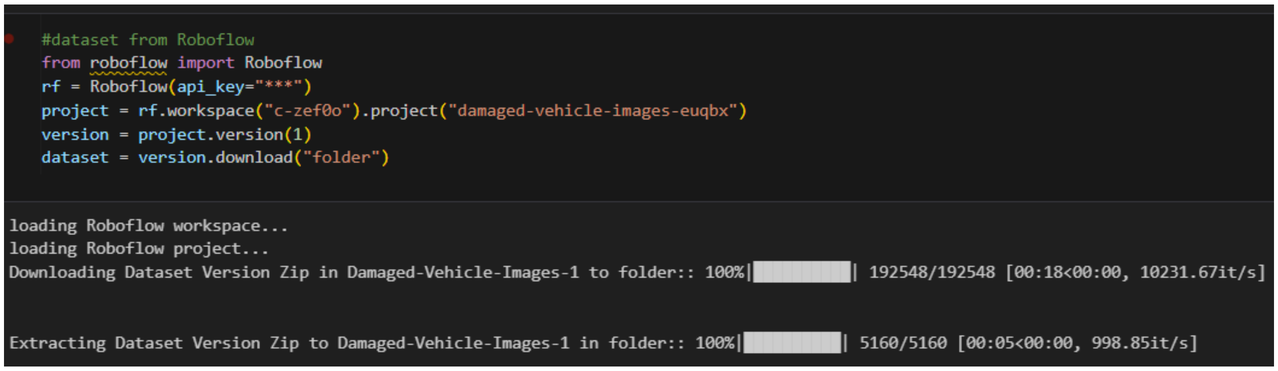
### 3.1 Data Preprocessing

The original dataset was distributed as follows: Train set: 94% (4,777 images), Validation set: 4% (193 images) and Test set: 2% (102 images).

To ensure a more balanced and standardized evaluation, we created a new project on Roboflow, uploaded the dataset, and used the platform's built-in tools to apply a new **80/10/10 split** for training, validation, and testing.

This choice follows best practices in dataset preparation aimed at balancing model generalization and avoiding overfitting, as discussed in *https://encord.com/blog/train-val-test-split/*.

Using the **Roboflow Python API** (as in Figure 1), we automatically exported and structured the dataset into our project directory, as follows:



```
#dataset from Roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="***")
project = rf.workspace("c-zef0o").project("damaged-vehicle-images-euqbx")
version = project.version(1)
dataset = version.download("folder")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Damaged-Vehicle-Images-1 to folder:: 100%|██████████| 192548/192548 [00:18<00:00, 10231.67it/s]


Extracting Dataset Version Zip to Damaged-Vehicle-Images-1 in folder:: 100%|██████████| 5160/5160 [00:05<00:00, 998.85it/s]
```

*Figure 1 - Usage of roboflow API for dataset download*

The resulting folder, his generates a folder **Damaged-Vehicle-Images-1/** containing train/, valid/, and test/, where each image is organized by damage category. An example of some images in the train folder in Figure 2.

