Automated car exterior damage assessment

PG Level Advanced Certification Programme in Computional Data Science

Team - 6

Cohort – 2

Brief Problem Statement:

In this project, we are proposing a computer vision solution to assess the damage to cars via its images captured post-accident. Here, we aim to solve the following problems.

1) **Where?**
    - Detect the area of the damage in the image (if there is any)
2) **What?**
    - Predict the class of the damage
3) **How severe?**
    - Estimate the severity of the damage

Background Information:

Domain Information:

Validation of car damage claims forms a large part of the operations in a car insurance company. Automated car damage detection systems can be very useful for car insurance companies. The standardization of car damage classification can plug leaks in claims and underwriting [1]. The process of visual inspection and validation can increase delays and be more erroneous than an automated one. In addition to this, it can bring great cost benefits to the insurance company in claims processing. An added benefit can be the ease of operation for the end-user. If applied in certain ways, this could lead to scenarios where the end-user can upload photos of the damage for the automated claim processing. However, care would have to be taken to get photos of good quality under correct illumination, etc. [2].

Problem description and analysis:

We want to automate the process of damage detection and classification of the damage from an image of the vehicle. To achieve this, we aim to do the following

1. Building a repository of car damage images via scraping and existing datasets of car damage.
2. Develop processes to reliably label and annotate damaged regions and images by reducing the possibility of human error.
3. The pre-processing of the input images to minimize the shadow, lighting, and reflection-related effects.
4. We will use deep learning CNN frameworks to achieve instance segmentation and classification. We aim to experiment and test using various models and frameworks to compare the final accuracies of our data.
5. Analysis of results on the test set and understanding areas where the model did good and where it lacked.

Challenges:

Some of the challenges of building such a model are -

1) As is with any Computer Vision related problem statement it isn't "Garbage In, Awesome Out". A lot rides on the quality of input images. In professional apps, we would have control over the image acquisition process that lets us guide the user to capture better quality images w.r.t image resolution, lighting of the scene i.e. the exposure, keeping the car to be inspected as the dominant object in the scene, and limiting other distractions in the scene. Since we cannot do this at this point, we must rely on images captured via web scraping and other publicly available datasets.

2) Irrespective of whether we had control over acquisition or not, some of the problems remain. For ex., shadow, lighting, and reflection-related effects.
3) Human error in labeling and annotation of damaged areas while generation of the train, validate, or test sets can severely affect our final model.

Possible Applications:

Although the model we have trained is specifically for vehicular damage, this could similarly be implemented for various other damage detection exercises, for example for climate change damages each year using satellite images. Or for other uses such as building damage detection [3].

Detailed dataset description and the dataset source

Data acquisition and cleaning are critical for the performance of our model. Our base reference dataset that we will use for this model is the COCO Car Damage Dataset. We intend to also add to our input data images obtained from the web. Some of our key strategies are listed below.

1. Our primary intention is to collect data that is relatively well illuminated, close-up pictures of the damage on the car body, with minimal perspective distortion.
2. We will use web scraper scripts using beautiful soup API and Selenium to scrape images from the web. In addition, we will use the COCO Car damage dataset. Sample keywords for our images will be –
    a. ‘Close Up Car Damage’, ‘Car Damage’, ‘Vehicle Damage’ and other similar phrases
3. The format for the data used in this project will be
    a. Raw images – jpg (collected in many formats but converted into jpg)
    b. Annotation files – JSON
4. The data collected will then have to be annotated, and for this purpose, we will use tools such as the VGG Annotation tool (VIA), Plainsight.ai, TagX, etc

Current Benchmark:

Following benchmark is from [1] and [4], the following results should be taken with pinch of salt as these papers did not publish data source on which these results were measured.

Table 4. Results on YOLOv3 (416x416) Map score: 0.7423

Threshold	Precision	Recall	F1-score	Avg. IOU
0.00	0.59	0.81	0.68	43.22%
0.25	0.82	0.73	0.77	61.25%
0.40	0.84	0.71	0.77	62.99%
0.50	0.86	0.70	0.77	64.18% .18%
0.70	0.88	0.69	0.77	65.70%

Table 5. Result on YOLOv3 (608x608) Map score 0.7778

Threshold	Precision	Recall	F1-score	Avg. IOU
0.40	0.70	0.80	0.74	50.21%

AP scores: **BD**: 77.16% **CR:** 59.21% **DD:** 88.21% **GS:** 71.07% **LB:** 64.56% **SM:** 85.19%

