Car Damaged Detection and Evaluation

Capstone Project Report

Pham Viet Duong
Do Huu Dai

Supervisor: MSc. Do Thai Giang

ĐẠI HỌC FPT

April 2023

Acknowledgments

About this thesis, we would like to express our gratitude to our supervisor, MSc. Do Thai Giang, for all of the assistance and guidance that he provided. Over the entirety of our academic investigation, his vast expertise and extensive experience have served as a source of inspiration for us. In addition, we would like to express our gratitude to all of the lecturers and professors at the Department of Artificial Intelligence who have trained us and provided us with fundamental knowledge that enables us to carry out this project. In conclusion, we would like to express our gratitude to FPT University for awarding each of us a studentship that enabled us to complete our research and write our thesis.

Abstract

The proposed car damage detection system follows a multi-stage cascade approach, where each stage consists of a classifier and a bounding box regressor. The system leverages the Cascade Mask R-CNN architecture with a Swin-FPN backbone, which provides multi-scale contextual information for accurate and robust object detection. The system is trained on a large dataset of labeled car images, including various types of car damages such as dents, scratches, cracks..., to learn the discriminative features for car damage detection.

To evaluate the performance of the proposed system, extensive experiments are conducted on a benchmark dataset of car images with ground-truth car damage annotations. The results show that the Cascade Mask R-CNN-based car damage detection system achieves good performance in terms of detection accuracy and computational efficiency. The proposed system is able to accurately localize and segment car damages in images.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Car insurance in Vietnam has become a significant source of revenue for insurance companies in recent years due to the rapid growth of the country's overall vehicle population. There is a risk of claims leakage occurring as a result of incorrect evaluations made by insurers during the processing of customers' claims. These incorrect evaluations may be the result of subjective views or insufficient training. Because of these mistakes, the companies would end up losing a significant amount of money [1]. In the absence of a deficiency in revenues, evaluation performed by human labor may be time-consuming, which may result in dissatisfaction on the part of customers [2]. An automatic car damage assessment system has the potential to cut down on human labor and insurance claim fraud. As a result of recent advancements in deep learning, in particular convolutional neural networks, artificial intelligence is currently a viable option for the construction of the system. The system can accurately identify and classify different types of damages such as dents, scratches, and cracks in a matter of seconds. This technology has the potential to revolutionize the insurance industry by reducing costs and improving customer experience.

1.2 Motivation

In the automotive field, detecting and identifying vehicle components is crucial for improving performance and ensuring safety when carrying out maintenance, repairs, inspections, and insurance. However, efficiently and accurately identifying vehicle components is a significant challenge, especially when dealing with thousands or millions of images from security cameras or autonomous vehicles.

For many years, segmentation-based methods have been used to address this issue. This method allows images to be divided into smaller regions and decisions made about which region belongs to which part of the vehicle, improving the accuracy and efficiency of the identification process.

However, traditional segmentation-based methods often require a lot of data for training and complex models, leading to difficulties in implementation on real systems. Therefore, it is necessary to use new and advanced methods to solve this problem.

In this study, we used image segmentation methods to identify vehicle com-

ponents with high accuracy and speed. We applied advanced techniques such as the Cascade Mask RCNN model with the Swin Transformer backbone to achieve optimal accuracy and speed. Additionally, we optimized the model training process to reduce training time and resources.

The results of this study will contribute to improving the efficiency and accuracy of identifying and classifying vehicle components, especially in the field of auto insurance. Insurance companies and automakers can apply this technology to enhance their insurance processes and product quality inspections before releasing them to the market. Additionally, this research can also be applied in the field of traffic automation, significantly improving safety and traffic performance on the road.

1.3 Contribution

To address the problem of recognizing car parts in car insurance, this study contributes some important points as follows:

Firstly, this study uses the segmentation method to recognize car parts, in which the Cascade Mask R-CNN model with Swin Transformer backbone is used to achieve high accuracy. The segmentation method allows us to accurately determine the position and size of each part of the car, making the evaluation of post-accident damages more accurate.

Secondly, to improve the accuracy of car part recognition, this study uses the Cascade Mask R-CNN model to recognize car parts. Using multiple stages helps the model learn more complex features of cars and improve the accuracy of the recognition results.

Thirdly, this study uses a large dataset containing many diverse car images, which helps the model learn more features and richer knowledge about car parts, thereby improving the accuracy of the recognition results.

Fourthly, in the model training process, this study uses the learning rate reduction method based on the value of the loss function, helping the learning process to be more stable and avoid the overfitting phenomenon of the model.

In summary, this study has proposed a new approach to recognize and segment car parts using a multi-task neural network. Experimental results show that the proposed method achieves high efficiency in recognizing car parts, significantly improving accuracy compared to traditional methods.

In addition, this study also provides a new dataset with higher quality and complete information about car parts, which can be used in related studies on car part recognition and segmentation.

Furthermore, this study has also tested and evaluated the effectiveness of different network architectures in recognizing and segmenting car parts. These results can provide useful information to the research community about network architectures and how to apply them to the problem of recognizing and segmenting car parts.

Finally, this study contributes to the development and improvement of tools and techniques in the field of image processing and object recognition, especially in the problem of recognizing and segmenting car parts. The results and experiences of this study can be applied in various fields such as production...

Chapter 2

Related Work

2.1 Rule based approaches

In recent years, there have been many studies on the use of machine learning algorithms for the purpose of car damage detection. However, non-machine learning approaches have also been investigated. For example, Jayawardena [3] collected 3D CAD models of undamaged cars and compared these models to photographs of damaged cars. Based on patterns that are not in ground-truth information, their system could identify and evaluate damaged regions. However, one disadvantage of this method is that the performance may be degraded over time as car models change yearly.

2.2 Deep Learning approaches

In "Deep Learning Based Car Damage Classification" by Kalpesh Patil [4] proposes a deep learning approach for car damage classification using convolutional neural networks (CNNs). They hand-crafted a dataset consisting of images belonging to different types of car damage: bumper dent, door dent, glass shatter, headlamp broken, tail lamp broken, scratch and smash. They trained on multiple DL models: Inception, Alexnet, VGG19, VGG16 and Resnet. The results show that the Resnet model achieves the highest accuracy in car damage detection with 88,24%.

In the meantime, a project by Qinghui Zhang and colleagues [5] proposes an enhanced Mask R-CNN algorithm for vehicle damage detection and segmentation. The model employs an optimized residual network (ResNet), and feature extraction is accomplished using the Feature Pyramid Network (FPN). The Anchor in the region proposal network (RPN) percentage and threshold are then modified. Bilinear interpolation in ROIAlign preserves the spatial information of the feature map, and different weights are incorporated in the loss function for different-scale targets. Finally, the results of self-created dedicated dataset training and testing reveal that the enhanced Mask RCNN has a higher Average Precision (AP) value, detection accuracy, and masking accuracy, as well as improved efficiency in addressing traffic accident compensation problems.

The paper "CarDD: A New Dataset for Vision-based Car Damage Detection" by Xinkuang Wang [6] introduced a new dataset for vision-based car damage

detection, called CarDD. The dataset contains over 15,000 car images with various types and degrees of damage annotations, including scratches, dents, and fractures. They experimented on multiple models with backbone ResNet-50 and ResNet101. All methods performed poorly on the APS and APbb S metrics, indicating that existing methods are incapable of recognizing small objects for damage detection and segmentation in automobiles.

In 2022, research, which was conducted by van Ruitenbeek, R.E. and Bhulai, S. [7], evaluated various object detection models with diverse backbones utilizing transfer learning and quantify the effect of various fine-tuning techniques. In addition, the researchers compared the model to domain experts and evaluated its performance in a production environment under the intense lighting conditions of a bright street.

Chapter 3

Proposal Methodology

3.1 Cascade Mask RCNN model

The cascade architecture serves as the foundation for a significant number of the most recent and cutting-edge architectural designs, such as segmentation. The use of cascading has resulted in a very powerful design that has improved the efficiency of a great deal of work. The combination of the Cascade R-CNN and the Cascade R-CNN is known as the Cascade Mask R-CNN. The Cascade R-CNN framework is a multi-stage approach to object detection that addresses both the problem of overfitting during training and the issue of quality mismatching during inference. On the other hand, Mask R-CNN is an extension of the Faster R-CNN .

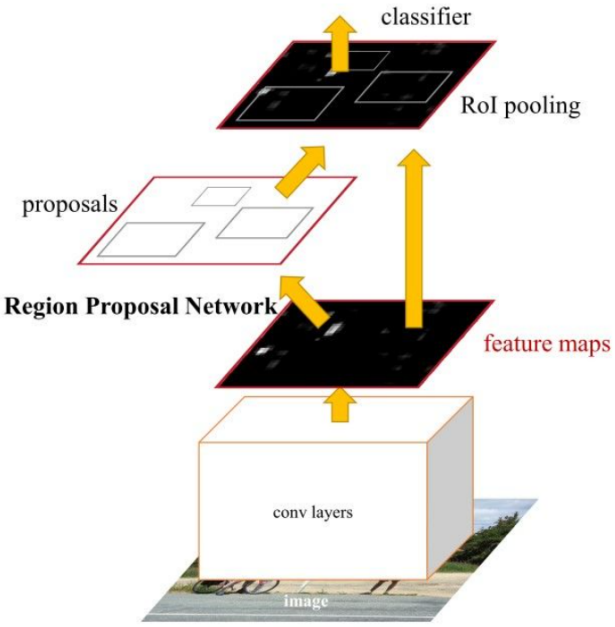


Figure 3.1: Architecture of Faster R-CNN

3.1.1 Faster R-CNN

The Faster RCNN algorithm is a two-stage process that detects objects[8]. The image will be fed into a fully convolutional network known as the Region Proposal Network (RPN), which will extract the region of the image that has the highest likelihood of containing an object based on the Anchor. Following the collection of regions of interest, the ROI Pooling layer is used to extract a feature vector of a fixed length from each of the areas. When the feature vectors have been recovered, they are classified using the Fast R-CNN.