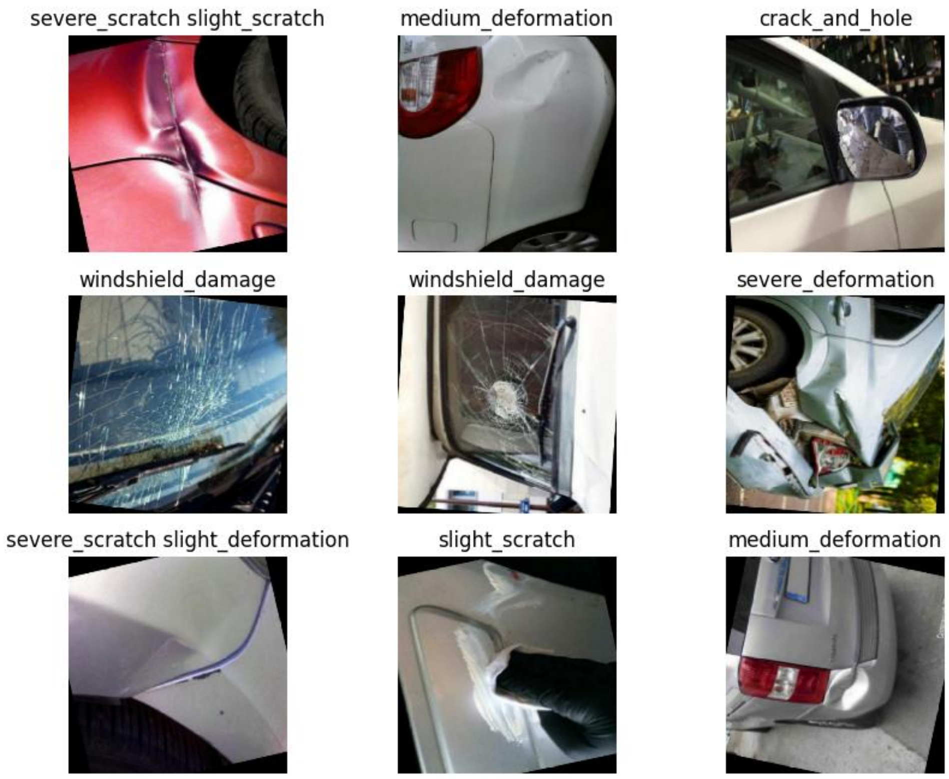
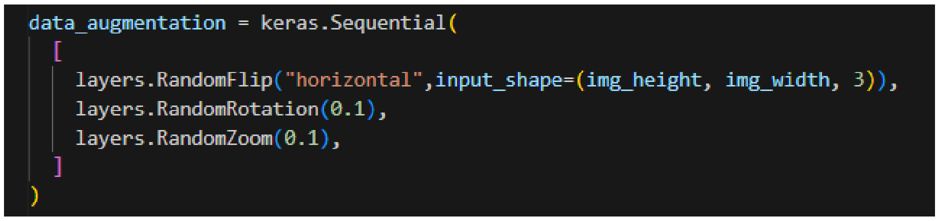


*Figure 2 - sample of images from the training set*

To improve training performance, we applied TensorFlow data pipeline optimizations:

- Dataset.cache() used to keep images in memory after the first epoch, avoiding I/O bottlenecks;
- Dataset.prefetch() overlaps preprocessing and model execution for improved efficiency.

We also implemented **data augmentation** (Figure 3) techniques to increase dataset variability and prevent overfitting. The augmentation pipeline including random horizontal flipping, random rotations and random zooms.



```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(img_height, img_width, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

*Figure 3 - Data augmentation using keras model: random flipping, random rotations and random zooms*

Below in Figure 4, is a sample of images from the augmented set:

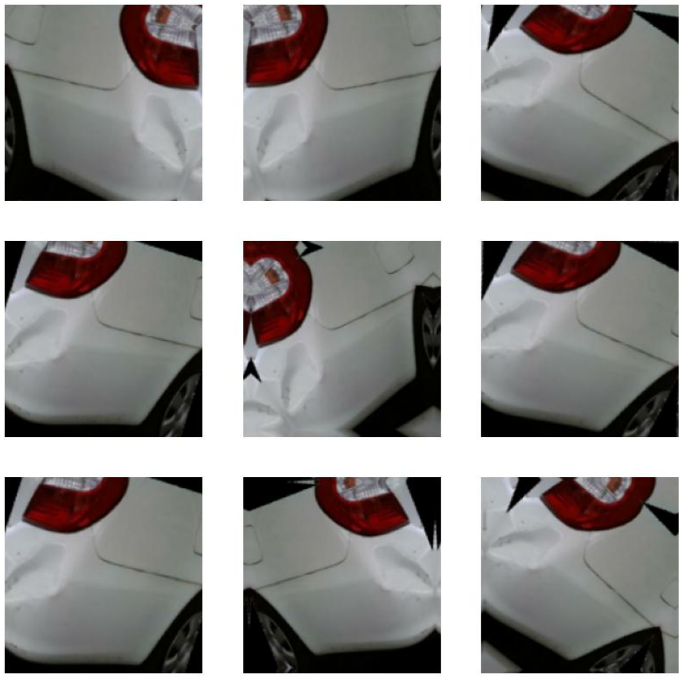


*Figure 4 - Sample of images after data augmentation*

## 4. Baseline Description and Benchmarks

To segment and classify car damage, as a starting point, we implemented a simple **baseline model** using the **Keras framework** (with TensorFlow backend). The goal of this model is to establish a performance reference for future improvements and comparisons.