Table 6. Results on damage detection Map score: 0.663

Threshold	Precision	Recall	F1-score	Avg. IOU
0.00	0.65	0.80	0.72	45.76%
0.25	0.81	0.78	0.80	57.20%
0.40	0.82	0.77	0.79	57.66%
0.50	0.82	0.77	0.79	57.66%
0.70	0.84	0.75	0.79	59.53%

Proposed Method

Pipeline Overview

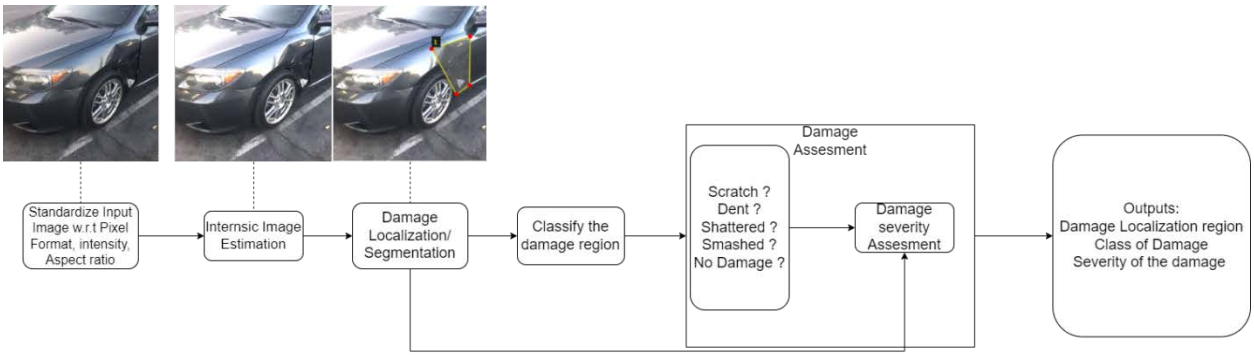


Fig 1: Proposed Pipeline

Pipeline stages

Stage1:

As the input image data set is a union of images found across several data sets and our web scraping, images might be varied in terms of pixel format, aspect ratio, etc. In this stage, we aim to convert all images to 8-bit per channel RGB images and downsample them to a pre-defined aspect ratio that is amenable w.r.t the damage localization task.

Packages and Tools:

- OpenCV – Python

Algorithms:

- OpenCV format converters
- Bicubic interpolation

Stage 2:

As the images are most likely captured by a mobile phone, it is subject to a lot of external factors such as the illumination of the scene. Poor illumination of the scene leads to an under-exposed image and some of the images taken under a bright light source lead to an over-exposed image. Apart from this, the position of the light source w.r.t the mobile phone camera can lead to certain portions of the image being over or underexposed. Additionally, there is another problem of reflection as the metallic surface of cars capture reflections of objects around them. We think object detection should benefit from factoring out shadow, lighting, and reflection effects, this is something we are willing to experiment and check as we are not

completely sure of the magnitude of impact it might have when trained with images corrected w.r.t illumination factors vs when trained without correcting. So, this is an experimental/optional module.

To solve the above-mentioned challenges we aim to estimate an intrinsic image. Given an image I, it can be decomposed into the following components,

$$I = R \times L$$

Reflectance component R and illumination component L . These components are used in estimating the intrinsic image which is a representation of the original image minus the shading, lighting, and reflection-related effects.

Packages and Tools:

- OpenCV – Python

Algorithms:

- The paper by Yu Li, Michael Brown et al, published in CVPR 14, is our reference for the implementation.

Stage 3:

The objective is to localize the damaged regions in the input. Each input may have single or multiple damaged regions. We are willing to experiment and compare results from several state-of-the-art deep learning architectures for instance segmentation (in this case damage segmentation).

Packages and Tools:

- OpenCV – Python
- Tensorflow and Keras
- Mask_RCNN
- UNet

Algorithms:

- YOLOv3
- Mask_RCNN
- UNet

Test metrics:

- Average Precision (AP):

  AP is popularly used as a test metric to evaluate the performance of an object location task. In practice, AP is the precision averaged across all recall values between 0 and 1. It is clear that to calculate AP, we need Precision, and Recall and for computing them we need to compute True Positive(TP), False Positive(FP), and False Negative (FN). As it is a localization problem, these values are computed using the IoU score. Let's see these concepts in detail.