Region Proposal Network

The Region Proposal Network is fully convolutional. It takes an n × n spatial window of the convolutional feature map as input and generates bounding box proposals, each with an objectness score. This score suggests how likely the bounding box contains an object .

RPN uses a sliding window to run across the feature map to generate proposal anchors. At each position, the window slides into, multiple anchors (number of anchors: K) will be generated with different scales and ratios (Figure 3.2). And the total number of anchor boxes with a feature map of size WxH and the number of anchors for each position on the feature map can be given as WxHxK. Anchors will be passed into two convolutional layers to be processed. One layer is the Bounding Box Classifier layer(cls), whose job is to determine whether or not the anchor box contains an object. The other is the Bounding Box Regressor layer (reg), which predicts the offsets between the true object bounding box and the anchor box.

These anchors will be assigned as positive or negative based on the overlap of IoU between anchors and ground truth bounding boxes. The anchor labeling can be interpreted as follows:

- Anchors, which have the highest overlap with ground truth, are labeled as positive.
- Anchors, which have IoU higher than 0.7, are labeled as positive.
- Anchors, which have IoU lower than 0.3, are labeled as negative.
- Anchors, which have IoU between 0.3 and 0.7, will not be used in training.

RPN needs a loss function for training to determine the prediction's effectiveness. The Loss function can be defined as:

$$L\left(\{p_i, t_i\}\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p_i, p_i^*\right) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}\left(t_i, t_i^*\right).$$

Here, i is an index of the anchor, and p_i represents the probability of anchor i whether an object or not. p_i^* is ground truth label. If it is 1, then the anchor is positive; if it is 0, then the anchor is negative. t_i is a vector that contains 4 values to determine an anchor and t_i^* is the ground truth bounding box of an object. L_{cls} is binary cross-entropy for identifying object vs *not* object.For the regression

loss, we use $L_{reg}\left(t_i, t_i^*\right) = R\left(t_i, t_i^*\right)$ where R is the robust loss function (smooth L1). N_{cls} and N_{reg} are normalized by mini-batch size and the number of anchor locations. λ is set by default as 10.

Overlapping, when anchors are generated, is unavoidable. To solve this problem, we use Non-max suppression. This means finding the ones with the highest objectness score and dropping the others.

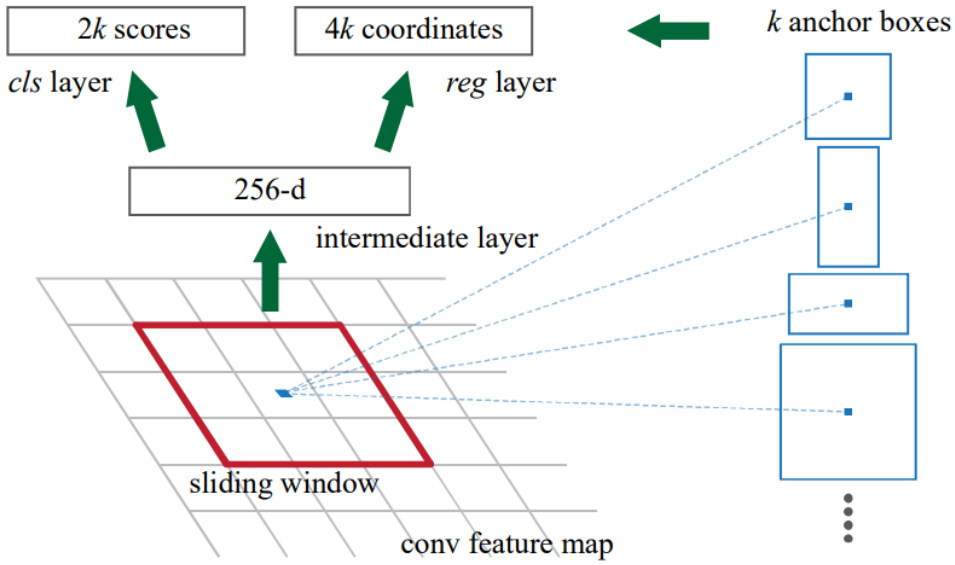


Figure 3.2: Region Proposal Network

Region of Interest pooling

The Region of Interest (ROI) pooling layer is a layer where each ROI was downsampled into a small feature map with a fixed size of H x W by Max pooling operation (Figure 3.3). H and W are hyper-parameter that do not depend on any ROI. The purpose of the ROI pooling layer is to extract a fixed-length feature vector from the feature map. And then, the vector will be put into two sibling fully connected layers (FCN) for finding the class and bounding box of objects.

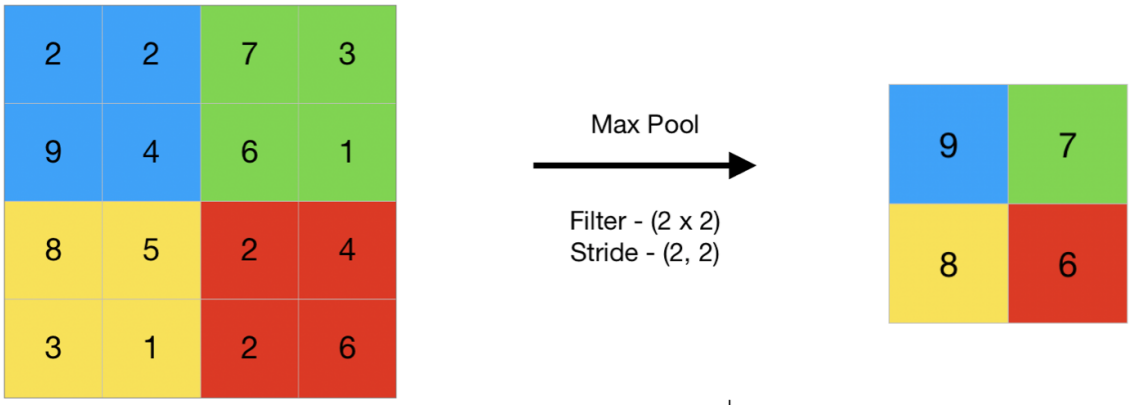


Figure 3.3: Max pooling

Object Detection

After completing the ROI pooling operation, we get multiple fixed-size feature map outputs. These feature maps will be flattened and passed through two fully connected layers to do two tasks: classify objects with N+1 classes (N is the total amount of classes, and 1 is the background) and find offset coordinates of objects' bounding boxes. This head works the same as two FCNs in RPN.

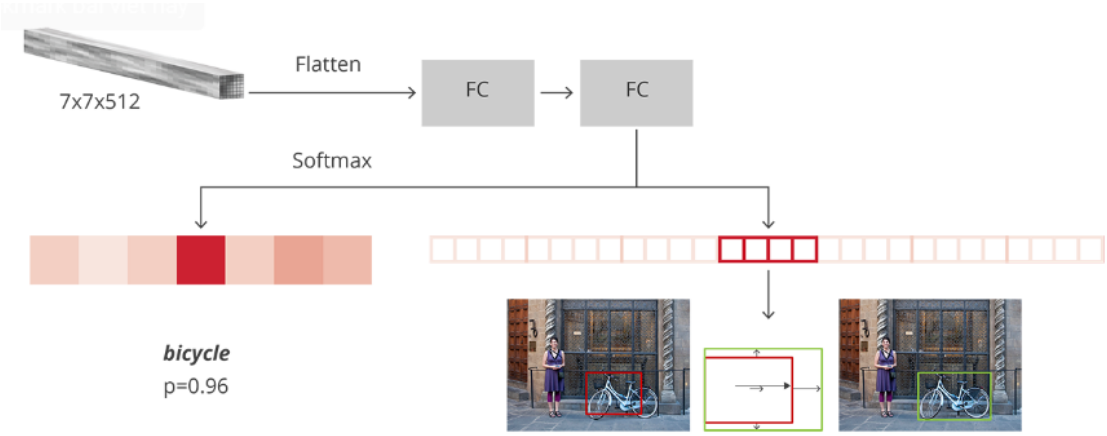


Figure 3.4: Fast R-CNN Detection Head

3.1.2 Feature Pyramid Network

The Feature Pyramid Network (FPN) is a deep neural network model used in computer vision and object recognition. It was introduced in 2017 [9]. FPN was developed to address the challenge of efficient feature extraction from images with varying resolutions, which is necessary for many computer vision applications such as object detection, object recognition, and object localization.