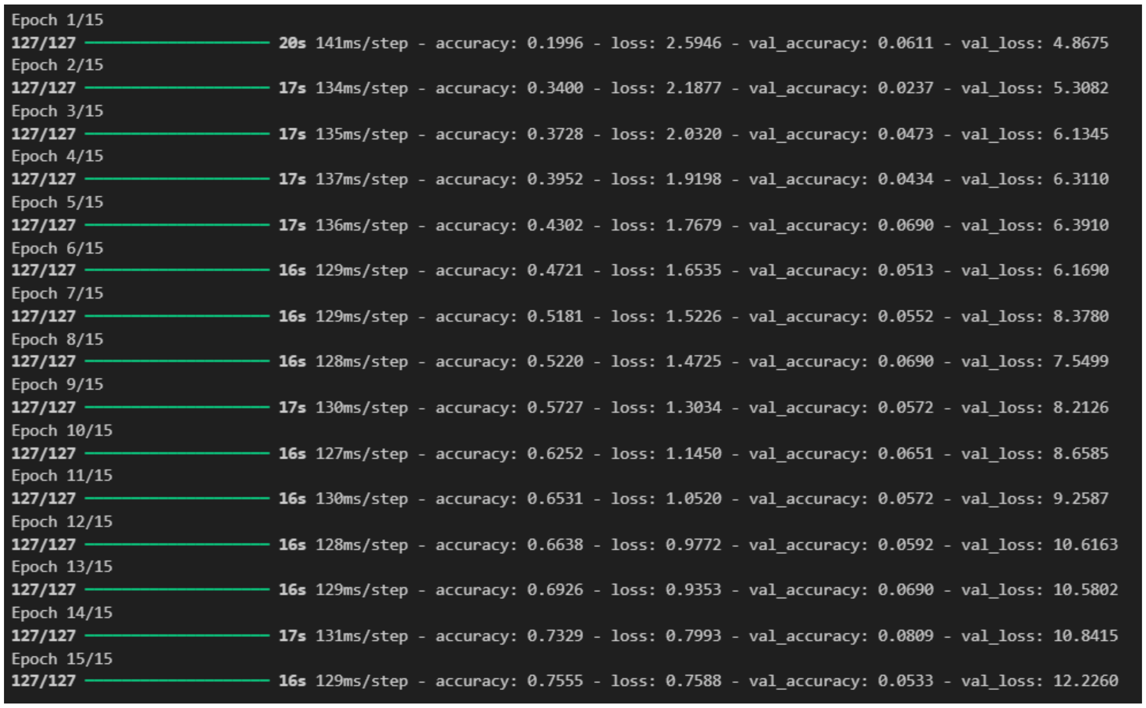
### Model Architecture and Training Configuration

- **Framework:** Keras (TensorFlow)
- **Architecture:** Basic Convolutional Neural Network (CNN)
- **Optimizer:** Adam
- **Epochs:** 15
- **Batch size:** 32

The preliminary results from this model were:

- **Training accuracy:** 0.7555

- **Training loss:** 0.7588
- **Validation accuracy:** 0.0533
- **Validation loss:** 12.2260



```
Epoch 1/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 20s 141ms/step - accuracy: 0.1996 - loss: 2.5946 - val_accuracy: 0.0611 - val_loss: 4.8675
Epoch 2/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 134ms/step - accuracy: 0.3400 - loss: 2.1877 - val_accuracy: 0.0237 - val_loss: 5.3082
Epoch 3/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 135ms/step - accuracy: 0.3728 - loss: 2.0320 - val_accuracy: 0.0473 - val_loss: 6.1345
Epoch 4/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 137ms/step - accuracy: 0.3952 - loss: 1.9198 - val_accuracy: 0.0434 - val_loss: 6.3110
Epoch 5/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 136ms/step - accuracy: 0.4302 - loss: 1.7679 - val_accuracy: 0.0690 - val_loss: 6.3910
Epoch 6/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step - accuracy: 0.4721 - loss: 1.6535 - val_accuracy: 0.0513 - val_loss: 6.1690
Epoch 7/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step - accuracy: 0.5181 - loss: 1.5226 - val_accuracy: 0.0552 - val_loss: 8.3780
Epoch 8/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 128ms/step - accuracy: 0.5220 - loss: 1.4725 - val_accuracy: 0.0690 - val_loss: 7.5499
Epoch 9/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 130ms/step - accuracy: 0.5727 - loss: 1.3034 - val_accuracy: 0.0572 - val_loss: 8.2126
Epoch 10/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 127ms/step - accuracy: 0.6252 - loss: 1.1450 - val_accuracy: 0.0651 - val_loss: 8.6585
Epoch 11/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 130ms/step - accuracy: 0.6531 - loss: 1.0520 - val_accuracy: 0.0572 - val_loss: 9.2587
Epoch 12/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 128ms/step - accuracy: 0.6638 - loss: 0.9772 - val_accuracy: 0.0592 - val_loss: 10.6163
