Car Damage Classification

Final project - Deep Learning (046211)

Yarden Shavit 206974883

Dor Yogev 209082130

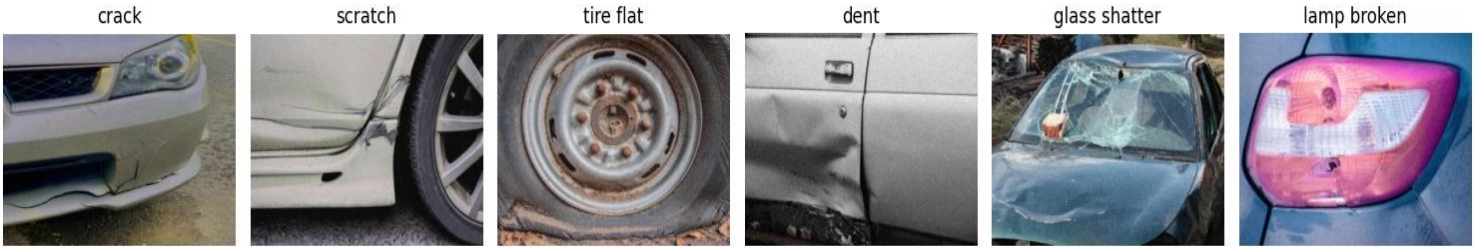
Ravid Goldenberg 206698912

August 20th , 2024

Introduction

The objective of our project is to take a pre-trained Convolutional Neural Network (CNN) and perform fine-tuning on certain parts of the network to use it for classifying types of car damages. We have a relatively small dataset consisting of 7,000 labeled images. To achieve the best possible learning process, we used Optuna to optimally tune our hyperparameters. Additionally, we applied data augmentations to address the limited number of images available for training.

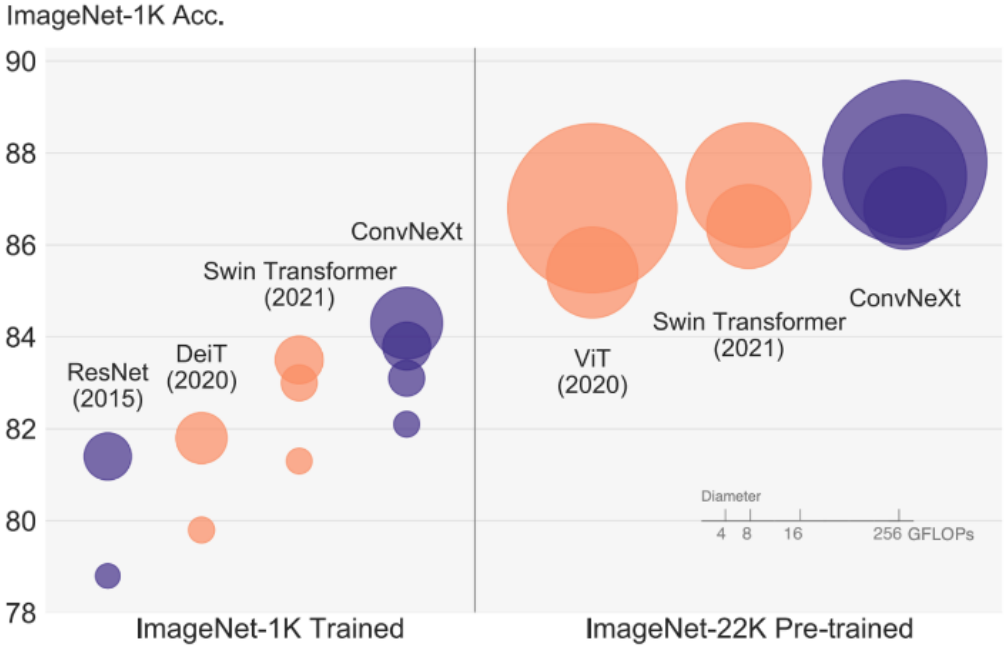
Our database contains 7,000 labeled images of various car damages. There are six possible labels: crack, scratch, tire flat, dent, glass shatter, and lamp broken. This classification can be beneficial for insurance companies and repair shops, providing them with quick and preliminary information about the types of car damages.



