# TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

# SALICO GmbH

Final report of the project:

# Emory Pro Car Inspections: AI Vision


| | |
|---|---|
| Authors | Dominik Sylari, Zelong Zheng, Ashwathi Nair |
| Mentor(s) | Utkarsh Siwach |
| Project lead | Dr. Ricardo Acevedo Cabra (MDSI) |
| Supervisor | Prof. Dr. Massimo Fornasier (MDSI) |


02, 2025

**Acknowledgements**

”Cultivate the habit of being grateful for every good thing that comes to you, and to give thanks continuously. And because all things have contributed to your advancement, you should include all things in your gratitude.” – Ralph Waldo Emerson.

**Abstract**

This project aims to improve the efficiency of vehicle inspection using AI-driven computer vision technology. Traditional vehicle inspection relies on manual damage detection and documentation, which is time-consuming and prone to human error. To improve this process, we integrated YOLOv5 for automated damage detection and Optical Character Recognition (OCR) for Vehicle Identification Number (VIN) extraction into the Emory Pro platform. The system is designed for mobile deployments using Android Studio and Flutter, allowing real-time AI processing on the device.

The Damage Detection module automatically recognizes scratches, dents and broken parts on vehicles. The workflow includes YOLOv5 segmentation for extracting car masks, damage detection and localization, and isomorphic transformations for mapping damage locations onto standardized labeled maps. The model was optimized for mobile execution by converting it to the TFLite format, enabling fast inference in the Emory Pro application. The Flutter-based implementation in Android Stuido ensures real-time damage visualization with reduced manual effort.

For OCR-based VIN recognition, a ResNet classification model filters images containing VINs, while a YOLOv5-based detection model isolates text regions. The extracted text is processed using Google ML Kit OCR, ensuring high recognition accuracy even under challenging conditions. The lightweight implementation allows instant VIN extraction on mobile devices, simplifying report generation.

To support efficient data storage and management, we leveraged the AWS cloud with a focus on AWS Lambda for automated file archiving and retrieval. The Lambda feature automatically moves files between S3 Standard and Glacier, ensuring seamless long-term data retention. The system leverages Amazon S3 and Glacier storage to optimize the costs while maintaining accessibility.

Experimental results demonstrate the effectiveness of our approach. The YOLOv5 Large model achieves 96% damage detection accuracy. YOLOv5 large achieves 0% missed segmentation in car segmentation. In addition, the OCR system enhanced with YOLOv5-based preprocessing has high recognition accuracy and robustness, and is able to successfully extract VINs under a variety of lighting conditions and distortions.

By integrating AI-driven damage detection and OCR functionality into Android Studio and Flutter, the project automated vehicle detection, reducing manual effort and increasing efficiency.

# Table of contents

# 1 Introduction

## 1.1 About the company

Emory Pro is a cutting-edge platform designed to revolutionize the way businesses conduct inspections. By replacing traditional paper-based methods with digital solutions, Emory Pro streamlines the inspection process, making it faster, more accurate, and highly efficient. The platform allows users to create detailed digital inspection reports, complete with photos, videos, and annotations, all in real-time using mobile devices or web applications. This eliminates common challenges such as lost reports, time-consuming data entry, and difficulty in analyzing inspection data.

The most important aspect of Emory Pro is its ability to digitize and automate the entire inspection workflow. From adding new items in the warehouse to assigning inspections to field agents, the platform ensures seamless collaboration and real-time updates. Inspectors can easily scan items, mark up issues, and generate comprehensive reports that can be shared instantly with stakeholders via shareable links or automated notifications. Emory Pro is an essential tool for businesses looking to enhance operational efficiency, maintain compliance, and make data-driven decisions.

The inspection process with Emory Pro is straightforward and efficient (Figure1):

- Create a report and add the vehicle details
- Select the view of the car and mark the damages in the labeling map by category
- Add images of the car showing the marked damages

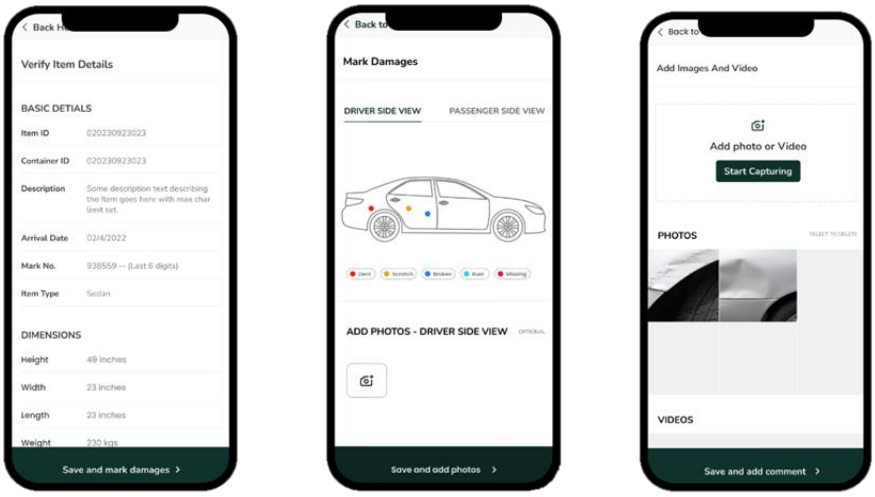


*Figure 1 How to use Emory Pro app*

Emory Pro's user-friendly interface and cross-platform compatibility (iOS, Android, and Web) make it a versatile tool for businesses across various industries, including logistics, automotive, and manufacturing. By digitizing inspections, Emory Pro helps organizations save time, improve accuracy, and stay ahead in a competitive market.

## 1.2 Project Goal

The primary goal of this project is to enhance the efficiency and accuracy of damage detection during vehicle inspections by leveraging AI vision models. As mentioned earlier, Emory Pro's current inspection process involves manual documentation of damages, such as manual labeling, which can be time-consuming and prone to human error. To address this, we propose integrating the YOLOv5 model to automate damage detection. The YOLOv5 model will segment the car and identify major

damages, mapping these detections onto a labeling map. This automated process will introduce a new label specifically for AI-detected damages, reforming the inspection workflow and ensuring a more comprehensive assessment of vehicle conditions. By automating this step, we aim to reduce manual effort, improve detection accuracy, and provide a more detailed and reliable inspection report for stakeholders. In the end, everything will be implemented in Android Studio. It is then provided for use on the app.

The second goal of the project is to develop an Optical Character Recognition (OCR) system capable of accurately identifying and extracting Vehicle Identification Numbers (VIN) from images of vehicles. The model for VIN recognition and extraction must be lightweight and efficient to run on mobile devices, while still maintaining high accuracy. We are aiming to use a pre-trained OCR model, finetune and integrate it into the Flutter application. The model will process images of VINs and output the recognized text for an easier user experience, and faster and more accurate report filling.