FPN works by building a pyramid system of features with different resolutions from the input images. It uses a backbone to extract features from these images and then synthesizes them to create a multi-level pyramid of features. Specifically, FPN generates a set of features that are reconnected from pre-trained convolutional networks such as ResNet or VGGNet.

One of the key features of FPN is its ability to propagate information from high-resolution features to low-resolution features, and vice versa. This allows FPN to help the model recognize objects of varying sizes, from small objects to large objects in the same image. High-resolution features are used for the precise localization of small objects, while low-resolution features are used for localizing large objects.

3.1.3 Mask R-CNN

In the field of computer vision, the Mask R-CNN [10] is a frequently encountered deep learning architecture that is used for instance segmentation. The Mask R-CNN is an extension of Faster R-CNN. It adds a branch for predicting an object mask in parallel with the branch that is already from Faster R-CNN for bounding box identification. The extended branch is a convolutional network that creates masks from the positive areas of the proposed region. The general structure can be depicted by looking at Figure 3.5.

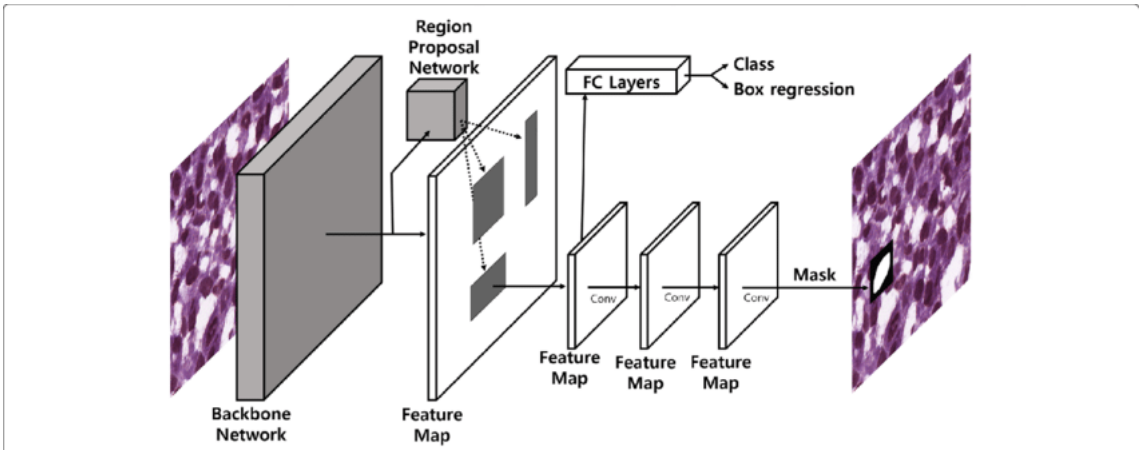


Figure 3.5: Overall architecture of Mask R-CNN

The Region of Interest alignment layer (ROI align) and the Instance Segmentation Head are two new features added to the Mask R-CNN. These features are distinct from those of the Faster R-CNN.

ROI aligns

ROI align was invented to replace the ROI pooling layer, because ROI pooling quantizes a floating-number RoI to the discrete granularity of the feature map. ROI pooling creates a misalignment between RoI and the extracted features. This will reduce a lot of information for the segmentation task. To solve this issue, ROI align uses bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each. RoI bin. Figure 3.6 demonstrates this operation.

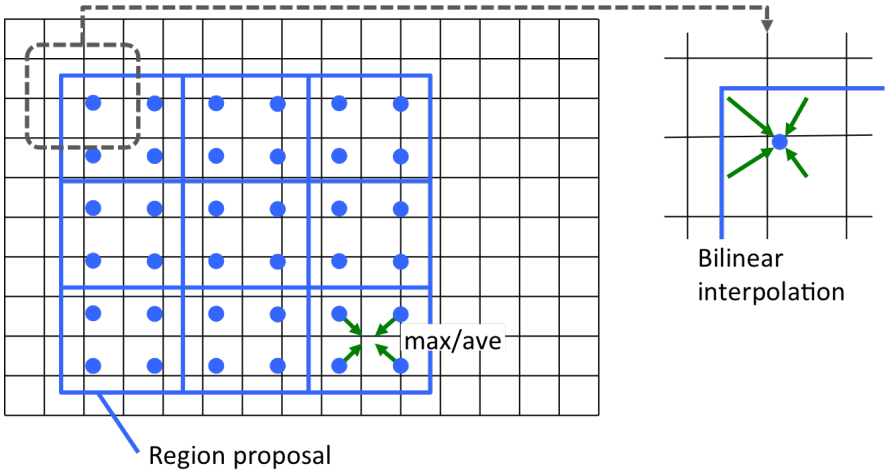


Figure 3.6: ROI align uses bilinear interpolation to compute the exact value of Region of Interest.

Instance Segmentation

The mask branch of the Mask RCNN is a straightforward structure; when the output of ROI align will be processed through a fully convolutional network to masking object in a Region of Interest. Mask R-CNN's multi-task loss function combines classification, localization, and segmentation mask losses: $L = L_{cls} + L_{box} + L_{mask}$. While L_{cls} and L_{box} has been defined in RPN, L_{mask} can be interpreted as below:

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} \left[y_{ij} \log \hat{y}^k_{ij} + (1 - y_{ij}) \log(1 - \hat{y}^k_{ij}) \right]$$

Where y_{ij} is the label of a cell (i, j) in the true mask for the m x m region;$\hat{y}^k_{ij}$ is the predicted value of the same cell in the learned mask for the ground-truth class k.

3.1.4 Cascade Networks

The implementation of cascading architecture has proven to be a highly effective strategy in enhancing the efficiency of various operations. Cascade Mask R-CNN is a fusion of the Cascade R-CNN and Mask R-CNN models. The Cascade R-CNN [11] is a framework for object detection that addresses the issues of overfitting during training and quality mismatching during inference through a multi-stage approach. The Hybrid Task Cascade (HTC) [12] is a cascade approach employed to improve the efficacy of instance segmentation. In contrast to Cascade Mask R-CNN, the HTC approach involves the integration of object detection and semantic segmentation branches, resulting in a multi-stage processing methodology.

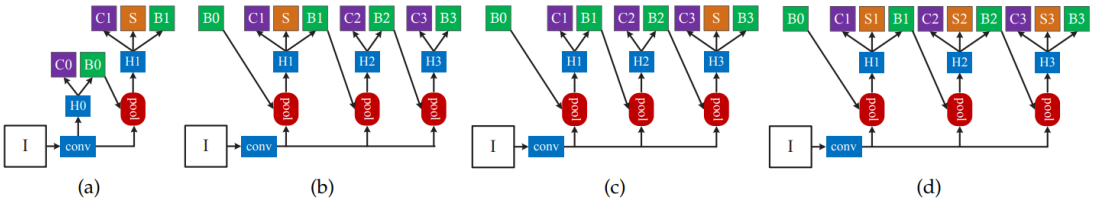


Figure 3.7: Cascade Mask R-CNN

The difference between Mask-RCNN and Cascade Mask-RCNN, along with their distinct methodologies for utilizing the instance mask-generating head, is illustrated in Figure 3.7. Here, C stands for Class, B for Bounding Box, S for Segmentation and H for Head. "I" refers to the input image, "conv" stands for the backbone network and FPN, C0 and B0 refer to the RPN, and "pool" to ROI pooling.

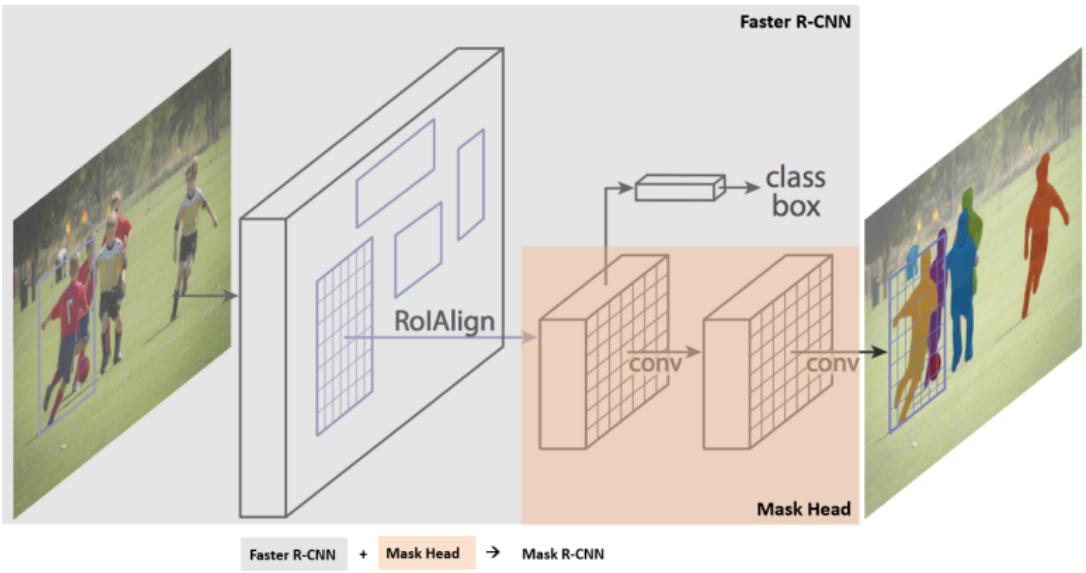


Figure 3.8: Mask RCNN uses a convolutional neural network to mask objects

3.2 Backbone Swin Transformer

3.2.1 Transformer

The Transformer model design avoids recurrence and relies solely on an attention mechanism to establish global dependencies between input and output. Prior to the emergence of Transformers, the prevailing models for sequence transduction relied on intricate recurrent or convolutional neural networks, which comprised both an encoder and a decoder. The Transformer model also utilizes an encoder and decoder architecture. However, it replaces recurrence with attention mechanisms, which results in a higher degree of parallelization compared to techniques such as RNNs and CNNs.