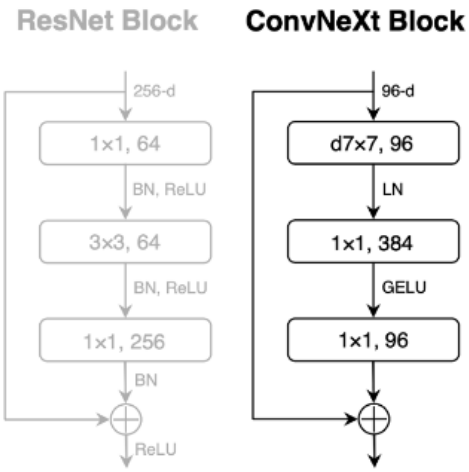
Our work did not rely on any prior work but was based on code from the course exercises, a database from Kaggle, and a GitHub repository that contained the implementation of the pretrained CNN. We chose the ConvNeXt model for our implementation.

Method

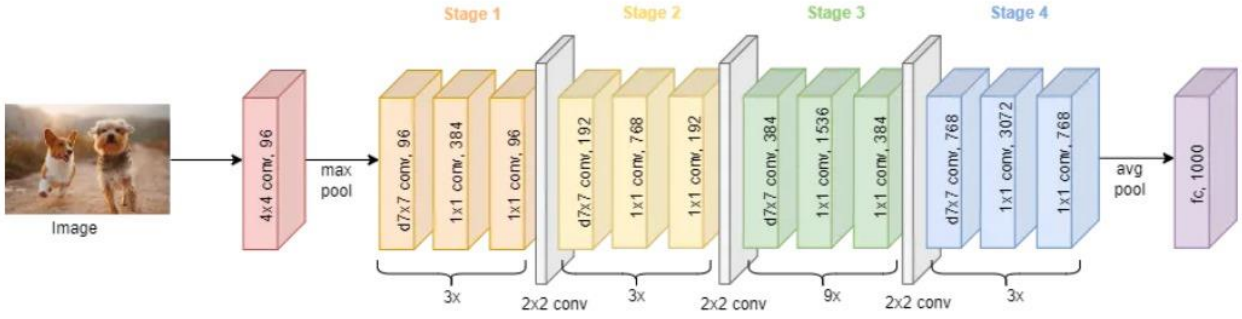
For the project, we used the ConvNeXt network, a CNN developed by Facebook in 2020. It was trained on ImageNet 1K as well as ImageNet 22K (different pre-trained models). The network demonstrated very strong performance, surpassing older CNNs like ResNet and networks that combine transformers, such as Swin Transformers.



The ConvNeXt architecture is built hierarchically, with a basic block that includes a 7x7 CNN layer with 96 channels, followed by a Layer Norm, then an FC layer with 384 channels and GELU activation, and finally another FC layer with 96 channels. The block also includes skip connection from beginning to end. The main innovation of ConvNeXt is the use of FC layers in the block (not just CNN layers), as well as using Layer Norm instead of Batch Norm and GELU instead of ReLU. The diagram shows the differences between the ConvNeXt block and the ResNet block graphically.