The third goal of the project is to leverage AWS services, specifically Amazon S3 and Glacier, to optimize storage costs. By utilizing S3 Standard for frequently accessed files and archiving less frequently used files in Glacier, the project aims to reduce long-term storage expenses. AWS Lambda functions will automate the archival and restoration processes, ensuring seamless movement of files between S3 and Glacier, minimizing manual intervention, and improving operational efficiency. This approach will streamline data management while maintaining accessibility when needed, ultimately enhancing cost-effectiveness.

# 2 Damage detection

## 2.1 Literature Review

**YOLOv5(**You Only Look Once version 5) is a state-of-the-art object detection model developed by Ultralytics. It is an evolution of the YOLO family of models, known for their speed and accuracy in real-time object detection tasks. The model is implemented in PyTorch and is widely used for tasks such as object detection, segmentation, and classification. The official repository for YOLOv5 is available on their page in GitHub. Its key features are high speed and accuracy; support for object detection, segmentation, and classification; easy-to-use API for training and inference.

YOLOv5 works by dividing the input image into a grid and predicting bounding boxes and class probabilities for each grid cell. The model outputs three types of predictions:

- **Bounding Boxes:** The coordinates of the detected objects.
- **Class Probabilities:** The likelihood that the detected object belongs to a specific class.
- **Confidence Scores:** The confidence that the bounding box contains an object.

The architecture of YOLOv5 is composed of three main components. The backbone, CSPDarknet53 (Cross Stage Partial Darknet), is responsible for feature extraction. The neck, PANet (Path Aggregation Network), aggregates features from different layers to enhance the model's ability to detect objects at various scales. The head consists of three detection layers that predict bounding boxes, class probabilities, and confidence scores, enabling the model to accurately identify and localize objects in the input image.

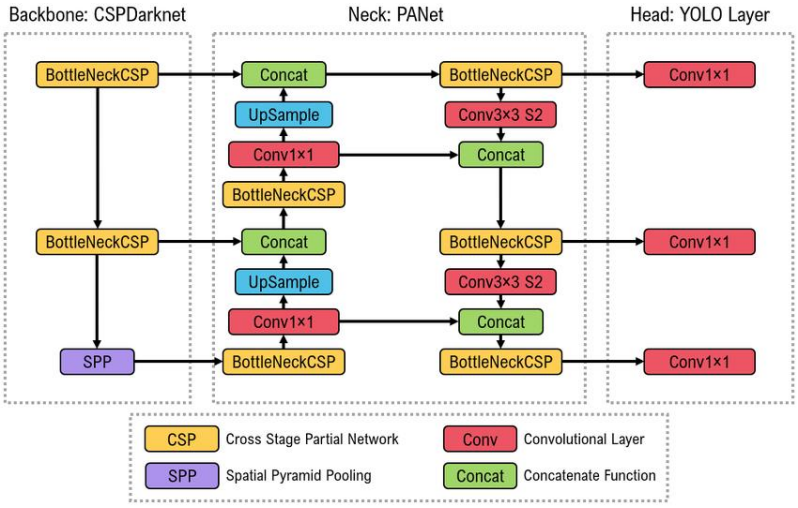


*Figure 2 The architecture of YOLOv5*

**The Homography Transformation Matrix** is a key concept in computer vision and image processing. It is a 3x3 matrix used to map points between two planes in projective geometry, which can represent transformations such as translation, rotation, scaling, and perspective distortions. When applied to images, it allows for the alignment or registration of two different views of the same scene, assuming the views are related by planar surfaces or camera motion without parallax. The matrix is particularly useful for tasks such as image stitching, augmented reality, and camera calibration. Mathematically, the homography $H$ transforms a point $x$ in the source image to a corresponding point $x'$ in the destination image using homogeneous coordinates:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} \text{where} H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \quad (1)$$

This is how the transformation of point $x$ from the plane $\pi$ into the point $x'$ in the plane $\pi'$ looks in the coordinate system.

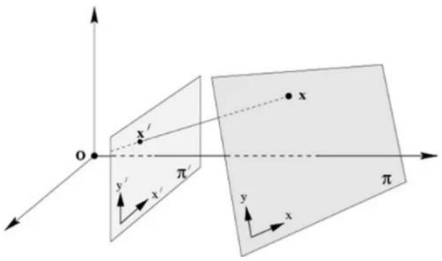


*Figure 3 Homography Transformation*

**Google ML Kit** is a comprehensive machine learning SDK designed to offer pre-built models for mobile applications. The text recognition feature within ML Kit uses advanced computer vision models to detect and extract text from images. It supports both on-device and cloud-based processing, with the on-device model being optimized for real-time performance and privacy.
The OCR process in ML Kit involves several stages: first, image preprocessing techniques are applied to enhance text visibility. Next, the model detects regions of interest containing text and performs character segmentation. Finally, recognized text is returned in a structured format, including bounding boxes, language, and orientation metadata. The ML Kit's lightweight, scalable architecture makes it ideal for mobile applications where computational resources are limited.

To enhance the classification of OCR-extracted text, a ResNet (Residual Neural Network) architecture was integrated into the pipeline. ResNet was introduced by He et al. in 2015 to address the vanishing gradient problem in deep neural networks by introducing shortcut connections, or "residual blocks," which allow gradients to flow more efficiently during backpropagation. In our implementation, ResNet was used to classify extracted text characters or context-specific symbols by recognizing

patterns within the visual and contextual data. Its robustness to noise and ability to generalize well across complex datasets made it an excellent choice for this classification task.

## 2.2 Goal and Framework Design Process

Our goal is to detect the specific location of any damage on a car when the user uploads a photo requiring damage assessment. We then want to mark the corresponding location of the damage on a labeling map, as shown in the figure 4.


*Figure 4 Mark the location on a labeling map*

The entire task consists of six parts, as illustrated in the figure 5. First, the user uploads an image that needs damage detection. Then, Yolov5 segmentation is used to segment out the car mask requiring detection. Once the car mask is obtained, we can proceed to select key points, which enable the computation of the homography matrix between the user-uploaded image and the labeling map. The homography matrix lays the groundwork for subsequent mapping. Next, Yolov5 detection is applied to detect the damaged areas on the car. The center of the bounding box is taken as the damage coordinate. Then, the homography matrix is used to transform this damage coordinate. The transformed coordinate can be mapped onto the labeling map. Finally, the labeling map will automatically display the damage markings.