Epoch 13/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step - accuracy: 0.6926 - loss: 0.9353 - val_accuracy: 0.0690 - val_loss: 10.5802
Epoch 14/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 17s 131ms/step - accuracy: 0.7329 - loss: 0.7993 - val_accuracy: 0.0809 - val_loss: 10.8415
Epoch 15/15
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step - accuracy: 0.7555 - loss: 0.7588 - val_accuracy: 0.0533 - val_loss: 12.2260
```

*Figure 5 - Accuracy, Loss, Validation accuracy and validation loss results, of the 15 epochs.*

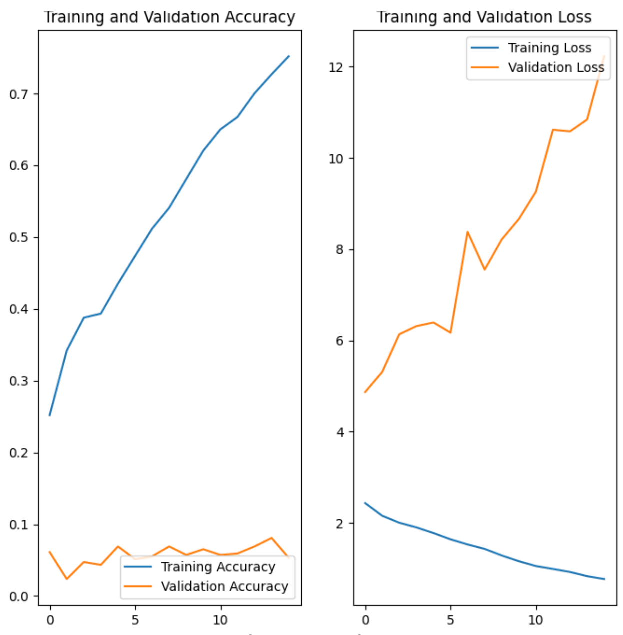


*Figure 6 - Sample of images after data augmentation*

**Benchmarking Purpose**

baseline serves as a **benchmark** for evaluating more sophisticated models. In the next steps, we plan to:

- Reimplement the pipeline in **PyTorch**
- Use **ResNet-50** as a feature extractor via transfer learning
- Compare performance across models in terms of **accuracy**, **training time**, and **generalization**.

**Next Steps**

To improve performance, we plan to re-implement the model in PyTorch, using ResNet-50, an architecture known for its robustness and efficiency in image classification tasks [5].
We will compare the two implementations (Keras vs. PyTorch) in terms of its accuracy, training speed and robustness

In the later phases, we also plan to incorporate object detection models such as **Faster R-CNN** for better damage localization, and explore regression models for cost estimation based on extracted damage features.

## 5. Evaluation and Expected Results

The performance of the system will be assessed using both **qualitative** and **quantitative** evaluation methods, focusing on two core tasks: **damage detection/classification** and **repair cost estimation**.

**Qualitative Evaluation**

We will visually inspect the model outputs to verify if the system correctly identifies damaged areas and focuses attention on relevant regions.

These visualizations will help validate whether the model is learning meaningful features and whether the predictions are interpretable and trustworthy.