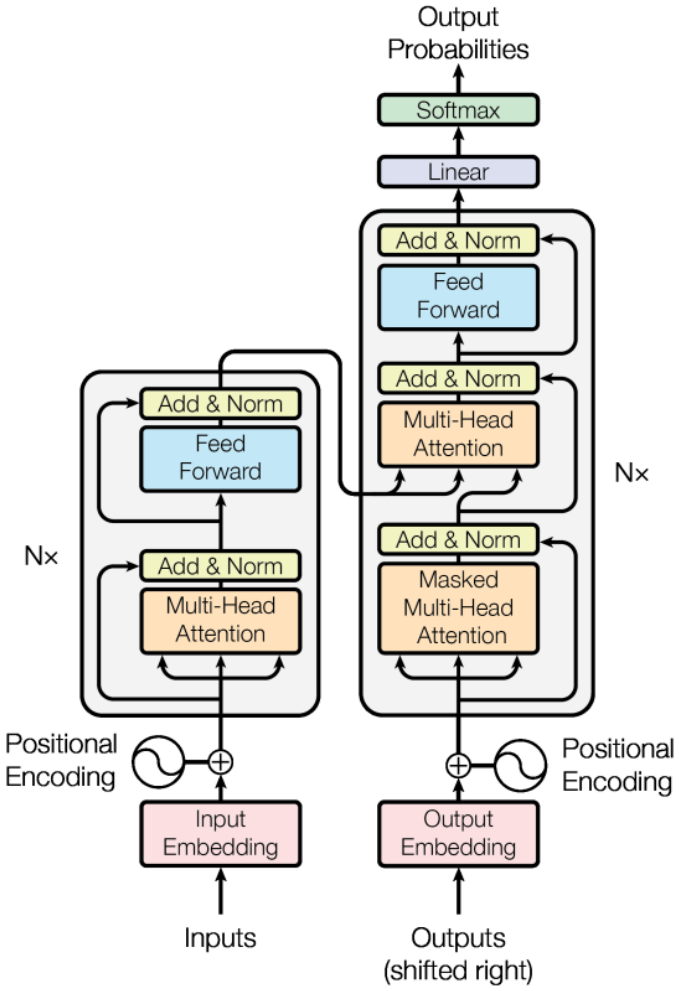


Figure 3.9: Transformer Architecture

Figure 3.9 shows the Transformer Architecture, which was interpreted in the original paper [13]. It unveils the architecture consisting of an encoder and a decoder layer. The Encoder layer contains a Multi-head Self Attention layer (MSA) and a Feed-forward layer. The Decoder comprises three sub-layers: MSA, masked MSA, and a Feed-forward layer. The masked MSA is adjusted MSA where parts of the attention mechanism have been masked out (set to $-\infty$), in this case masking out all connections between a word and future words.

Positional Encoding

Positional encoding defines the location or position of an entity in a sequence by assigning a unique representation to each position. Each position/index is mapped to a vector in the transformers' positional encoding scheme. Consequently, the output of the positional encoding layer is a matrix in which each row represents an encoded object of the sequence, along with its positional information. Sine and cosine functions of different frequencies determine the positional encoding:

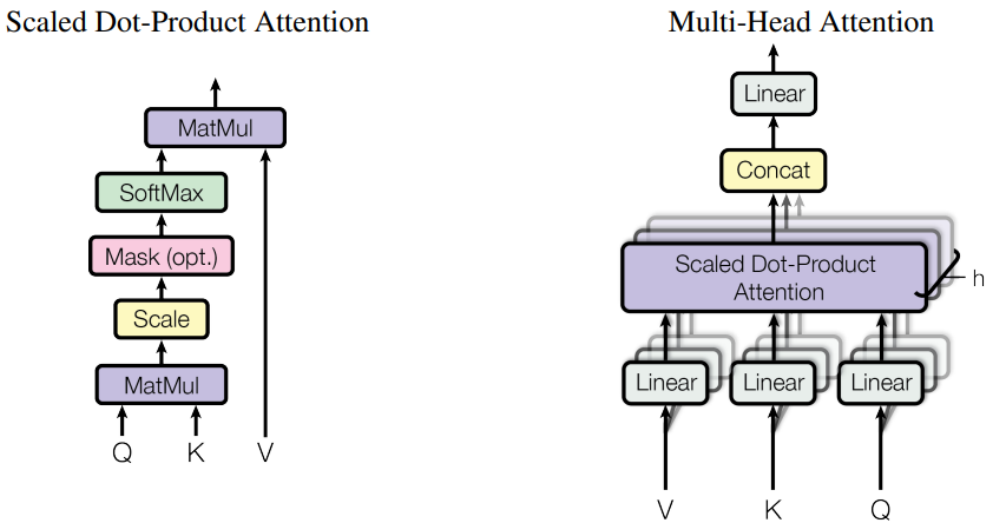


Figure 3.10: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention

$$\mathrm{PE}_{(pos,2i)} = \sin(\mathrm{pos}/10000^{2i/d_{model}})$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos(\mathrm{pos}/10000^{2i/d_{model}})$$

The Formula above is how positional encoding is calculated; where pos is the position of the word in the sentence, d_{model} is the dimension of the output embedding space, i is used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions.

Scaled Dot-Product Attention

The Scaled dot-product attention involves taking queries and keys of dimension d_k, and values of dimension d_v as inputs and performing a dot-product computation between the queries and keys as the initial step. The outcome is subsequently adjusted by the square root of d_k resulting in the generation of attention scores. Subsequently, the inputs are loaded into a softmax operation, resulting in a collection of attention weights. Ultimately, the attention weights are employed to scale the values via a multiplication operation with weights. The whole procedure can be interpreted under the following math equation, where and represent the queries, keys, and values, respectively:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\mathrm{V}$$

Multi-Head Attention

In self-attention, only one portion was focused on; consequently, only that portion has global relationships with other portions. Multi-Head Self Attention(MSA) was used to establish a global relationship between all parts. The outputs of independent attention are then concatenated and transformed linearly into the desired dimension. MSA procedure was illustrated in Figure 3.10 or following Equation :

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O$$

$$\text{where } head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Residual Connection

Residual Connection is used in every "two sub-layer" layer of Transformer. The residual connections contribute in network training by enabling gradients to travel directly through the networks.

Feed Forward Network

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

3.2.2 Vision Transformer

Vision Transformer was introduced in the original paper in 2020 [14]. The vision transformer architecture used the transformer architecture in a way that was similar to a transformer-based text predictor. Each square patch of an image was like a word in a text, and different parts of the image were focused on. An enhanced version of ViT called Swin Transformer was brought up in 2021 [15]. Swin Transformer uses a shifted windowing scheme to allow for cross-window attention connections. This helps the global attention of the image much better.

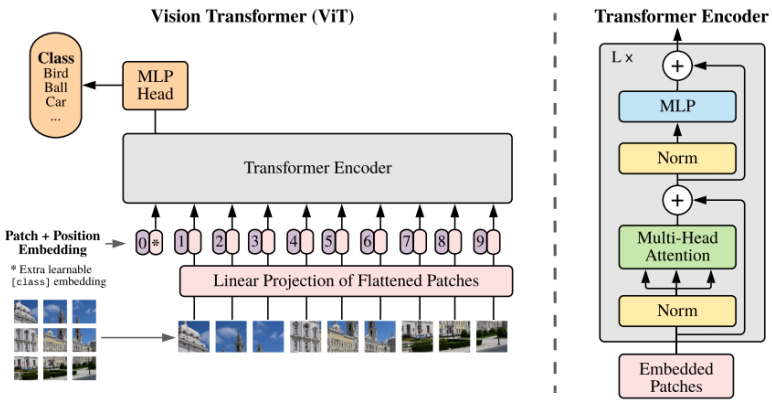


Figure 3.11: Vision Transformer Architecture

Patches + Position Embedding

As shown in Figure 3.11, A 2D image will be split into a sequence of flattened 2D patches in this process. Each patch has a PxP size. The number of patches will be determined by $N = HW/P^2$; where N is the number of patches and (H, W) is the resolution of the original image. Each element that has been flattened is inserted into a linear projection layer, which produces the so-called "Patch embedding." Position embeddings are then linearly added to the collection of image patches to preserve the positional information of the images. According to the position of the image patch, an extra learnable embedding is prepended to the sequence.