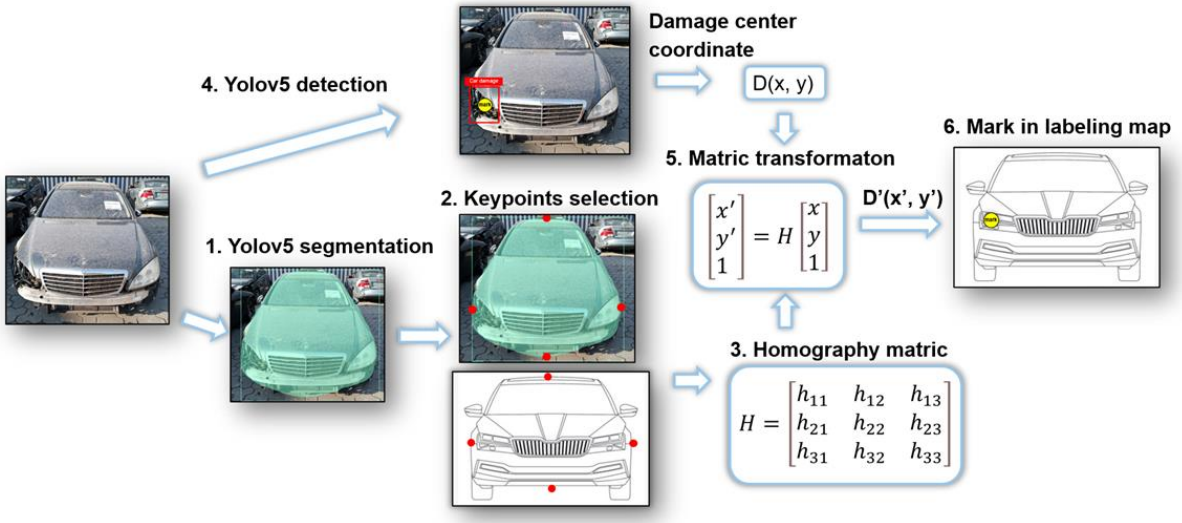


*Figure 5 The overview of the detection task*

## 2.3 Model details

1. First, using YOLOv5 segmentation to obtain the mask of the damaged car in the image (Figure 6).

- The user will upload an image where the damaged area needs to be marked on the labeling map. The image typically contains multiple cars, but the target car is generally located at the center of the image and occupies the largest area. After segmentation, we

will calculate the mask area of each detected car and select the one with the largest area as the region of interest for damage detection.

*Figure 6 The segmentation in the image*

2. Select four key points within the mask region and correspondingly select four key points in the labeling map. This will establish four sets of corresponding key points (Figure 7).

- Key Point Selection: Once the mask region is obtained, identify the leftmost and rightmost coordinate points (X_right, Y_right) and (X_left, Y_left) as two key points. Then, compute the middle vertical line based on X_right and X_left, and find the highest and lowest points on this middle line within the mask region (X_top, Y_top) and (X_bottom, Y_bottom) as the remaining two key points.
- Similarly, in the labeling map, select the leftmost and rightmost coordinate points (X_right_G, Y_right_G) and (X_left_G, Y_left_G). Then, determine the highest and lowest points on the middle vertical line (X_top_G, Y_top_G) and (X_bottom_G, Y_bottom_G).

*Figure 7 Selection of key points*

3. Compute the homography matrix using the four sets of corresponding key points.

- To compute the homography matrix H, at least four pairs of corresponding (x, y) - (x', y') key points are required. The Direct Linear Transformation (DLT) method is used to solve for the homography matrix:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \quad (2)$$

4. Use YOLOv5 detection to identify the exact location of car damage (Figure 8).

- In Emory Pro, the detection of damage is quite specific, including door damage, crushed front ends, and shattered windows, which are severe types of damage. Therefore, we adopted the pre-trained YOLOv5 model on the COCO dataset as a baseline. We then fine-tuned the model using our custom dataset of approximately 3,500 labeled images containing damaged cars.
- Since the training images are large (around 2MB per image) and we use the YOLOv5 large model, we leveraged AWS EC2 cloud computing to train our YOLOv5 model

efficiently. The training was conducted for 100 epochs until the loss converged. The next section will explain how to train the YOLOv5 model using AWS EC2.

- Once the model produced satisfactory results, we applied post-processing techniques: Non-Maximum Suppression (NMS) was used to filter out redundant bounding boxes and keep the best predictions. The center of the bounding box was selected as the damage coordinate point.
- We verified whether the detected damage coordinate point remained inside the car mask region. If a coordinate was outside the mask, it was removed.


*Figure 8 The detection result*

5. Coordinate problem

- The issue we face is that although we can detect the damage location on the user-uploaded photo, we cannot map that damage location onto the labeling map. Specifically, the damage coordinates identified in the uploaded photo cannot be directly applied to the fixed labeling map. As shown in the figure, the size, shape, and viewing angle of the car in the user's photo often differ significantly from those in the labeling map, causing a large positional discrepancy for the corresponding damaged area between the two images.
- To address this, we have considered using homography (a projective transformation) to make the user-uploaded photo appear similar to the labeling map. After this transformation, the shape, size, and viewing angle of the car in both images become similar, as illustrated. Consequently, once the coordinates of the damage in the original image are transformed, they can be directly mapped onto the labeling map. In this way, we can automatically mark the damage detected in the original image onto the labeling map (Figure 9).


*Figure 9 How to solve the mapping problem*

6. Using the homography matrix to transform the damage location and obtain the corresponding damage coordinates on the labeling map.

- Retrieve the previously computed homography matrix H:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \quad (3)$$

- Apply the homography transformation to all valid damage coordinate points p = (x, y) using:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4)$$

This results in the transformed damage marker coordinates p' = (x', y').

7. Mark the damage on the labeling map and upload it to the app user interface for users to view (Figure 10).

- Once the transformed damage marker coordinates p' = (x', y') are obtained, they are marked on the labeling map to indicate vehicle damage.
- The final labeling map with damage marks is then uploaded to the user's mobile device. After the user uploads an image for detection, the entire model runs locally on the mobile device, without the need to upload images to a cloud server. This ensures that no privacy information is leaked during the detection process.


*Figure 10 Mark the damage on the labeling map*

## 2.4 Dataset

### 2.4.1 Data Extraction from AWS into MySQL

To build our dataset, we extracted the necessary data from AWS and stored it in MySQL. We used SQL queries to retrieve specific information required for our analysis.

- **Extracting Item Data:**
  We retrieved item-related details such as item ID, name, markings, organization ID, file names, URLs, and titles. The query ensured that only items with non-empty markings were selected while filtering by a specific organization.
- **Extracting VIN Numbers:**
  A separate query was used to extract vehicle identification numbers (VINs). We selected items with a VIN longer than 13 characters and ranked them by their associated file URLs. This helped us retrieve the top three file URLs for each VIN.
- **Extracting X, Y Coordinates:**
  JSON data containing marking information was processed to extract spatial coordinates. Each marking index was converted into (x, y) coordinates, where 'x' represented row numbers and 'y' represented column numbers. This transformation allowed us to map the markings correctly for further analysis.