**Quantitative Evaluation**

For the **damage classification** component, we will compute standard classification metrics:

- **Accuracy; Precision; Recall; F1-score;  Confusion matrix**

For the **repair cost estimation** component, we will consider two scenarios:

- Categorical estimation (such as low/medium/high cost range). Evaluation using classification **accuracy** and **F1-score.**
- **Continuous estimation** exact monetary value. Evaluation using **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**

These metrics will allow us to measure both the classification effectiveness and the accuracy of the cost prediction component.

We expect to produce performance plots such as learning curves and confusion matrices, along with segmented image examples and cost prediction distributions, to fully document and communicate the system's results.

## 6. New dataset

Following the issues observed in the baseline model, where, as shown in Figure 5 and Figure 6, the 15-epoch plain CNN reached 75% training accuracy but only 5% validation accuracy (a clear case of overfitting or potential data leakage), we decided to first revise our dataset before altering the modeling strategy.

We identified that the original dataset might have contributed to the poor generalization performance. Therefore, we replaced it with a new and more structured dataset: ***"Car Damage Severity Detection – CarDD"*** [6]. With this substitution the goal was to rerun the model under better-controlled conditions and observe whether these issues could be mitigated.

- **Dataset structure**:

The new dataset uses the **YOLOv8** format, which includes:

- **Images** and corresponding **TXT annotation files** for object detection;
- A **YAML configuration file** that defines label names and IDs;

There is a total of **11 damage categories**, including:

- 0 - car-part-crack, 1 - detachment, 2 - flat-tire, 3 - glass-crack, 4 - lamp-crack, 5 - minor-deformation, 6 - moderate-deformation, 7 - paint-chips, 8 - scratches, 9 - severe-deformation, 10 - side-mirror-crack.

Unlike the previous dataset, this one not only provides **class labels** but also specifies the **location** and allows a count of each damage type per image. This is particularly useful for both object detection and tasks such as cost estimation.

The dataset is split in a 70/20/10 ratio, with: **Train:** 1395 images and labels; **Validation:** 401 images and labels; **Test:** 204 images and labels. An example of some images with the corresponding labels is shown in Figure 7 below.

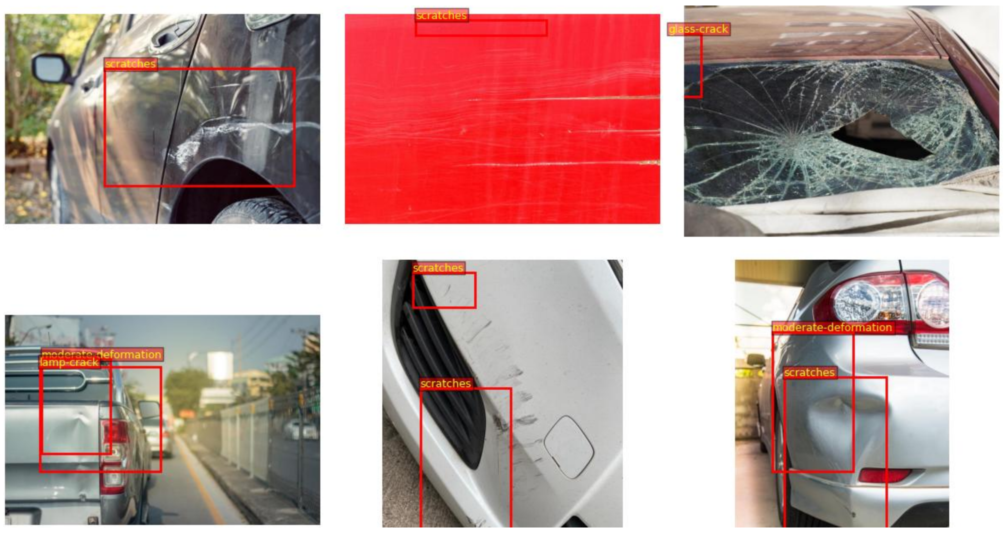


*Figure 7 - Sample of images and labels of the new dataset*

## 7. Preprocessing and Sanity Checks

To ensure dataset integrity and avoid potential sources of error, such as data leakage, image corruption, or class imbalance, we conducted some **sanity checks** before training our models.