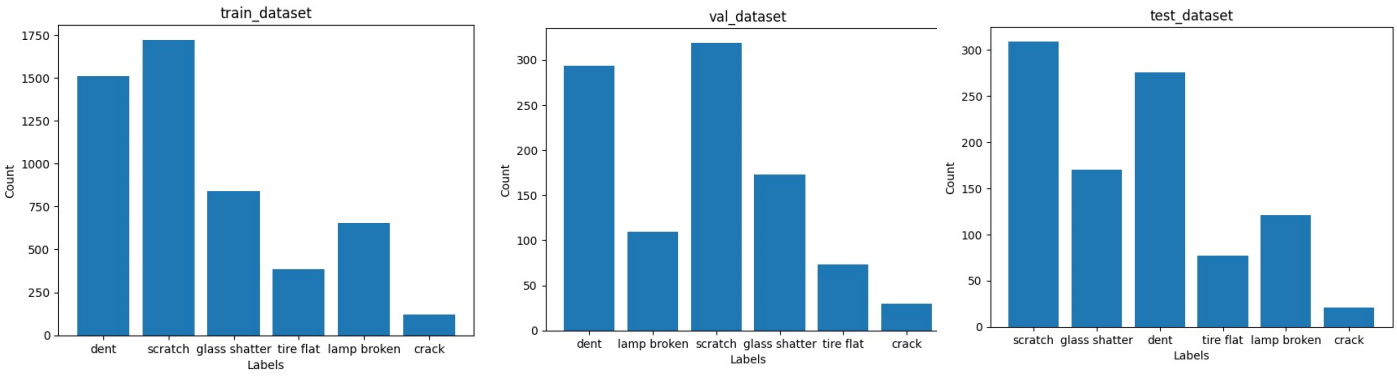
After understanding what a ConvNeXt block is, let's describe the architecture. The ConvNeXt network consists of 4 stages, each containing a different number of blocks depending on the size of the chosen model. Between each stage and at the network's input, there are downsampling layers. Initially, we tried working with ConvNeXt_Small, but the number of parameters and computational operations repeatedly caused our GPU to crash due to insufficient memory. Therefore, we switched to a smaller model, ConvNeXt_Tiny, which has 28 million parameters and performs 4.5 GigaFLOPs. This model contains 3 blocks in its first two stages, 9 blocks in the third stage, and 3 blocks in the last stage. After a failed attempt to perform fine-tuning for all the model's layers, we realized the number of parameters was too large. Gradually, we reduced the number of stages we trained until we only trained the last stage. This was because the third stage is the largest in the network and required a lot of memory, which the GPU could not handle.



ConvNext — General Structure

Let's describe the training stage. First, we took our dataset, consisting of 7,000 labeled images, and split it into train, validation, and test sets. It's important to note that the original Kaggle dataset contains 7,000 labeled images for training and an additional 1,000 unlabeled

images for testing. Therefore, we had to take all the labeled images and split them ourselves. We divided them into 5,000 images for training, 1,000 for validation, and 1,000 for testing. Below is the distribution of the images, which shows that the original dataset is not evenly distributed.



we took the pre-trained ConvNeXt_Tiny network and replaced its final FC layer, which was originally designed to classify 1,000 labels, with an FC layer that outputs 6 labels, to match our dataset which includes 6 categories. Then, we took Stage 4 and applied Dora to the FC networks within its three blocks to reduce the number of parameters and make the model trainable. The rest of the model was frozen to prevent gradient backpropagation.

After modifying the model's layers, we wanted to obtain optimal hyperparameters using Optuna. We decided to allow Optuna 200 trials, with 10 epochs per trial, to optimize the following hyperparameters:

- Lr_slow: learning rate for the stage layers.
- Lr_fast: learning rate for the final FC layer.
- Optimizer: type of optimizer (Adam, SGD, RMSprop).
- Weight_decay_slow: weight decay for the stage layers.
- Weight_decay_fast: weight decay for the final FC layer.
- Batch_size: 16, 32, 64.

For the final FC layer, we used one optimizer, and for the rest of the layers, we used a different optimizer.

Later in the results section we will present the results of the hyper-parameters tuning with optuna.

After obtaining the hyperparameters, we wanted to increase the number of images the model trains on. Therefore, we decided to perform data augmentation on the training dataset only, within the training loop. We used useful augmentations that helped us in the course assignments and were suitable for the dataset:

```
aug_list = AugmentationSequential(
    K.ColorJitter(0.1, 0.1, 0.1, 0.1, p=0.9),
    K.RandomHorizontalFlip(p=0.5),
    K.RandomAffine(degrees=15.0, translate=(0.1, 0.1), scale=(0.8, 1.2), shear=(10.0, 10.0), p=0.5),
    K.RandomGaussianNoise(std=0.05, p=0.9),
    K.RandomCrop(size=(224, 224), padding=None, p=0.2),
    same_on_batch=False
)
```

Colorjitter Apply a random transformation to the brightness, contrast, saturation and hue of a tensor image, RandomHorizontalFlip flip horizontally the image , RandomAffine Apply a random 2D affine transformation to a tensor image, RandomGaussianNoise adds Gaussian Noise and RandomCrop Crop random patches of a tensor image on a given size.