Vision Transformer Encoder

The Transformer Encoder is used for calculating Attention between Patches. Like the Encoder of the original Transformer, the Encoder in ViT consists of a multi-headed self-attention block, MLP block, and Norm layer for every sub-layer. But in ViT, the Norm layers are applied before the MSA and MLP blocks. Residual connection is still used after every sub-layer. The MLP contains two layers with a GELU non-linearity.

3.2.3 Shifted Window Transformer

Figure 3.12 depicts the Architecture of the Tiny version of the Swin Transformer. To begin, the input image is an RGB image of size HxWx3, which is separated into patches identical to Vision Transformer, with each patch having a size of 4x4 and is turned into a vector of length 4x4x3 = 48 to load into the Swin Transformer. There are four versions of the Swin Transformer that introduces in the original paper[15]:

- Swin-T: C = 96, layer numbers = {2, 2, 6, 2}.
- Swin-S: C = 96, layer numbers = {2, 2, 18, 2}.
- Swin-B: C = 128, layer numbers = {2, 2, 18, 2}.
- Swin-L: C = 192, layer numbers = {2, 2, 18, 2}.

where C is the channel number of the hidden layers in the first stage.

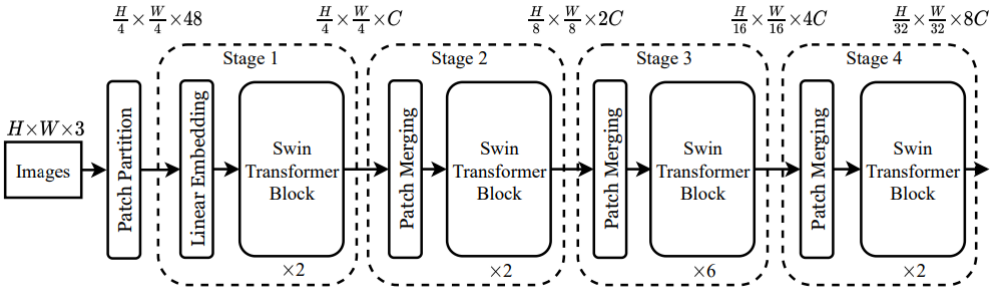


Figure 3.12: Swin Transformer Architecture

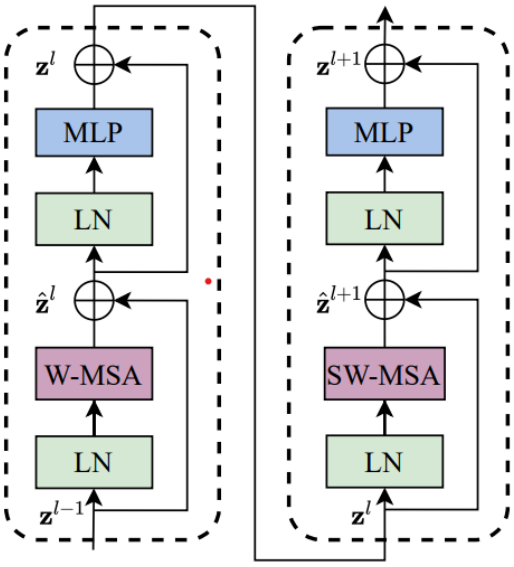


Figure 3.13: Two Transformer Blocks

Swin Tranformer Stages and Patch Merging

At Stage 1, The Linear Embedding layer converts the original vector space to another vector space with dimension C, which is then processed through a few Swin Transformer Blocks, resulting in a total of $\frac{H}{4} \times \frac{W}{4}$ Transformer tokens (also known as patches). At Stages 2, 3 and 4, each Stage consists of 2 main components, a Patch Merging class and a few Swin Transformer Blocks. As the network goes deeper, the number of tokens is reduced by patch-merging layers to provide a hierarchical representation. The Patch Merging layer is in charge of lowering the number of tokens by merging four patches into a single patch, hence the number of tokens when traveling through Stage 2 will be $\frac{H}{8} \times \frac{W}{8}$ and the length of 1 token will be 4C dimension (due to combining four pathways into one). The tokens will then be sent via a Linear layer to lower the dimension to 2C before proceeding through multiple Swin Transformer Blocks. The output of each Stage is $\frac{H}{16} \times \frac{W}{16} \times 4C$ and $\frac{H}{32} \times \frac{W}{32} \times 8C$, as with Stages 3 and 4.

Shifted Window

Swin Transformer Block, unlike Vision Transformer, employs multi-head self-attention (MSA) on a local window region rather than the entire image. The right side of Figure 3.13 shows two consecutive Swin Transformer Blocks, W-MSA and SW-MSA, which are multi-head self-attention in Transformers with "regular window" and "sliding window" mechanisms, respectively. According to the graphic, the input of the block will be transferred through Layer Norm (LN) and then through W-MSA (or SW-MSA.) and MLP, with a residual connection in between.

In the window-based self-attention module, attention calculation is done locally within patches inside each window boundary (A window contains M $\times$ M patches). But this method lacks connections across windows, which lower the model performance. The Shifted Window kicks in to solve this issue. The idea

is : Shift the window by a factor M/2, where M is window size (Figure 3.14). In the original paper[15], Swin Transformer uses cyclic shift to reduce computation heavy. That means it moves patches into an empty slots to complete a window.

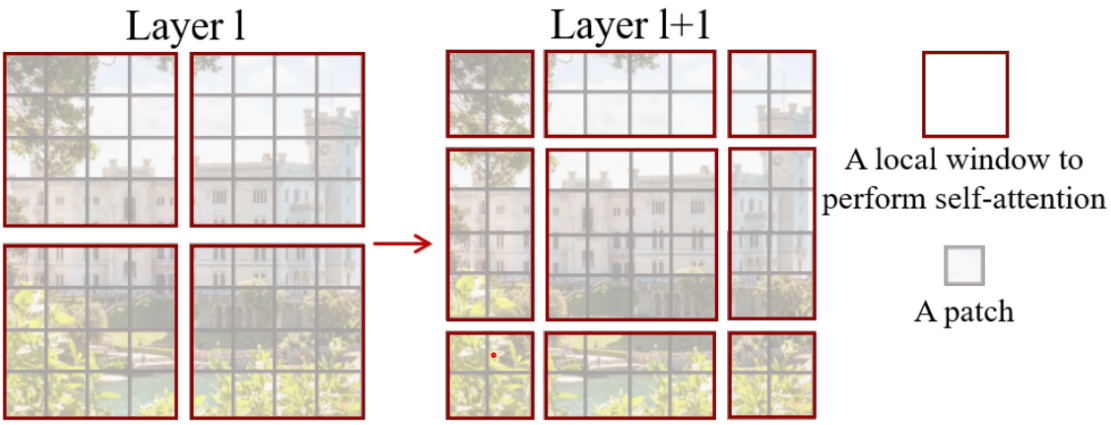


Figure 3.14: Shifted Window

3.3 Implementation Method

3.3.1 Implemented Model

After some research and experiments with our Mentor, we decide to choose Cascade Mask R-CNN as our instance segmentation model. We also use Swin Transformer integrates FPN as the backbone for our model.

FPN is used to generate a feature pyramid that captures features at multiple resolutions, which can be used for detecting objects of different scales in an image. The FPN in Swin Transformer is typically added on top of the output of the last transformer stage, which already captures global contextual information. FPN then enhances the Swin Transformer by providing additional contextual information at different scales, which is important for accurate object detection. The combination of Swin Transformer with FPN allows the model to capture both local and global contextual information from the transformer layers and also capture multi-scale features through the FPN. This enables the model to effectively detect objects of varying sizes in an image, making it highly accurate in object detection tasks.

As depicted in Figure 3.15, our implemented Model uses Swin-B Transformer combine with FPN as a feature extractor. Images go into the Swin Transformer model. Output from each Stage goes to Topdown Feature Pyramid Network. The feature maps from different levels of the FPN pyramid are combined to generate a single set of fused feature maps, which capture both local details and global context at different levels of granularity. The output feature map from FPN goes into the RPN block and ROI aligns each Stage of Cascade Mask R-CNN. The model contains three identical detection stages. Each stage consists of a class head, bbox head and mask head.

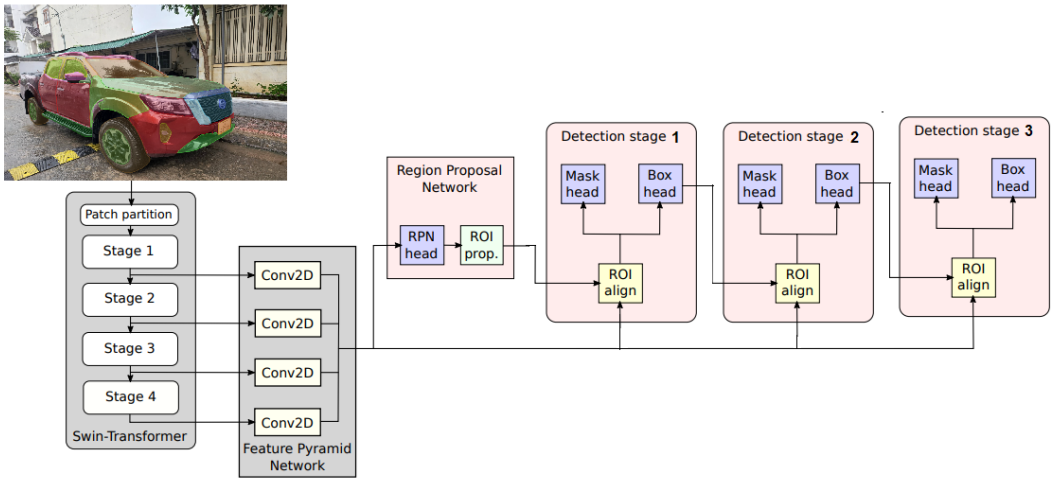


Figure 3.15: The overview structure of implemented Model

3.3.2 Source Code Implementation

We use MMdetection [16] as our base implementation. MMDetection is an object detection toolbox that contains a rich set of object detection, instance segmentation, and panoptic segmentation methods as well as related components and modules. MMDetection was written mostly on Pytorch, which is created by Python language.