### a) Data Leakage Detection

As a first step, we verified whether there were any duplicate or near-duplicate images shared between the train and valid splits.

To do this, the process scanned all *.jpg* files in both folders, then for each image, it computed a SHA-1 hash to detect exact byte-level duplicates and a perceptual hash (pHash) to detect similar images, allowing for minor differences of Hamming distance $\leq 5$ bits.

Then compared the hashes across both splits and it would flag and list any image pairs that were exact copies (with identical SHA-1 hashes) or near-duplicates (pHash Hamming distance $\leq 5$).

**Result**:
Since no exact duplicates or near-duplicates were found between the train and valid sets, we can confirm that there is **no data leakage**.

### b) Image corruption

To ensure file integrity, we scanned all images in each split (train, valid, and test), using *PIL.Image.open()* function, and calling *img.verify()* on each file.

This allows us to validate and check whether each image file can be opened, without fully loading it into memory. If any exception is raised, due to file corruption or unreadability, the corresponding file is flagged and listed.

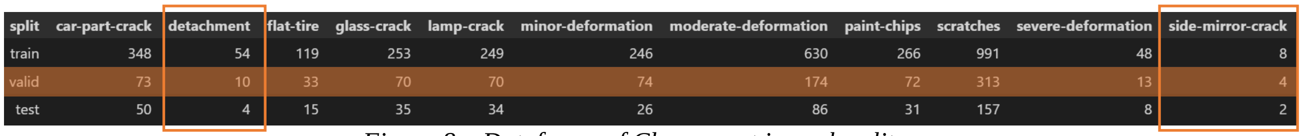
**Result:**
No corrupt or unreadable images were detected. Indicating that all images across the sets are intact and readable.

### c) Class and Label distribution

We analyzed the label distribution by counting the total number of bounding boxes per class in each data split (train, validation, and test). This allows us to detect potential class imbalance and verify if each class has at least 10 samples, particularly in the validation set.

To do this, a dataframe was generated, with one row per split (train, val, test), one column per class and each cell containing the number of bounding boxes for that class in the corresponding split.

**Result:**
From the resulting dataframe, shown in Figure 8, we observe that most classes have sufficient representation across splits. However, some classes such as *"detachment"* and *"side-mirror-crack"* have counts just above the minimum threshold of 10 samples in the validation set.



| split | car-part-crack | detachment | flat-tire | glass-crack | lamp-crack | minor-deformation | moderate-deformation | paint-chips | scratches | severe-deformation | side-mirror-crack |
|---|---|---|---|---|---|---|---|---|---|---|---|
| train | 348 | 54 | 119 | 253 | 249 | 246 | 630 | 266 | 991 | 48 | 8 |
| valid | 73 | 10 | 33 | 70 | 70 | 74 | 174 | 72 | 313 | 13 | 4 |
| test | 50 | 4 | 15 | 35 | 34 | 26 | 86 | 31 | 157 | 8 | 2 |

*Figure 8 – Dataframe of Class count in each split*

This suggests a **class imbalance**, which we plan to address during training using one or more of the following strategies:

1. **Redo the dataset split** using stratified sampling by class and shuffling (so 80/10/10 split instead of relying on the default 70/20/10 split);
2. Apply **targeted data augmentation** to underrepresented classes using transformations;
3. **Merge rare classes** into similar ones to increase class support and simplify the prediction.

Once these strategies are applied, we will re-run the training to confirm if the validation accuracy, which would confirm effective mitigation of the imbalance. These steps are essential to ensure fair evaluation and to reduce bias toward the most frequent damage categories.

## 8. Experiments and Results

### Re-evaluation of Baseline Model on New Dataset

Before implementing the strategies mentioned before, we re-ran the same baseline model used in Milestone 1 on the new CarDD dataset to evaluate whether the overfitting issue would persist. And compared model performance across 15, 50 and 100 epochs.