### 2.4.2 Image Labeling for Damage Detection

Since publicly available datasets were not suitable for our damage detection model, we manually labeled over 15,000 images using LabelMe and LabelImg (Figure 11). This labeling process involved marking areas of damage on images to create a high-quality dataset for

training our model. The labeled dataset significantly improved our model's accuracy in detecting damages in various images.

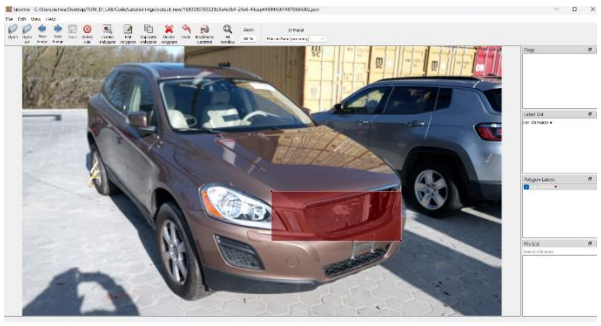

*Figure 11 Labeling tools*

## 2.5 Implementation

### 2.5.1 Python

Yolov5 Segmentation:

1. The pre-trained weights of the Yolov5 large model on the COCO dataset. These pre-trained weights are sufficient to accurately segment cars. Even if a car is heavily damaged, the model can still accurately segment the car.
2. After segmentation, post-processing is applied. Since the images generally contain multiple cars, and the car we need to detect is usually in the center of the image and has the largest area, we calculate the areas of all car masks and retain only the largest car mask.

Homography Transformation Matrix

1. In Python, OpenCV provides cv2.findHomography() to compute the homography matrix.

Yolov5 Detection

1. We first trained the Yolov5 detection model using a custom dataset. The car damage images that Emory Pro needs to detect have some specific characteristics: (1) The image size is relatively large, around 2MB; (2) The images generally capture a full view of the car; (3) The car damage is usually severe, as shown in the image below. Due to these image characteristics, it is difficult to find a dataset that meets these criteria online. Therefore, we downloaded data from the Emory Pro database and used Labelme and LabelImg for annotation, ultimately forming our own custom car damage detection dataset.
2. We attempted to train the Yolov5 Median model using the custom dataset. The initial training was conducted on a PC with an Nvidia 3060 Ti GPU (8GB). The training details are as follows: 40 epochs, batch size = 32, optimizer = SGD, learning rate = 0.01. The dataset consists of 3,049 training images and 369 validation images.

AWS training

After achieving good results with the Yolov5 Median model, we trained the Yolov5 large model to obtain better results. Compared to Yolov5 Median, the Yolov5 Large model is bigger, with more network layers and parameters. Therefore, we used AWS EC2 cloud computing platform for training.

1. Obtaining account and operational permissions

- Since we are using Emory Pro's AWS, necessary permissions such as launching EC2 instances, deleting EC2 instances, and connecting to S3 must be obtained.

2. Training Yolov5 on AWS EC2 cloud computing platform
   - We used a G4dn.xlarge instance on EC2. This instance includes an Nvidia T4 (16 GB) GPU. G4dn.xlarge is a cost-effective GPU instance suitable for small to medium-scale deep learning training. It supports TensorRT and CUDA to accelerate deep learning inference speed and supports FP16 and INT8 optimization for AI computation performance. Overall, this EC2 instance is suitable for our dataset and Yolov5 model training.

3. EC2 Training Details
   - After uploading all datasets and project code to the cloud computing service platform, training settings need to be configured. The number of epochs is set to 100, and an early stopping period is set to 30 epochs to prevent overfitting. The training dataset size is set to 3049, and the validation dataset size is set to 369.
   - Before training, it is necessary to log in to the AWS account and connect to Emory Pro's S3. The trained weights and logs are automatically uploaded to S3. Then, an automated shutdown is set for the EC2 instance after uploading to avoid excessive costs.
   - The screen function is enabled so that training can continue in the background even if the EC2 connection is lost.

4. Testing the trained weights. After training, the weights are tested. Once testing is complete, the weights are converted to a TFLite file so that they can be used in Android Studio.

### 2.5.2 Android Studio

Let's now have a step-by-step explanation of the implementation of the Flutter-based application designed to detect damages on vehicles and localize these damages on a labeling map. We created this very basic and simple application to test its functionality before we integrate it into the Emory Pro app. The application uses YOLOv5 models for object detection and segmentation, combined with image processing techniques to achieve accurate damage detection and localization. The implementation is centered around the *main.dart* file, which orchestrates the entire process. We will walk through the functionality of the code, explaining the key steps and their relationships, as well as the basic concepts behind the implementation.

**YOLOv5 Segmentation and Damage Detection**

The application utilizes two YOLOv5 models: one for segmentation and another for damage detection. The segmentation model is used to segment the car from the input image, even in cases where the car is heavily damaged. The damage detection model is specifically trained to identify damages on the car. These models are loaded into the Flutter application using the tflite_flutter package, enabling offline inference on mobile devices. The segmentation model generates masks that outline the car, while the damage detection model identifies and localizes damages on the car.

**Preprocessing**
Before running inference, the input image must be preprocessed to match the model's input requirements. T heYOLOv5 models expect input images of a specific size (640x640) and resizing ensures compatibility with the model. Also, normalization scales the pixel values to a range suitable for the model, improving inference accuracy. The function that handles this part resizes the image to 640x640 pixels while maintaining the aspect ratio and adds padding if necessary. This ensures that the image fits the input size expected by the YOLOv5 models.

Steps in the code:

- The input image is decoded using the image package.
- The image is resized to 640x640 pixels. If the aspect ratio is not 1:1, padding is added to fill the remaining space with a gray background (RGB: 114, 114, 114).
- The pixel values are normalized to the range [0, 1] by dividing by 255. This normalization is required for the model to process the image correctly.

**Running the Models**
Once the image is preprocessed, the application runs inference using the loaded models.

– Segmentation Inference:
  - The segmentation model processes the input image and outputs bounding boxes, class probabilities, and segmentation masks.
  - The output includes a list of detected objects and their corresponding masks.
– Damage Detection Inference:
  - The damage detection model processes the input image and outputs bounding boxes for detected damages.
  - The output includes the coordinates of the damages bounding boxes and their confidence scores.

**Post-Processing**
After running inference, the raw output from the segmentation model is post-processed to extract useful information.

Steps for segmentation model output (Figure 12):

- The segmentation masks are computed by multiplying the mask coefficients with the prototype masks.
- Non-Maximum Suppression (NMS) is applied to filter overlapping masks.
- The mask is a 2D array where each pixel is either 0 (background) or 1 (car).
- The masks are scaled to match the original image size.

Then the largest mask is chosen from the detected masks. This is a critical step because the input image may contain multiple cars or objects, but the system is designed to focus on the primary car of interest, which is typically the largest and most central object in the image. In the code the bounding boxes for all detected objects are extracted from the model output and the area of each of them is calculated using the simple rectangle area formula. Once the largest mask is identified, it is retained for further processing, while smaller masks are discarded.