3.4 Evaluation Metrics

When working with object detection or segmentation, we need evaluation metrics to measure the performance of our model. One of the most often used metrics for object detection, instance segmentation, and semantic segmentation is intersection over union (IoU)[17], which is interpreted in the formula below, where B_p is the predicted bounding box and, B_gt is the ground truth bounding box.

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

Using only IoU is impractical when measuring multiple object detection. Mean Average Precision is a better evaluation metric. It uses the concept of Precision and Recall .

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Here, TP (True Positive) is the result in which the model predicts the positive class accurately. FP (False Positive) is the number of predicted bounding boxes that do not identify any ground truth boxes accurately. FN (False Negative) is the number of ground truth boxes that have not been correctly detected. In summary, Precision is the ratio of successfully recognized bounding boxes to anticipated boxes, whereas recall is the percentage of properly identified boxes to the number of boxes in the dataset.

The mean Average Precision (mAP) calculates the Precision-Recall curve's AUC [17]. Higher values mean better instance segmentation algorithms. If accuracy remains high as recall grows, an instance segmentation algorithm is good. mAP^{50} means mAP with IoU cutoff at 0.5. mAP^{70} means mAP with IoU cutoff at 0.7.

$$mAP = \frac{1}{N}\sum_{i=2}^{N}(r_i - r_{i-1})\frac{p_{i-1} - p_i}{2}$$

Chapter 4

Experiments

4.1 Dataset

4.1.1 Dataset Description

With the help of the company where one of us works, I got permission to use the company's dataset and add more newly prepared data myself to increase the diversity of the dataset. We use AICycle's automotive image data set, which includes images of different car manufacturers and photos with auto damage by type. Specifically include 25,606 images in which the train set data is 22951 images spread over 62 popular car models in Vietnam such as Toyota Innova, Mitsubishi Xpander, KIA Sorento,...Some examples in the dataset are shown in Figure 4.1.

Figure 4.1: Sample in dataset

The test set data has 2755 images in which 24 vehicle lines are added as test data. In addition, out of a total of 22,951 images in the training set, there is additional information about the damage as follows:

Table 4.1: Training Data Set

Training Data Set	
Damaged name	Number of photos
Dent, flatten (thumb)	5684
Cracked	533
Broken, punctured, torn	2491
Scratched	16027
Shed	403

Table 4.2: Testing Data Set

Testing Data Set	
Damaged name	Number of photos
Dent, flatten (thumb)	684
Cracked	46
Broken, punctured, torn	264
Scratched	2169
Shed	56

4.1.2 Dataset Preprocessing

We conduct web scraping to collect data from some sources on the Internet, such as:

- Facebook: Data is collected from Facebook, including diverse images of new undamaged cars to serve for processing car parts. Additionally, there are images from accidents to include damages and scratches.
- Google: Similar data is collected from Facebook.

Moreover, we are supported by a company to obtain a portion of data from car insurance companies to increase the number of specific images of damages and car parts.

After preparing a large number of photos, we labeled the car parts consisting of 68 parts. For example: Roof, Front bumper, Logo,...(Figure 4.2) Manual labeling helps us better identify and understand vehicle parts.(Figure 4.3)

(a) 1a (b) 1b

Figure 4.2: Sample in dataset

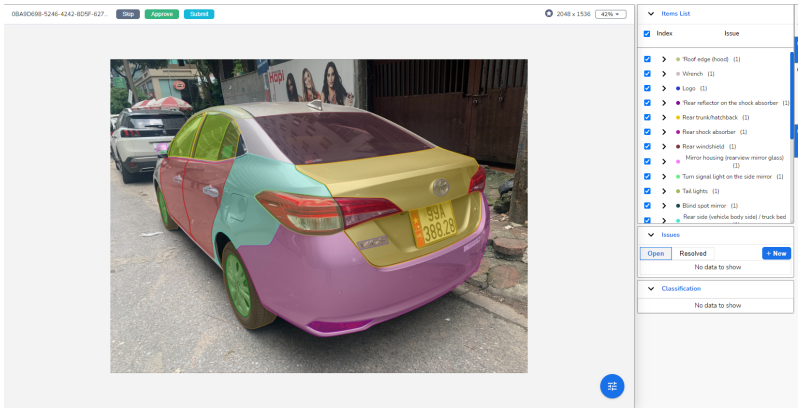


Figure 4.3: Labeling Data

Finally, from the labeled data, we output the JSON file. In there:

- **id**: the id of the annotation.
- **image_id**: id of the image containing the object.
- **category_id**: id of the object (the list of ids listed in the categories field).
- **segmentation**: contains information about the x, y coordinates for the polygons surrounding the classes.
- **area**: bounding box area of the object.
- **bbox**: coordinates of the bounding box relative to the object (x_top left, y-top left, width, height).

The training pipeline is a sequence of preprocessing steps applied to a training dataset in preparation for model training. The steps in the training pipeline can vary depending on the specific requirements of the problem and the model architecture used. Here are the preprocessing steps commonly used in the training pipeline for object detection:

Training Pipeline:

- Load Image From File: This is the first step in the training pipeline, where the training images are loaded from the image file into memory.
- Load Annotations: After loading the image, the next step is to download the bounding box information of the objects in the image from the data file containing the annotation information.
- Random Flip: This step is used to generate new versions of the data from the training images by flipping the images horizontally or vertically. This enhances the training data and improves the generalizability of the model.
- Auto Augment: This step is used to apply automatic data enhancement techniques such as color enhancement, rotation, zoom, etc. to generate new data versions and diversify the training data.
- Normalize: This step is used to normalize the values of the pixels of the training image to return to a certain range of values. Usually, the mean and standard deviation of the pixels is calculated based on the entire training data set.
- Pad: This step is used to bring the training images to the same size by adding white pixels to the uninformed positions in the image.
- DefaultFormats Bundle: This step is used to reformat the training images and bounding boxes of the objects in the image into the input format of the model.
- Collect: The final step of the training pipeline is to collect all the preprocessed training data and put them in a batch to use for training the model.

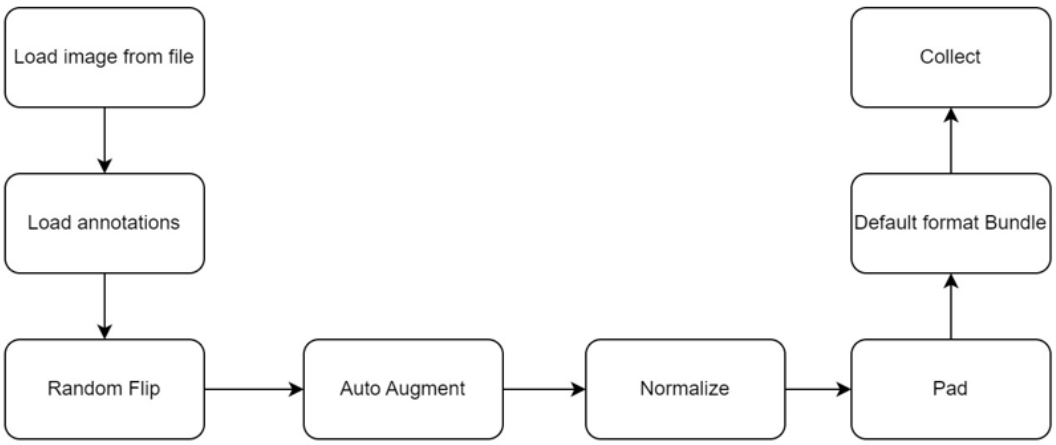


Figure 4.4: Training Pipeline

Test pipeline: CascadeRCNN's test pipeline consists of two main steps: Load Image From File and Multi-Scale Flip Aug.

- Load Image FromFile takes as input the path to the image file and returns the image tensor with the original size.
- Multi-Scale Flip Aug is used to apply different data enhancement techniques to the input image:
  - The first step of MultiScale Flip Aug is resizing to scale the image to the desired size.
  - RandomFlip is applied to invert the random image vertically or horizontally to generate more diverse data.
  - Pad is used to add zero value to the missing parts of the image to ensure that the image size after cropping does not change.
  - Collect is used to collect the converted images into a batch to feed into the model for prediction.