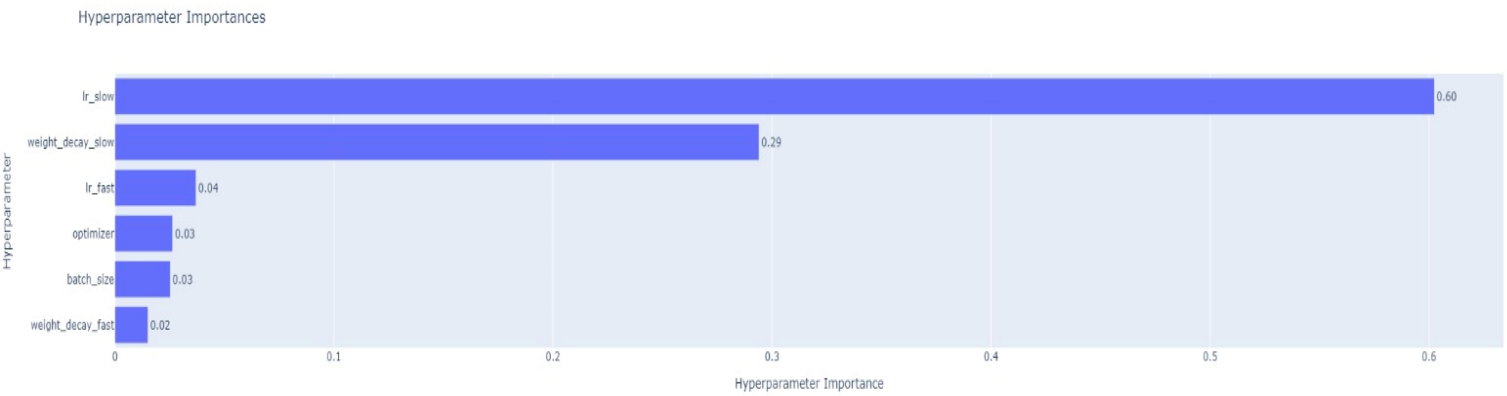
After setting the augmentations, we began training the loop on the train set, with a total of 200 epochs. After each epoch, we calculated the accuracy on both the train set and the validation set. When the accuracy on the validation set improved, we saved the model parameters. For optimization, we used the cross-entropy loss function, but only for the optimizer; to evaluate the model's performance, we looked at the accuracy on the validation set. Additionally, after one training attempt, we noticed that the accuracy on the validation set jumped erratically after 10 epochs. Therefore, we decided to add a scheduler of type MultiStepLR to reduce the learning rate by a factor of 10 for both optimizers, once after 75 epochs and again after 150 epochs.