*Figure 12 The segmentation result in the Android Studio*

In this case here, there is more than one car detected in the image but after post-processing only the mask with the biggest area will be selected. Also you can notice that in this image we have added padding to match the model required size without losing any details.

Steps for damage detection model output (Figure 13):

- The bounding boxes are extracted from the model output.
- Non-Maximum Suppression (NMS) is applied to filter overlapping boxes.
- The remaining boxes are scaled to match the original image size (cause the model gets the 640x640 image).

*Figure 13 The detection result in the Android Studio*

The raw output from the model includes many overlapping boxes. Post-processing ensures that only the most relevant detections are retained. Also, the center of the bounding boxes is calculated and its coordinates will be used as the damage location.

**Homography Transformation Matrix**

1. Concept and Calculation
The homography transformation matrix is used to map the detected damages from the input image to a labeling map. We calculate the matrix using four keypoints from the input image and the labeling map.

2. Keypoint selection
After segmenting the car using the YOLOv5 segmentation model, the application identifies the edges of the car mask that has been selected (the biggest area mask as discussed above) to determine the keypoints required for the homography transformation. The keypoints are calculated by scanning the mask for the extreme values in the horizontal and vertical directions. We first find the leftmost and rightmost points by checking the minimum and maximum x-coordinates where the mask value is 1. We calculate the middle line of these two

points and then the top point and bottom point are found by checking the minimum and maximum y-coordinates along the middle line of the mask.

3. Mathematical Foundation

The homography matrix *H*, a 3x3 transformation matrix that maps points from one plane (the input image) to another plane (the labeling map), is calculated using four corresponding keypoints from the input image and the labeling map.

Given four pairs of corresponding points *(xi,yi)* in the input image and *(xi',yi')* in the labeling map, the homography matrix HH satisfies the following equation for each pair of points:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} \tag{5}$$

Expanding this equation, we get:

$$x_i' = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \tag{6}$$

$$y_i' = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \tag{7}$$

To solve for the 8 unknowns in H (since h33 is typically set to 1), we rearrange the equations into a linear system A · h = b, where:

- A is a matrix that represents the coefficients of the linear system derived from the correspondence between the source points (input image) and destination points (labeling map). Each pair of corresponding points contributes two rows to the matrix A.
- h is a vector containing the elements of H.
- B is a vector representing the right-hand side of the linear system. It contains the coordinates of the destination points (xi′,yi′)(xi′ ,yi′ )

For each pair of points, we add two rows to the system:

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i' & -y_i y_i' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} \tag{8}$$

This system is solved using **Gaussian elimination** to find the values of vector h. Once h is computed, the homography matrix H is constructed as:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \tag{9}$$

4. Transforming Damage Coordinates (Figure 14)

Once the homography matrix is computed, the application uses it to transform the coordinates of the detected damages from the input image to the labeling map. This transformation is

achieved through matrix multiplication and normalization. The transformed coordinates are then used to mark the damages on the labeling map.

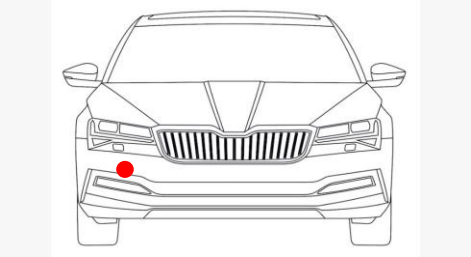

*Figure 14 mark the damages on the labeling map*

Summary

The implementation of our damage detection and localization system in Flutter is a multi-step process that involves loading models, preprocessing images, running inference, post-processing the output, and visualizing the results. Each step is carefully designed to ensure accurate and efficient damage detection and localization.

## 2.5.3 Results

1. Yolov5 Segmentation
    - Yolov5 median model. Among 2000 images containing damaged cars, the Yolov5 median model successfully segmented the damaged car's mask in 1877 images, leaving 133 images where it failed to segment the car's mask. Thus, most cars were successfully segmented, with only 133 failing to be segmented (Figure 16). Upon reviewing these unsegmented images, we found that the cars were so severely damaged that they had lost most of their defining features, causing the model to fail in segmentation. Examples are shown below. The accuracy is illustrated in the figure 15. The performance does not meet the requirements for real-world applications.

*Figure 15 The missing segmentation example*

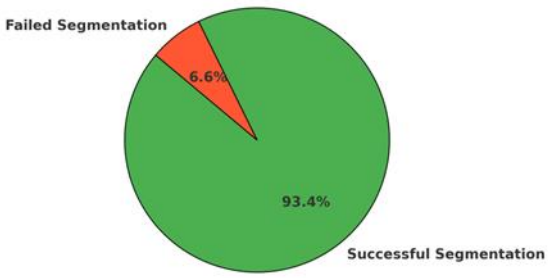


*Figure 16 The evaluation in the test set*

    - Yolov5 large model. Among 2000 images containing damaged cars, the Yolov5 large model successfully segmented the damaged car's mask in all 2000 images, with 0 images failing to be segmented. The Yolov5 large model can therefore segment cars in the vast majority of situations, making Yolov5 large the final model of choice.

2. Yolov5 Detection
    - First, we conducted training locally on a PC using an Nvidia 3060ti GPU (8G). We chose the Yolov5 median model, with 3049 images in the training set, 369 images in the validation set, and 93 images in the test set. Due to limited computational power, we only trained for 40 epochs, which took about 400 minutes.
    - The training results are shown in the figure 17. After each epoch, the model was evaluated on the validation set in terms of mAP50 and mAP95. We found that the model achieved an mAP50 of around 0.6 (Figure 17), indicating a detection capability suitable for real-world applications. The mAP95 was around 0.28. Because our dataset was manually labeled by team members with varying understandings of car damage, the boundaries for the same type of damage can differ. As a result, performance measured by the stricter mAP95 metric is not as good. Therefore, mAP50 more accurately reflects the model's detection performance in real-world scenarios.
    - We used the Yolov5 median weights that performed best at mAP50 to detect car damage in the test set. Summarizing the results: only 13 damaged images were missed, giving an accuracy of 86%. There were just 3 cases where undamaged images were mistakenly labeled as damaged, resulting in a false positive rate of only 3%.