  Intersection over Union (IoU) –

  IoU is a measure based on Jaccard Index that evaluates the overlap between two bounding boxes. It requires a ground truth bounding box B_{gt} and a predicted bounding box B_{p}. By applying the IoU we can tell if detection is valid (True Positive) or not (False Positive).

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

  TP, FP, and FN can be computed as

  - True Positive (TP): A correct detection. Detection with IoU $\geq$ *threshold*
  - False Positive (FP): A wrong detection. Detection with IoU $<$ *threshold*
  - False Negative (FN): A ground truth not detected

  threshold: to be determined via experimentation

  Based on this Precision and Recall of the local region can be computed as

  Precision is the ability of a model to identify only the relevant objects. It is computed as

$$\mathrm{Precision} = \frac{TP}{TP + FP} = \frac{TP}{all\ detections}$$

  Recall is the ability of a model to find all the relevant cases (all ground truth bounding boxes). It is computed as

$$\mathrm{Recall} = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths}$$

Stage 4:

The localized region from the previous stage is cropped out and fed as input to this model whose objective is to do a multi-class classification. The classes are shown in Fig 1. It might be observed that the instance segmentation framework mentioned in stage 3 can do both damage localization and damage classification. Here, we are willing to experiment and check if given a cropped region of damage that is free of any other distraction in the scene of the input image and with transfer learning from state-the-art architectures can yield better classification accuracy. Another added advantage of this approach is, it decouples damage localization from damage classification. This would help us in

- Expanding to other classes of damages without impacting the damage localization model.
- Since these damage categories are the same for all vehicles, this model may need minor to no update in training when it is to be expanded to all kinds of vehicles as it only takes the damage region as input. Only the damage localization model needs to be expanded.

Based on the class of the damage obtained via this model and its extent of it obtained from damage localization we can infer the damage severity i.e. bucketize as Mild, Moderate, and Severe categories.

Packages and Tools:

- Tensorflow and Keras

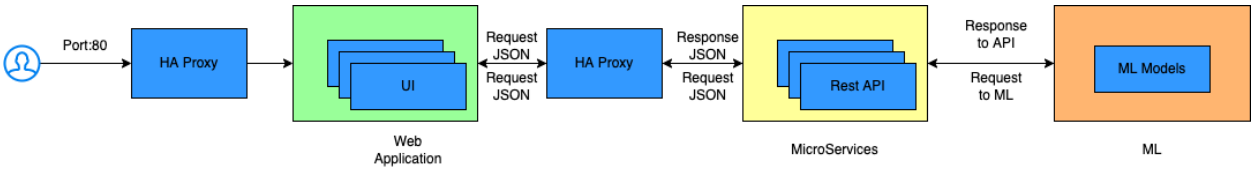
Algorithms:

- Transfer learning via – the only backbone
    - VGG19
    - ResNet50
    - InceptionV3
    - Mobile Nets
- Training Dense MLP with softmax with specified class labels attached to the pre-trained backbone.
- Fine-tuning by opening up all layers for final training.

Test Metrics:

- Class-wise Precision, Recall, and F1-score
- Weighted and macro F1 score.

Deployment plan (tentative)



Approaches

We are willing to experiment with the following combination approaches to see which leads to higher accuracy.

- With Intrinsic Image Estimation + Various Instance segmentation frameworks + Various architectures for classification
- Without Intrinsic Image Estimation + Various Instance segmentation frameworks + Various architectures for classification
- With Intrinsic Image Estimation + Instance segmentation and object localization in the single model via wide networks etc.
- Without Intrinsic Image Estimation + Instance segmentation and object localization in the single model via wide networks etc.

Stages with defined deliverables

Phase	Deliverables
Phase 1: • Data processing and Labelling • Data pre-processing	• Web scraping the car damage images. • Set up guidelines for labeling damaged regions. • Label polygon regions for damages and class label per polygon via the VGG Image Annotator tool. • Implement an intrinsic image estimation method as highlighted in stage 2 of the proposed method.
Phase 2: • Training the damage localization model • Fine-tuning the damage localization model	• Train and explore models mentioned in stage 3 of the pipeline. • Evaluate and fine-tune model w.r.t data augmentation, • Hyperparam tuning like model config in YOLO, LR scheduling, etc.
Phase 3: • Training the damage classification model • Fine-tuning the damage classification model	• Train and explore models mentioned in stage 3 of the pipeline. • Evaluate and fine-tune model w.r.t data augmentation, • Hyperparam tuning like model config in YOLO, LR scheduling, etc.
Phase 4:	• Select the best approach

• Compare and contrast approaches • Final report	• Final report w.r.t results and explanation of results w.r.t use-cases where model performed better and where it lacks.