Results

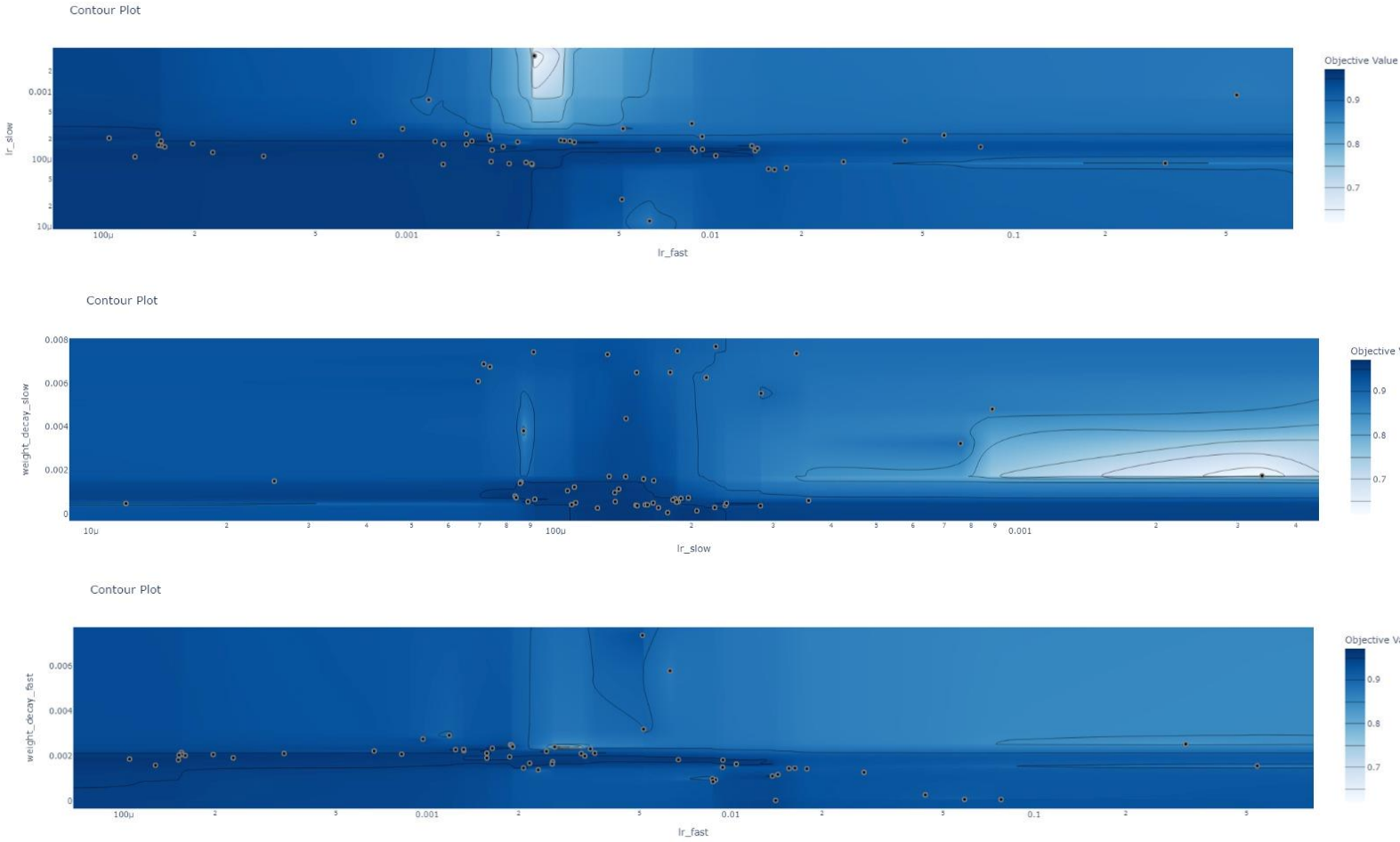
First, let's start with the results of the hyperparameter tuning using Optuna. The results we obtained are as follows:

```
Study statistics:
 Number of finished trials:  200
 Number of pruned trials:  143
 Number of complete trials:  57
Best trial:
 Value:  0.970912738214644
 Params:
 lr_slow: 0.00013955902469821164
 weight_decay_slow: 0.0011343113514715552
 optimizer: Adam
 lr_fast: 0.009427355395105285
 weight_decay_fast: 0.001827833478313847
 batch_size: 32
```