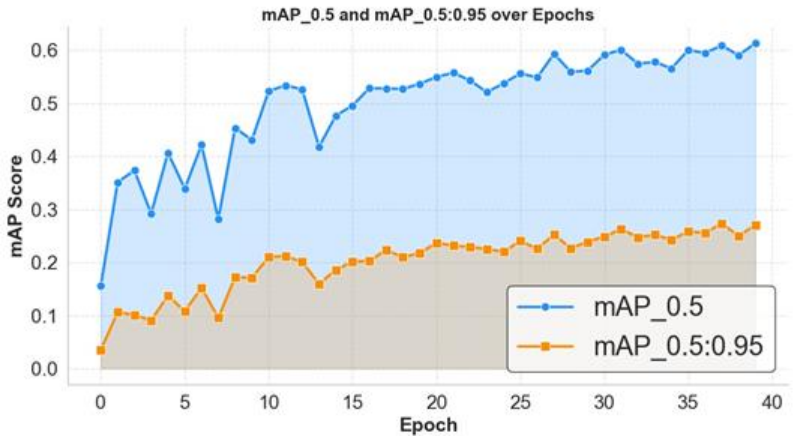


*Figure 17 The result of the YOLOv5m training*

3. Yolov5 Detection in AWS
    - After obtaining promising results using the Yolov5 median model, we attempted to train a larger model, Yolov5 large, on the AWS cloud computing platform. On an EC2 instance (G4dn.xlarge), which includes an Nvidia T4 (16 GB) GPU, we used 3049 images for the training set, 369 for the validation set, and 93 for the test set. Due to limited computing resources, we trained for only 60 epochs, taking about 130 minutes.
    - The training results are shown in the figure 18. The model achieved an mAP50 of around 0.7, which is 0.1 higher than the Yolov5 median model. Yolov5 large also attained an mAP95 of approximately 0.3.
    - Using the Yolov5 large weights that performed best at mAP50, we conducted detection on the test set. Summarizing the results: only 4 damaged images were missed, giving an accuracy of 96%. Such a result is sufficient to address the vast majority of real-world scenarios, effectively detecting most car damages.

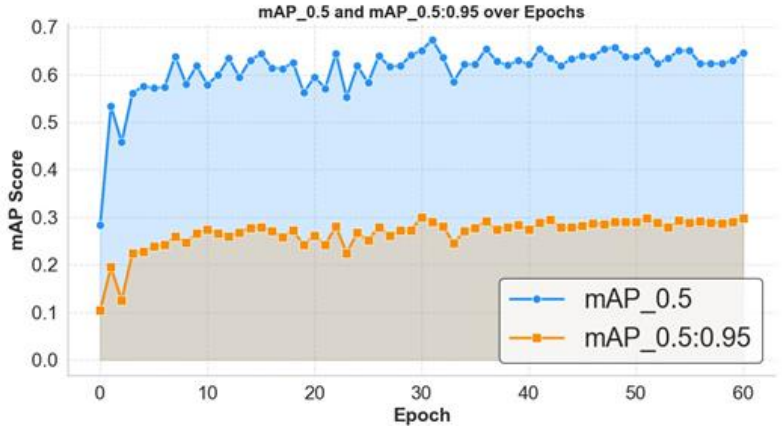


*Figure 18 The result of the YOLOv5l training*

4. Final results

Here below you can see the implementation of the full project in the Emory Pro app (Figure 19). As you can see the user needs to enable the machine learning models first before using them. The feature was added in this way in order to not interfere with the current regular process of the reports creatin and editing in the app.

Then after creating the report you can go under ***Markings*** section and by using the arrows in the right bottom it is possible to choose the labeling map that we want to use based on the view of the image of the car we want to inspect. In this case we select the front view of the car as the labeling map. To upload the image of the car to be detected is possible by clicking the top right camera icon.

The final photo shows the labeling map with the marking in the front left light where the model has detected the damage.

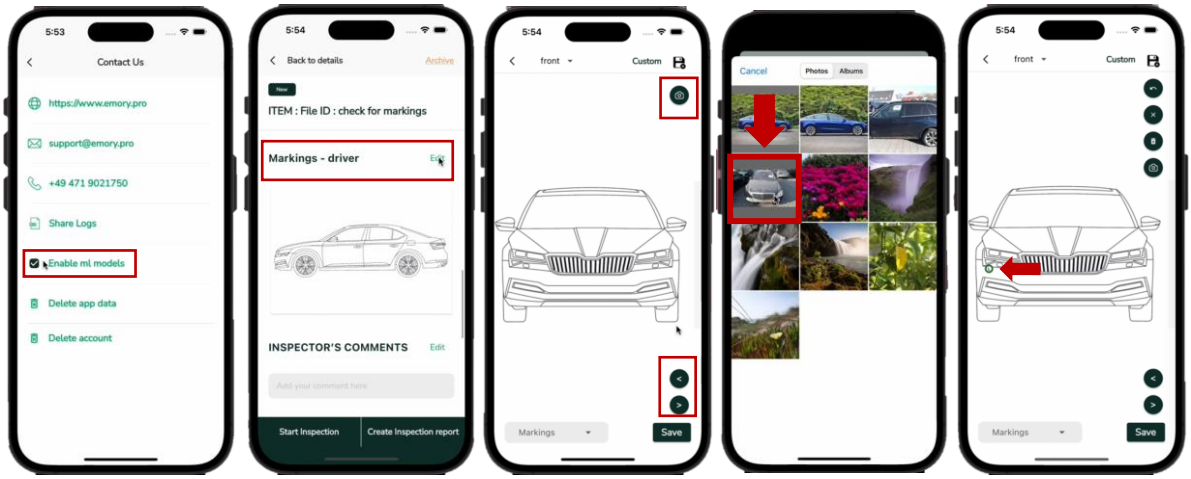


*Figure 19 The implementation of the task in the Emory Pro app*

# 3 AWS

## 3.1 Introduction

Amazon Web Services (AWS) is a leading cloud computing platform that provides a wide range of services for storage, computing, and application development. It offers scalable and cost-effective solutions that help businesses and developers build and deploy applications with high reliability and security. In this section, we will discuss AWS services relevant to our

project, particularly S3 and AWS Lambda, and how they are used for file archival and restoration.

### 3.1.1 What is AWS?

AWS is a cloud service platform that provides infrastructure and software services such as computing power, storage solutions, databases, networking, analytics, and artificial intelligence. It allows users to host applications and store data without the need for on-premises hardware. AWS offers flexible pricing models, making it suitable for businesses of all sizes. It ensures high availability, scalability, and security, making it one of the most widely used cloud platforms worldwide.

### 3.1.2 S3

Amazon Simple Storage Service (S3) is an object storage service that provides high scalability, data availability, security, and performance. It allows users to store and retrieve any amount of data from anywhere on the web. S3 is commonly used for backup storage, data archiving, website hosting, and big data analytics. The service supports different storage classes, allowing users to optimize costs based on access frequency and retention requirements.

### 3.1.3 Standard Vs. Glacier

Amazon S3 offers multiple storage classes to balance performance and cost. The **Standard** storage class is designed for frequently accessed data, providing low-latency and high-throughput performance. On the other hand, **Glacier** is optimized for long-term archival storage, significantly reducing costs but requiring longer retrieval times. Glacier is ideal for data that is rarely accessed but must be retained for compliance or backup purposes. The selection between Standard and Glacier depends on the use case and cost considerations.