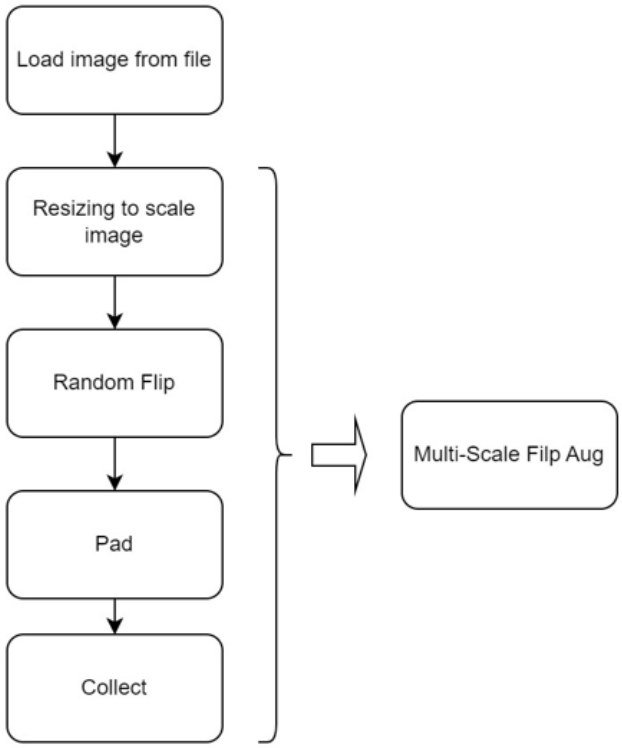
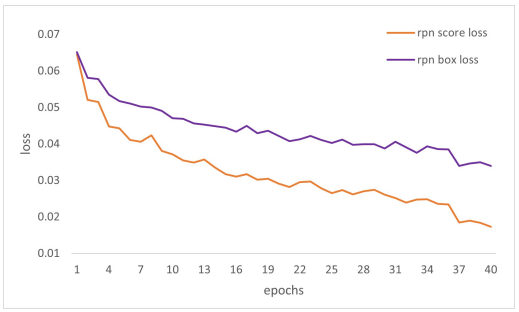


Figure 4.5: Test Pipeline

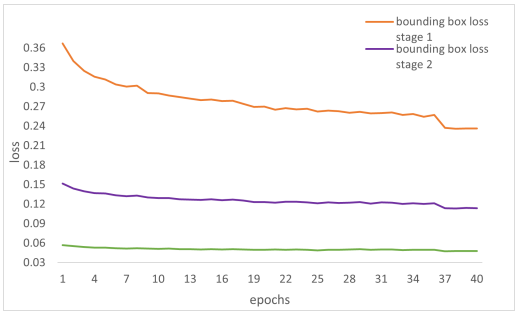
4.2 Result

We used a computer with AMD Ryzen 7 5700G with Radeon Graphics NVIDIA RTX A4000. We spent 3 days for training. An effective system when applied to mid- and panoramic shots of a car. Car parts are detected by boxes and corresponding segments.

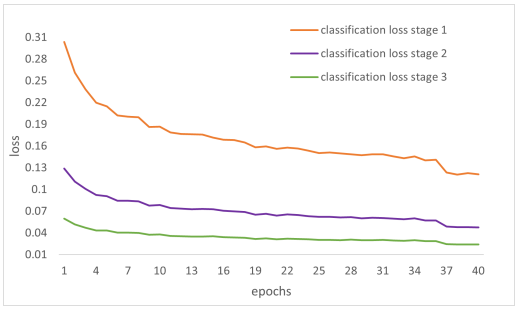
Our training process is depicted in Figure 4.6 and Figure 4.7.



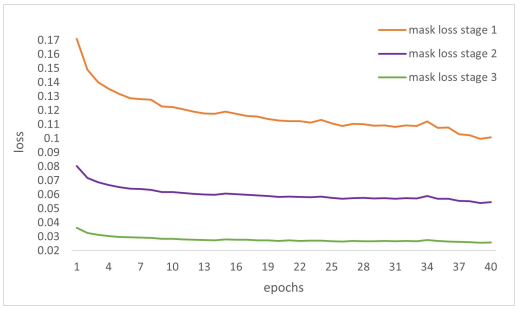
(a) RPN loss



(b) Bounding box loss



(c) Class loss



(d) Mask loss

Figure 4.6: Loss over epochs for SwinB-Cascade Mask RCNN

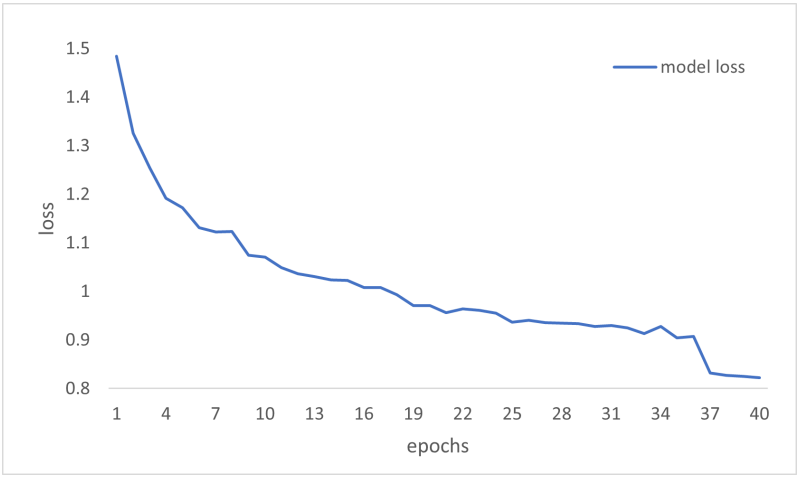


Figure 4.7: Total loss of Model

As in Table 4.3 shows, Cracked damage has the highest recall and mAP with the score of 0.74 and 0.547, respectively. In constrast, Shed has the lowest Recall and mAP with the score of 0.282 and 0.083. The others damage types are quite moderate.

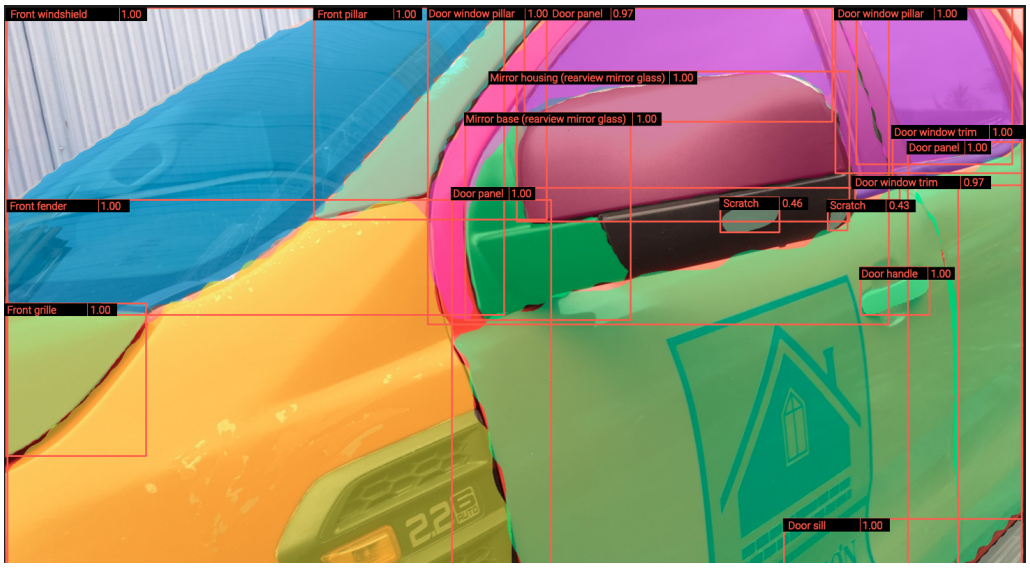
Table 4.3: Result on each type of Damages

Result		
Damaged name	Recall	mAP
Dent, flatten (thumb)	0.586	0.421
Cracked	0.740	0.547
Broken, punctured, torn	0.415	0.117
Scratched	0.584	0.380
Shed	0.282	0.083

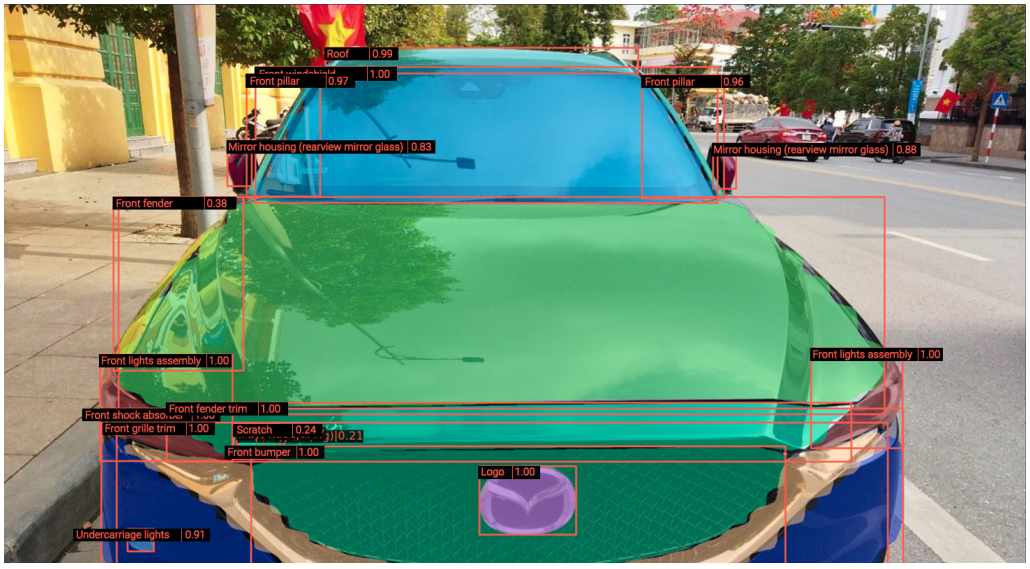
We have some results 4.4 of other models to compare. The result shows that HTC with Resnext101 backbone performs the best object detection performance with a mAP50 score of 0.838. In the mean while, our model - Cascade Mask RCNN with SwinB backbone, achieved the best result among the tested models in the segmentation task with an mAP of 0.817.