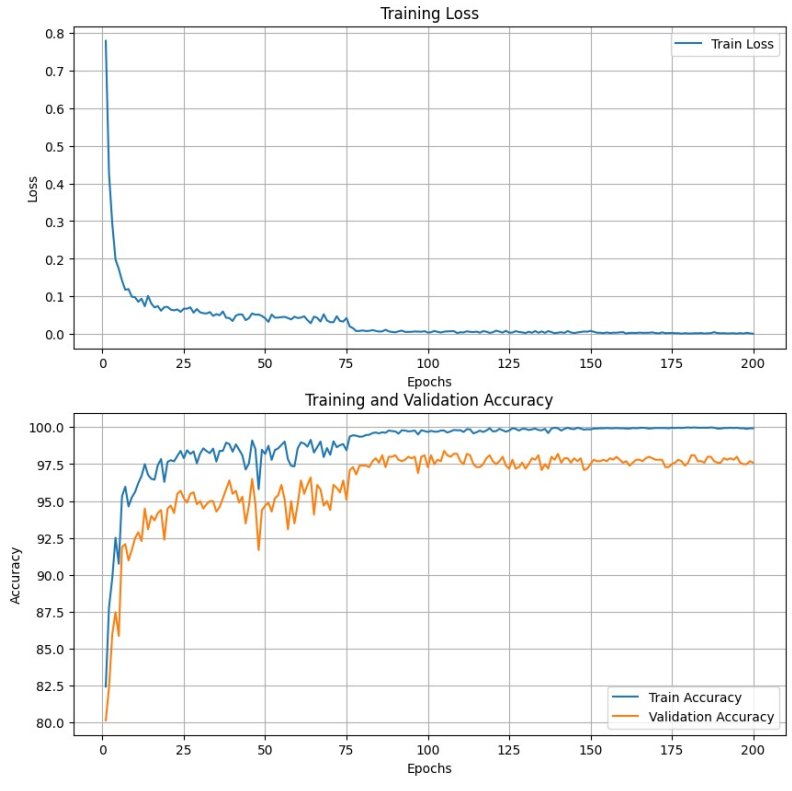
It can be seen that Optuna was highly efficient, cutting down 143 out of 200 trials and saving a lot of runtime. Additionally, in the following figure, you can see its analysis regarding the importance of the hyperparameters. The significant weight given to lr_slow validates our approach of splitting the optimizer into two: one for the final FC layer and one for the last stage.



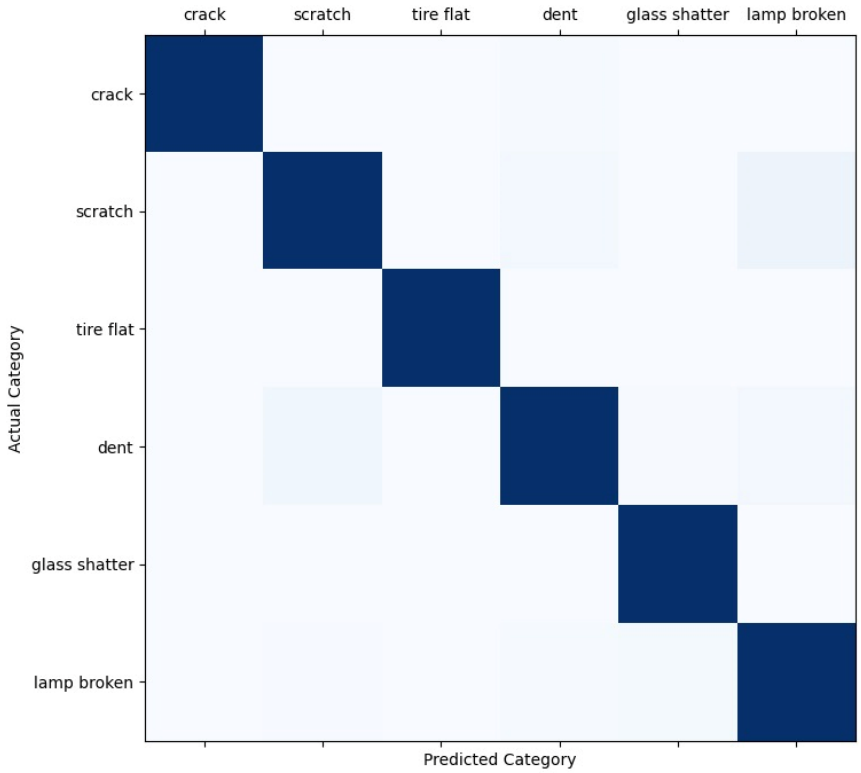
We will present several 2D spaces of hyperparameters that Optuna shows, including the search points and its analysis of the space's shape.



Now we will present the training results. First, we will show the progress of the training loss during training, which indicates good optimization performance. Additionally, we will display the accuracy graphs for both the training and validation sets.