Expected Outcome

- With reasonable accuracy localize damaged regions in the input image.
- With reasonable accuracy predict the class of damage and its severity.

Preliminary Exploratory Data Analysis

Data acquisition & annotation:

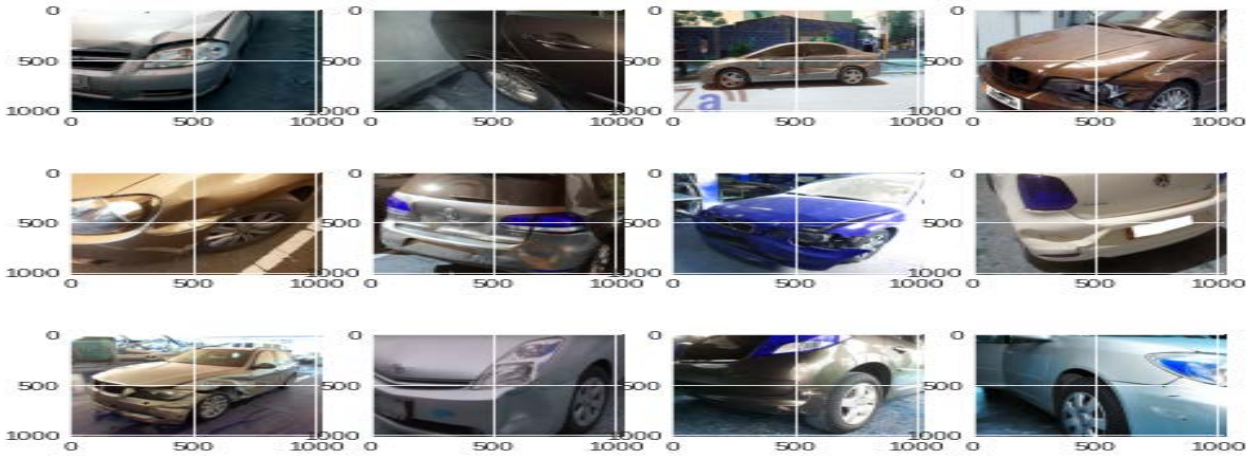
Apart from the COCO dataset, 500 images collected so far from web scraping.

Data Acquisition, Data Cleaning, and Filtering (Pre-processing)::

Data acquisition is the first step to EDA and pre-processing image data sets is a crucial step in speeding up and obtaining better training results for models. To facilitate a robust training process, a python script has been developed to Scrape Data from the web for damaged car images under different conditions. The process is still ongoing. Post collection, images are being manually checked to remove unwanted images and prepare the dataset for annotation.

Approach:: Web scraping is done through a python script, Tools like OpenCV / PIL have been used for image pre-processing.

Displaying Few Samples of web scraped images::



1. **Data Annotation::**

For the automated car damage assessment model to fully realize its potential, machine learning algorithms need to be fed with accurate annotated data. These systems will be required to identify and analyze objects and scenarios in endlessly complex real-world environments. Annotations are used to provide supplemental information about a program and help to associate metadata (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.

Professional annotation services, such as the VGG Annotation tool (VIA), Plainsight.ai, TagX, etc make the job simpler.