Table 4.4: Comparison between Models

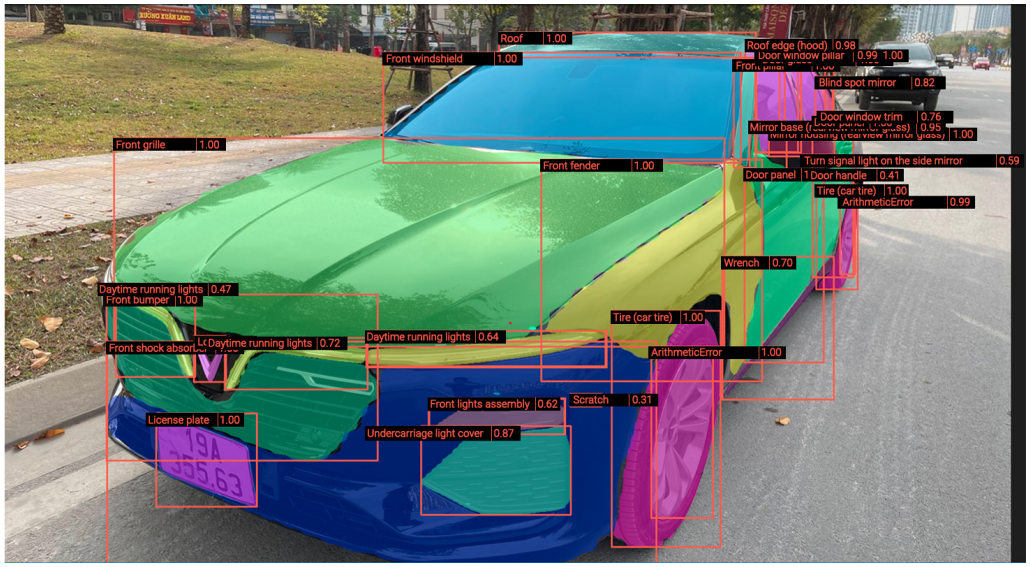
Comparision between Models				
Model	bbox mAP 50	bbox mAP 75	seg mAP 50	seg mAP 75
Resnext101CascadeMaskRCNN	0.817	0.721	0.797	0.650
Resnext101 HTC	**0.838**	**0.724**	0.805	0.656
Swin-T Cascade Mask RCNN	0.817	0.699	0.801	0.661
Swin-B Cascade Mask RCNN	0.836	0.771	**0.817**	**0.705**



(a) Result 1



(b) Result 2



(c) Result 3

Figure 4.8: Detection Results

Chapter 5

Conclusion

Car damage detection is an important application of computer vision that has many practical uses, such as insurance claim processing, car inspections, and vehicle maintenance. In this thesis, we introduces a solution for classifying car damage using deep learning. We uses Cascade Mask RCNN with back bone Swin Transformer, which intergrates Feature Pyramid Network. On testing dataset, the model achieved mean average precision of 81,7 % on segmentation , which indicates the effectiveness of the model.

Overall, our study demonstrates that the Cascade Mask R-CNN model is a powerful tool for car damage detection, with significant potential for real-world applications in areas such as car insurance and maintenance. Future work could focus on improving the model's performance on specific types of car damage, as well as extending the dataset larger.

References

[1] FPT.AI [2022], 'Fpt.ai cardamage – buoc tien moi trong bao hiem xe hoi'. "https://fpt.ai/vi/fptai-cardamage-buoc-tien-moi-trong-nganh-bao-hiem-xe-hoi".

[2] LAB, T. [2022], 'How to reduce insurance claims leakage—and the loss ratio—via standardization, business intelligence, and robotic process automation (rpa)'.

[3] Jayawardena, S. [2013], Image based automatic vehicle damage detection, PhD thesis, College of Engineering and Computer Science (CECS).

[4] Patil, K., Kulkarni, M., Sriraman, A. and Karande, S. [2017], Deep learning based car damage classification, *in* '2017 16th IEEE international conference on machine learning and applications (ICMLA)', IEEE, pp. 50–54.

[5] Zhang, Q., Chang, X. and Bian, S. B. [2020], 'Vehicle-damage-detection segmentation algorithm based on improved mask rcnn', *IEEE Access* **8**, 6997–7004.

[6] Wang, X., Li, W. and Wu, Z. [2023], 'Cardd: A new dataset for vision-based car damage detection', *IEEE Transactions on Intelligent Transportation Systems* .

[7] van Ruitenbeek, R. and Bhulai, S. [2022], 'Convolutional neural networks for vehicle damage detection', *Machine Learning with Applications* **9**, 100332.

[8] Ren, S., He, K., Girshick, R. and Sun, J. [2015], 'Faster r-cnn: Towards real-time object detection with region proposal networks', *Advances in neural information processing systems* **28**.

[9] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. and Belongie, S. [2017], Feature pyramid networks for object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 2117–2125.

[10] He, K., Gkioxari, G., Dollár, P. and Girshick, R. [2017], Mask r-cnn, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 2961–2969.

[11] Cai, Z. and Vasconcelos, N. [2018], Cascade r-cnn: Delving into high quality object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 6154–6162.

[12] Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W. et al. [2019], Hybrid task cascade for instance segmentation, *in* 'Proceedings of the IEEE/CVF conference on computer vision and pattern recognition', pp. 4974–4983.

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. [2017], 'Attention is all you need', *Advances in neural information processing systems* **30**.

[14] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. et al. [2020], 'An image is worth 16x16 words: Transformers for image recognition at scale', *arXiv preprint arXiv:2010.11929* .

[15] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. and Guo, B. [2021], Swin transformer: Hierarchical vision transformer using shifted windows, *in* 'Proceedings of the IEEE/CVF international conference on computer vision', pp. 10012–10022.

[16] Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J. et al. [2019], 'Mmdetection: Open mmlab detection toolbox and benchmark', *arXiv preprint arXiv:1906.07155* .

[17] Padilla, R., Netto, S. L. and Da Silva, E. A. [2020], A survey on performance metrics for object-detection algorithms, *in* '2020 international conference on systems, signals and image processing (IWSSIP)', IEEE, pp. 237–242.

Appendix

Table 5.1: Average precision (AP) scores for car parts.

Car part	AP	Car part	AP
Roof	0.946	Mirror base (rearview mirror glass)	0.926
Roof edge (hood)	0.924	Door window pillar	0.976
Wrench	0.907	Door pillar	0.942
Front windshield, front fender	0.974	Rear side cover (vehicle body side) / truck bed	0.916
Front grille	0.978	Rear trunk lid trim	0.917
Front bumper	0.955	Door sill	0.97
Spare tire at the back of the car	0	Front fender trim	0.904
Front shock absorber	0.978	License plate	0.986
Front shock absorber grille	0.777	Daytime running lights	0.683
Front shock absorber cover	0.807	Door trim	0.812
Logo	0.979	Front bumper lower cover trim	0.715
Rear reflector on the shock absorber	0.929	Rear shock absorber cover trim LR	0.824
Rear trunk/hatchback	0.979	Front shock absorber cover trim LR	0.643
Rear shock absorber	0.961	Door trim FB	0.628
Rear shock absorber cover	0.92	Front blind spot mirror	0.82
Rear windshield	0.962	Vehicle model name	0.858
Undercarriage lights	0.796	Front grille trim	0.796
Undercarriage light cover	0.92	Door window trim	0.87
Front lights assembly	0.949	Door trim	0.769
Mirror face (rearview mirror glass)	0.983	Tail lights	0.951
Mirror housing (rearview mirror glass)	0.942	Blind spot mirror	0.891
Turn signal light on the side mirror	0.838	Side window	0.958
Front pillar	0.94	Truck bed cover	0.806
Front fender	0.951	Truck bed cover LR	0.842
Front fender cover	0.826	Rear truck bed cover	0.921
Rear shock absorber turn signal light	0.953	Rear truck bed glass	0.915
Rear shock absorber turn signal light cover	0.935	Rear truck bed glass LR	0.949
Rear side (vehicle body side) / truck bed	0.956	Gas tank lid	0.943
Rear pillar	0.943	Door panel	0.976
Rear trunk lid taillights	0.96	Door glass	0.974
ArithmeticError	0.987	Door handle	0.936
Tire (car tire)	0.987		