The best model on the validation set was at epoch number 105 with an accuracy of 98.395%. We took the model parameters and tested them on the test set to obtain the final performance of our model, achieving an accuracy of 97.328%. Below is the confusion matrix showing the model's classification results; the color of the incorrect classifications is barely visible due to the very few misclassifications.



It can be seen that when the model made classification errors, it confused dent with scratch and lamp broken with scratch. This is quite reasonable given that almost every car damage also results in scratches. However, our dataset only provides a single label per image.

Conclusions from the project: On Kaggle, we found one similar project that did fine-tuning on an EfficientNet model. We did not use it during our project because its code was cumbersome and unclear. According to the reported results, they achieved an accuracy of 98% on their test set. However, it is unclear how they created this set since the test set of the dataset is unlabeled. We can say that our model achieved similar and high performance, perhaps slightly lower, which might indicate that the EfficientNet architecture is better suited for classifying this specific dataset compared to ConvNeXt.

Possible future work includes fine-tuning more stages in ConvNeXt within feasible limits to improve model accuracy. Additionally, we could change the architecture and perform fine-tuning on it. Another approach could be to combine architectures and connect their last layers with an FC layer, potentially gaining the best from each different model.

Ethics Statement

Part1:

Student names: Yarden Shavit , Dor Yogev ,

Project Title: Fine Tune ConvNeXt CNN for Car Damage Classification

Project Description:

This project involves using ConvNeXt CNN (tiny ConvNext) to classify car damages such as flat tires and broken lights. We aim to optimize hyperparameters using Optuna and fine-tune the ConvNeXt model with data augmentations to improve classification accuracy.

Part2:

a. List 3 types of stakeholders that will be affected by the project.

Car owners.

Insurance companies.

Car repair shops.

b. What will an explanation that is given to each stakeholder look like?

Car owners:

"We have developed a machine learning model to help identify specific types of car damage, such as flat tires and broken lights. This tool aims to provide quick and accurate assessments to facilitate faster repairs and claims processing, ultimately enhancing your overall experience as a car owner."

Insurance companies:

"Our project leverages advanced neural networks to classify car damages with high accuracy. This technology can streamline your claims process by providing reliable damage assessments, reducing the time and resources spent on manual inspections, and improving the efficiency of claim settlements."

Car repair shops:

"We have created an AI-based tool to identify various car damages, which can aid your diagnostic processes. By integrating this technology, you can offer quicker and more accurate assessments to your customers, improving service quality and operational efficiency."

c. Who is responsible for giving the explanation to each stakeholder?

Car owners:

The customer service departments of car insurance companies and repair shops are responsible for providing clear explanations about how the AI tool works and how it benefits the car owners.

Insurance companies:

The project developers or the technology integration team should present detailed explanations to insurance companies, highlighting the technical aspects and potential benefits of the AI model for their claims process.

Car repair shops:

The project team or the AI solution provider should explain the tool's functionalities to car repair shop managers and technicians, ensuring they understand how to integrate and use the technology effectively in their workflow.

Part3:

a. What do you think needs to be added/changed in the Generative AI responses, to make the explanation more ethical?

We think that the chat GPT only gives positive views on the potential benefits of each stakeholder from the new AI tool, it doesn't give any potential bad influence like losing jobs in car insurance companies (classify and value the damage), it is too positive. The Chat wrote

about the logical responsibility of whom that needs to explain the new tool but it didn't mention the moral responsibilities to explain to the stockholder what is the possible consequences for their lives. It's very clear that chat GPT is very rational and logical but lacks basic understanding in human morals and ethics.

References:

[1] We took our dataset from:
https://www.kaggle.com/datasets/sudhanshu2198/ripik-hackfest

[2] We took our pretrained model from:
https://github.com/facebookresearch/ConvNeXt/blob/main/README.md

[3] We looked for accuracy comparison for our model with this model based of efficientnet:
https://www.kaggle.com/code/sudhanshu2198/instant-vehicle-damage-insurance-verification/notebook#Evaluation

[4] We also use many of the code and data that in the course material.