*Table 1 - Training Results (Keras CNN, no augmentation)*

| *Epochs* | *Training Accuracy* | *Training Loss* | *Validation Accuracy* | *Validation Loss* |
|---|---|---|---|---|
| *15* | 0.4359 | 1.4874 | 0.3317 | **1.9439** |
| *50* | **0.9961** | 0.0180 | **0.3940** | **5.4632** |
| *100* | 1.0000 | **0.0028** | 0.3541 | **8.0316** |

**Conclusions:**

Even when using the improved CarDD dataset, the baseline model exhibited clear signs of overfitting when trained without data augmentation and without addressing class imbalance.

As shown in Table 1, we can see that:

- After 50 epochs: training accuracy approached 99%, while validation accuracy plateaued around 39%, and validation loss increased dramatically from ~2 to 5.5.
- After 100 epochs: the training loss dropped near zero, indicating the model was memorizing the training data, and validation loss exceeded 8.0, confirming overfitting.

In the appendix, the Figures A1-A3, present the learning curves for both the accuracy (left) and loss (right) of the training and validation:

- **Accuracy curves:** show that training accuracy climbs toward 1.0 starting around epoch 30, while validation accuracy flattens between 0.35–0.40, likely limited by class imbalance or lack of regularization techniques.
- **Loss curves:** the training loss decreases steadily until around epoch 30, where it reaches near zero. This coincides with the training accuracy curve climbing toward 1.0, confirming overfitting. Validation loss remains close to training loss during the first ~10 epochs,

however, after approximately epoch 15, validation loss begins to rise steadily, forming a clear upward slope, a classic sign of high variance and overfitting.

## 9. Next Steps and Planned Improvements

The results obtained strongly support the need to introduce additional regularization strategies to improve model generalization and address class imbalance. These include:

- **Albumentations-based data augmentation**, to increase variability in underrepresented classes
- **Focal loss**, to give more weight to rare classes and stabilize training
- **Transfer learning**, using frozen backbones such as **ResNet-50** or **YOLOv8**, to leverage pretrained feature extractors