### 3.1.4 AWS Lambda Functions

AWS Lambda is a serverless computing service that allows users to run code without provisioning or managing servers. It automatically scales based on demand and charges only for the execution time used. Lambda functions are triggered by events such as file uploads to S3, API requests, or scheduled tasks. In our project, AWS Lambda is used to automate the archival and restoration of files, reducing manual intervention and optimizing costs.

## 3.2 Goal

### 3.2.1 Cost Optimization of the Overall Project

One of the primary goals of using AWS services is to optimize storage costs. By leveraging Amazon S3 and Glacier, we reduce expenses associated with long-term storage. Frequently accessed files remain in S3 Standard, while older or less frequently used files are archived in Glacier. This approach minimizes costs while ensuring data availability when needed.

### 3.2.2 Creating a Lambda Function for Archival and Restoration

Another goal of the project is to develop AWS Lambda functions for automating file archival and restoration processes. The functions ensure that files are systematically moved from S3

Standard to Glacier for cost savings and retrieved back when needed. The automation eliminates manual file management, reducing operational complexity and improving efficiency.

## 3.3 Implementation

### 3.3.1 Archival Process

The first AWS Lambda function is designed to manage the archival of files stored in an S3 bucket. The main steps involved are:

- **Grouping and Organizing Files:** The function retrieves file keys from the S3 bucket (emory-archive-trial-1). Files are grouped based on organization and item type to maintain a structured archive.
- **Creating ZIP Files:** The grouped files are downloaded and compressed into ZIP archives, preserving the folder structure. Temporary storage (/tmp) is used during this process.
- **Uploading to Glacier:** The ZIP files are uploaded to another S3 bucket (emory-archive-glacier-trial-1) with the StorageClass set to GLACIER for cost-effective long-term storage.
- **Error Handling:** The function includes error handling mechanisms to log and manage failures during file processing and uploads.

### 3.3.2 Restoration Process

The second AWS Lambda function facilitates the retrieval of archived files from Glacier. The key steps include:

- **Checking Restoration Status:** The function checks the metadata of files in the Glacier storage bucket to determine if they are archived or currently being restored.
- **Initiating Restore Requests:** If a file is still in Glacier, a restore request is initiated with a one-day retrieval period.
- **Downloading Restored Files:** Once the restoration is complete, the files are downloaded from the Glacier-backed S3 bucket to a temporary location (/tmp).
- **Extracting and Uploading Files:** If the restored file is a ZIP archive, it is extracted and uploaded to the destination S3 bucket (emory-archive-trial-1) under a structured path (restored/).
- **Cleanup:** Temporary files are deleted after successful processing to optimize Lambda execution and storage space.
- **Error Handling:** The function logs any issues encountered during download and extraction, ensuring smooth execution.

# 3.4 Results

## Current Storage Costs:

Currently, the storage system utilizes Amazon S3 Standard for all image storage, leading to a monthly bill of **€40**. The breakdown of storage requirements is as follows:

- **Total items/containers**: **25,000**
- **Items** contain **30 images each → 750,000 images**
- **Containers** contain **150 images each → 3,750,000 images**
- **Total images stored**: **4,500,000**
- **Each image is ~1MB**, leading to a total storage requirement of **4.5 TB**

## Data Growth and Processing:

- The system processes **900 cars/containers per month**.
- Average images per car/container: **90 images**.
- New images added per month: **81,000 images (~81 GB).**

## Cost Reduction via Glacier Deep Storage:

To optimize costs, all data older than **6 months** is migrated to **S3 Glacier Deep Archive**, keeping only recent files in S3 Standard.

- **New active data (last 6 months)**: **486 GB** remains in S3 Standard.
- **Archived data (older than 6 months)**: **4.01 TB** moved to Glacier Deep Archive.

## New Storage Cost Breakdown:

| Storage Type | Data Stored | Cost per GB | Total Cost |
|---|---|---|---|
| S3 Standard | 486 GB | €0.023 | €11.18 |
| Glacier Deep Archive | 4.01 TB | €0.00099 | €3.97 |
| Retrieval Cost | 2.7 GB | €0.02 | €0.054 |
| **Total New Cost** | - | - | **€15.20** |

## Cost Savings Analysis:

- **Previous Monthly Cost**: **€40**
- **New Monthly Cost**: **€15.20**
- **Total Monthly Savings**: **€24.80 (~62% reduction)**

## Projected Costs Over 3 Years:

| Time Frame | Total Cost (S3 Standard) | Total Cost (Optimized) | Total Savings |
|---|---|---|---|
| 1 Year | €480 | €182.40 | €297.60 |
| 3 Years | €1,440 | €547.20 | €892.80 |

## Conclusion:

By implementing an automated archival strategy, the project achieves a 62% reduction in storage costs, significantly optimizing expenses while ensuring accessibility for frequently used data. AWS Lambda functions further streamline the process, ensuring seamless movement of files between S3 Standard and Glacier. Over three years, this strategy results in a total savings of €892.80, making long-term data management highly cost-effective.

# 4 OCR

## 4.1 Goal

Our goal is for the application to automatically recognize the VIN number when a user uploads a photo containing a VIN number, and then populate the VIN number into the report (Figure 20). This replaces manual entry of the VIN number into the report, thereby reducing labor and improving efficiency.

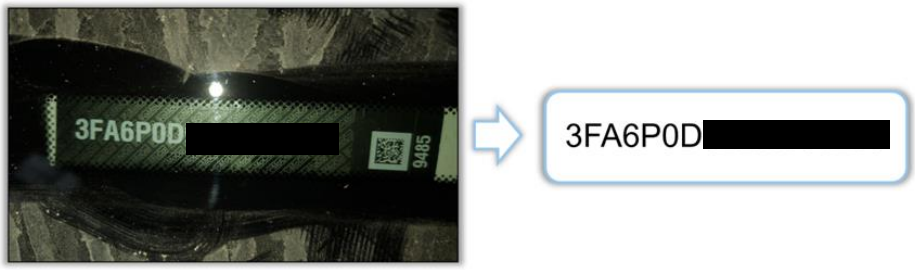


*Figure 20 The OCR task*

## 4.2 Dataset

1. We used a dataset from the Emory Pro database (Figure 21). The images in the dataset contain VIN numbers from cars, as illustrated below. The dataset totals 3,000 images, covering 1,000 reports. In other words, it includes 1,000 car VIN images and their corresponding VIN labels.
2. Many images of the VIN number contain significant interference elements, such as dust, rain, glare, and so on. These factors can seriously affect the OCR (Optical Character Recognition) performance. By using cropping techniques, we eliminated most of this interference, thereby improving the OCR results.

*Figure 21 The Emory Vin number data*

## 4.3Implementation

1. Downloading images containing VIN numbers from the Emory Pro database

- Each car report contains a large number of images with different content. We observed that images containing VIN numbers are usually among the first three images in the report. Therefore, we downloaded the first three images from 3,000 reports in the Emory Pro database, along with the VIN numbers from the reports for later labeling.

2. Training a ResNet classification model using data augmentation

- The ResNet classification model is used to distinguish between images that contain VIN numbers and those that do not. Then we can obtain a dataset that only contains VIN numbers.
- To quickly obtain labeled data, we employed data augmentation techniques. Initially, 100 images were manually labeled and used to train a ResNet classification model.
- The trained model was then used to classify the remaining images. We retained images with a VIN number probability greater than 0.9 and manually filtered out those without VIN numbers, forming a dataset that only contains VIN numbers.
- The new dataset was used for further training of the ResNet classification model. The same process was repeated, keeping only images with a VIN number probability greater than 0.9. This training cycle was repeated three times, ultimately resulting in a ResNet model capable of accurately classifying images containing VIN numbers.

3. Training a Yolov5 detection model to crop VIN number regions (Figure 22)

- Since OCR recognition is often affected by background noise and image size, we need to preprocess images by removing distractions. As VIN numbers occupy a small proportion of the image, many irrelevant elements exist. Thus, we use Yolov5 to detect VIN number regions and crop them to obtain smaller images containing only VIN numbers.
- We employed data augmentation to train the Yolov5 detection model. Initially, 100 images were manually labeled to train a Yolov5 model capable of detecting VIN numbers.
- The trained Yolov5 model was then used to detect VIN numbers in unlabeled images. Since the model was trained on only 100 images, it initially performed poorly and detected both VIN and non-VIN images.
- At this stage, we used the previously trained ResNet classification model to filter the Yolov5 detection results, keeping only images containing VIN numbers. The bounding box information was then used to automatically label the results, expanding the training dataset.
- The updated dataset was used to retrain Yolov5, repeating the process for a total of three training cycles. This resulted in a Yolov5 model that could accurately detect VIN numbers.
- Finally, detected images were cropped based on bounding box information to remove background noise, producing clean VIN number images.


*Figure 22 Training a Yolov5 detection model to crop VIN number regions*

4. OCR Recognition Using Google ML Kit in Android Studio (Figure 23)

- Google ML Kit was integrated into Android Studio to implement OCR recognition functionality.
- The OCR accuracy significantly improved when using the cropped images from Yolov5 detection compared to the unprocessed images.


*Figure 23 The whole pipeline in Android Studio*

# 5 Conclusion

This project successfully developed an AI-driven vehicle inspection system that utilizes computer vision techniques to automate damage detection and vehicle identification number (VIN) recognition. By integrating YOLOv5 (for damage detection) and OCR-based VIN extraction into the Emory Pro platform, we significantly improved the efficiency and accuracy of vehicle inspection.

Key results of the project include:

1. Automated damage detection:
- The YOLOv5-based damage detection model achieves 96% accuracy in identifying dents and broken parts on vehicles.
- By applying the uni-stochastic matrix transformation, we successfully mapped the detected damages onto a standardized labeled map, enabling accurate visualization of damage locations.
- We optimized the entire process using TFLite for execution on mobile devices, ensuring real-time inference in the Emory Pro app.

2. Recognizing VINs using OCR:
- A ResNet-based classification model is used to filter images containing VINs to improve processing efficiency.
- Prior to applying OCR using Google ML Kit, the YOLOv5 detection model accurately localized text regions, improving recognition accuracy under difficult conditions such as glare and distortion.
- The final system enables instant, highly accurate VIN extraction on mobile devices, reducing manual input errors and increasing workflow automation.

3. Cloud-based storage and cost optimization:
- The project leverages AWS S3 and Glacier for efficient data storage and retrieval, balancing cost and accessibility.
- AWS Lambda capabilities automate file archiving and recovery, reducing operational overhead and ensuring smooth scalability.

Impact and future work

The integration of AI-driven damage detection with OCR-based VIN recognition significantly reduces labor in vehicle inspection, increasing speed and accuracy. Real-time mobile implementation ensures availability in real-world scenarios without the need to rely on cloud processing, resulting in improved data privacy and system responsiveness.

Highlights of future improvements include:

- Expanding the damage detection dataset to include more vehicle types and damage scenarios to further improve model robustness.
- Enhancing isomorphic transformations to account for more diverse vehicle perspectives.
- Optimization of AWS-based data pipelines for smarter data management, such as automated damage trend analysis and predictive maintenance recommendations.
- Extension of the OCR module to support additional document types relevant to vehicle inspection.

By implementing these enhancements, the Emory Pro platform can evolve into a comprehensive AI inspection tool that sets a new industry standard for automated vehicle evaluation.

**Bibliography**


[1]. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI:10.1109/CVPR.2016.91

[2]. Jocher, G., et al. (2022). *YOLOv5 by Ultralytics.* Available at: https://github.com/ultralytics/yolov5

[3]. Smith, R. (2007). *An Overview of the Tesseract OCR Engine.* Ninth International Conference on Document Analysis and Recognition (ICDAR). DOI:10.1109/ICDAR.2007.4376991

[4]. Google Developers. (2023). *ML Kit for Text Recognition.* Available at: https://developers.google.com/ml-kit/vision/text-recognition

[5]. Baek, J., Kim, G., Lee, J., Park, S., Han, D., Yun, S., Oh, S., & Lee, H. (2019). *What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis.* Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). DOI:10.1109/ICCV.2019.00010

[6]. He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI:10.1109/CVPR.2016.90

[7]. Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition.* arXiv preprint arXiv:1409.1556. DOI:10.48550/arXiv.1409.1556

[8]. Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization.* arXiv preprint arXiv:1412.6980. DOI:10.48550/arXiv.1412.6980

[9]. Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization.* arXiv preprint arXiv:1412.6980. DOI:10.48550/arXiv.1412.6980

[10]. Hartley, R., & Zisserman, A. (2003). *Multiple View Geometry in Computer Vision.* Cambridge University Press. DOI:10.1017/CBO9780511811685

[11]. OpenCV Documentation. *FindHomography Function.* Available at: https://docs.opencv.org/

[12]. Amazon Web Services. (2023). *Amazon S3: Scalable Storage in the Cloud.* Available at: https://aws.amazon.com/s3/

[13]. Amazon Web Services. (2023). *AWS Lambda: Serverless Compute.* Available at: https://aws.amazon.com/lambda/

[14]. Dean, J., & Ghemawat, S. (2008). *MapReduce: Simplified Data Processing on Large Clusters.* Communications of the ACM. DOI:10.1145/1327452.1327492

[15]. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... & He, K. (2017). *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.* arXiv preprint arXiv:1706.02677. DOI:10.48550/arXiv.1706.02677