These will be the focus of our next training iterations. But, before implementing and training more advanced models, we will first apply these strategies incrementally to the baseline setup starting with class balancing techniques and re-run training to evaluate their impact.

Subsequently, we will incorporate the full regularization pipeline and train the more complex architectures, allowing us to compare performance across all configurations and select the most effective approach based on empirical results.

## References:

[1] R. E. van Ruitenbeek and S. Bhulai, *"Convolutional Neural Networks for Vehicle Damage Detection,"* Machine Learning with Applications, vol. 7, 2022. [Online]. Available: https://doi.org/10.1016/j.mlwa.2022.100332

[2] Celebal Technologies, *"Leading Insurance Provider Leverages AI-Powered Vehicle Damage Detection,"* Case Study. [Online]. Available: https://celebaltech.com/case-studies/leading-insurance-provider-leverages-ai-powered-vehicle-damage-detection

[3] Roboflow dataset, *"Damaged Vehicle Images Dataset,"* [Online]. Available: https://universe.roboflow.com/project/damaged-vehicle-images/dataset/3

[4] Kaggle, *"Electric Vehicle Data (1997–2024 Update Version),"* [Online]. Available: https://www.kaggle.com/datasets/iottech/electric-vehicle-data-1997-2024-update-version

[5] Roboflow, *"What is ResNet-50?"*, Roboflow Blog, [Online]. Available: https://blog.roboflow.com/what-is-resnet-50

[6] Roboflow dataset, *"Car Damage Severity Detection – CarDD"* [Online]. Available: https://universe.roboflow.com/car-damage-detection-cardd/car-damage-severity-detection-cardd

## Appendix:

Learning curves:

- 15 epochs:

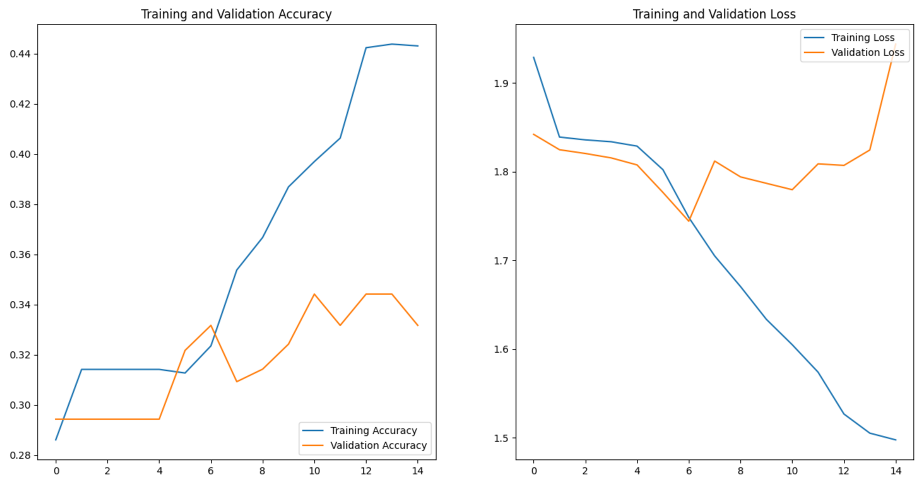


*Figure A1 - Learning Curves: Accuracy and Loss (Baseline Model, CarDD Dataset) over 15 epochs.*

- 50 epochs:

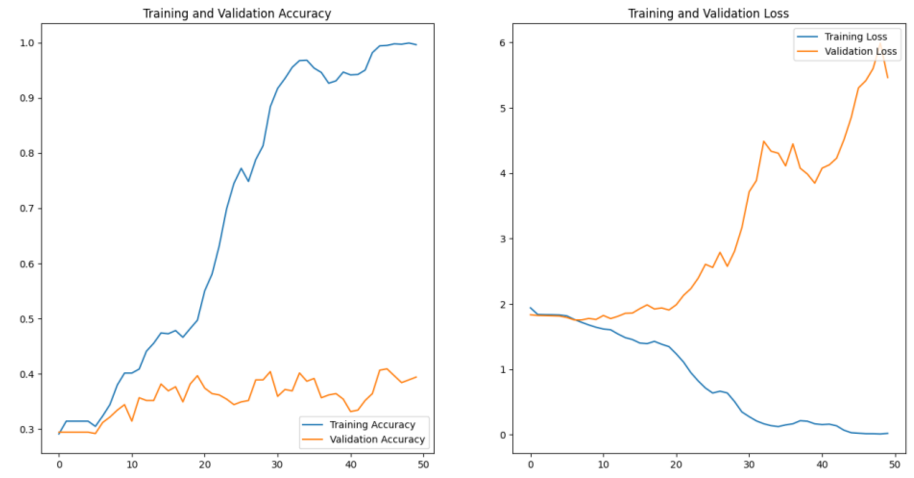


*Figure A2 - Learning Curves: Accuracy and Loss (Baseline Model, CarDD Dataset) over 50 epochs.*

- 100 epochs:

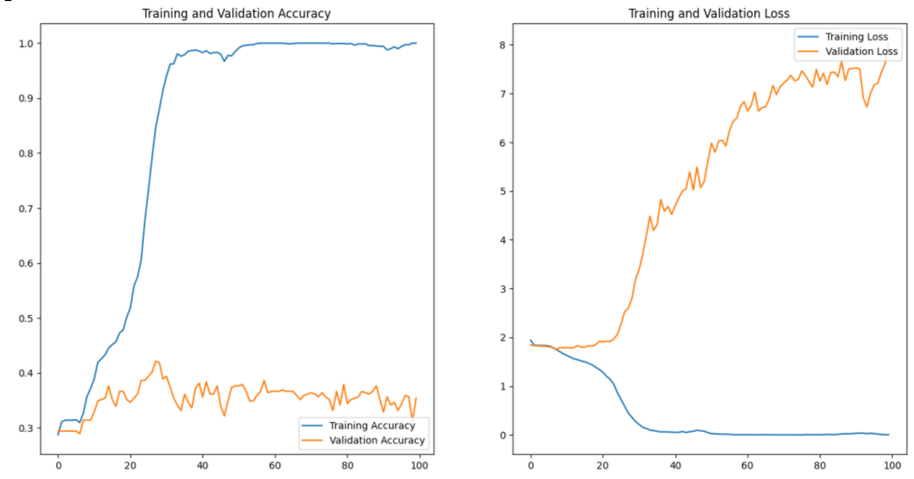


*Figure A3 - Learning Curves: Accuracy and Loss (Baseline Model, CarDD Dataset) over 100 epochs.*