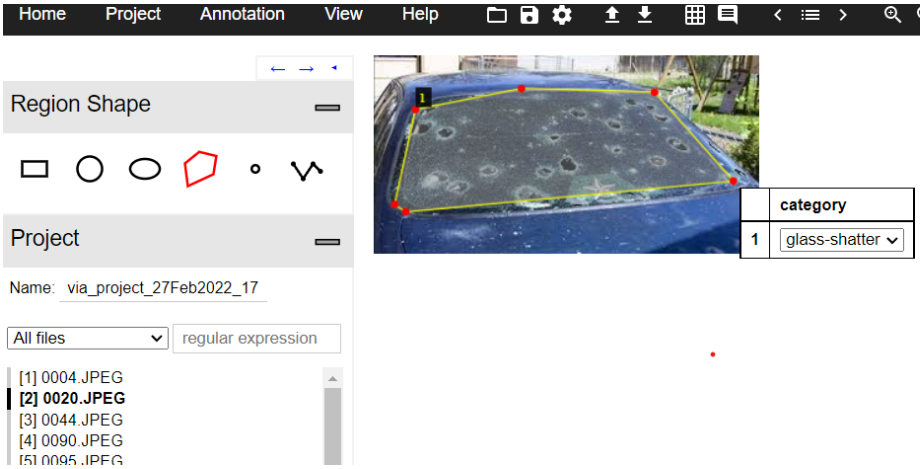
Approach:: Tools for Annotation: VIA used to annotate Web scraped data to facilitate ***Instance Segmentation***

Annotation categories took for the bounding box:: 4 categories **– scratch , dent, glass-shatter, smash**

Annotations on few web scraped images and a snapshot of the dataframe with the annotations ::

filename	region_area	region_shape	region_count	region_category	reg_attrib_x	reg_attrib_y
10.jpg	66088	polygon	1	dent	[250,440,334,214,193,216]	[205,219,509,493,389,293]
13.jpg	59970	polygon	1	dent	[396,539,668,574,473,380,354,358]	[182,161,483,807,765,615,421,348]
16.jpg	63318	polygon	1	scratch	[507,647,647,579,527,498]	[485,382,498,557,536,526]
25.jpg	110591	polygon	2	smash	[212,524,555,444,202]	[712,601,608,894,779]
25.jpg	110591	polygon	2	smash	[527,735,747,680,586,545,517,503,495]	[466,384,580,635,533,560,558,543,538]

Snapshot of sample annotated images through VIA tool::



2. **Image sizes and aspect ratios::**

In general, most datasets fall into one of 3 categories -

- **Uniformly distributed** - Images have the same dimensions – So, resizing will mainly depend on objects area, size, and aspect ratios.
- **Slightly bimodal distribution** - Images are in the aspect ratio range of (0.7-1.5), similar to the COCO dataset. Resize -> Pad approach. Padding will be necessary but to a degree such that it does not blow the size of the dataset.
- **Dataset with a lot of extreme values** (very wide images mixed with very narrow ones) – Excessive padding to be avoided. Sampling batches of images based on the aspect ratio. This can introduce a bias to the sampling process – so should not be strong enough.

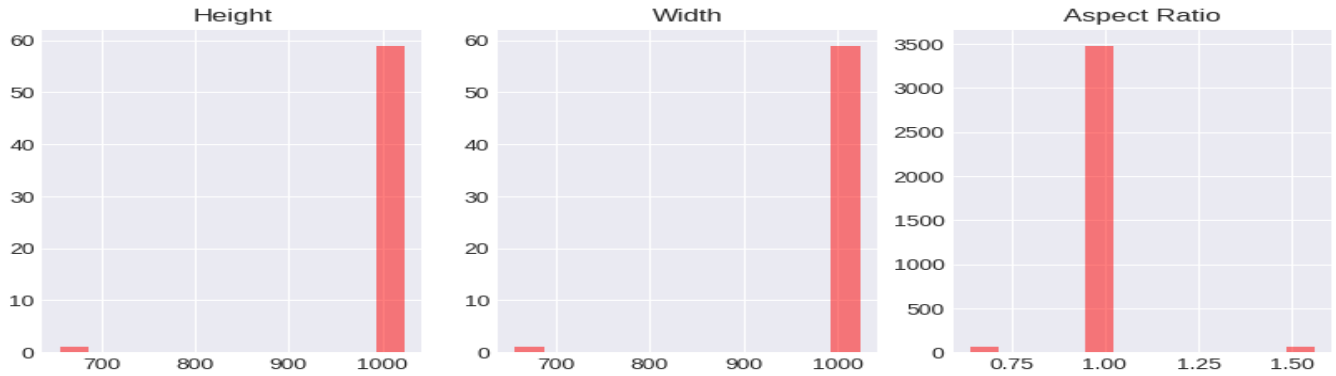
Findings:: For the COCO train set:: Image properties for samples studied :

Total files taken into consideration = 60

Max Height, Min Height, Avg Height :: 1024, 656, 1017.86

Max Width, Min Width, Avg Width :: 1024, 656, 1017.86

Visualization ::



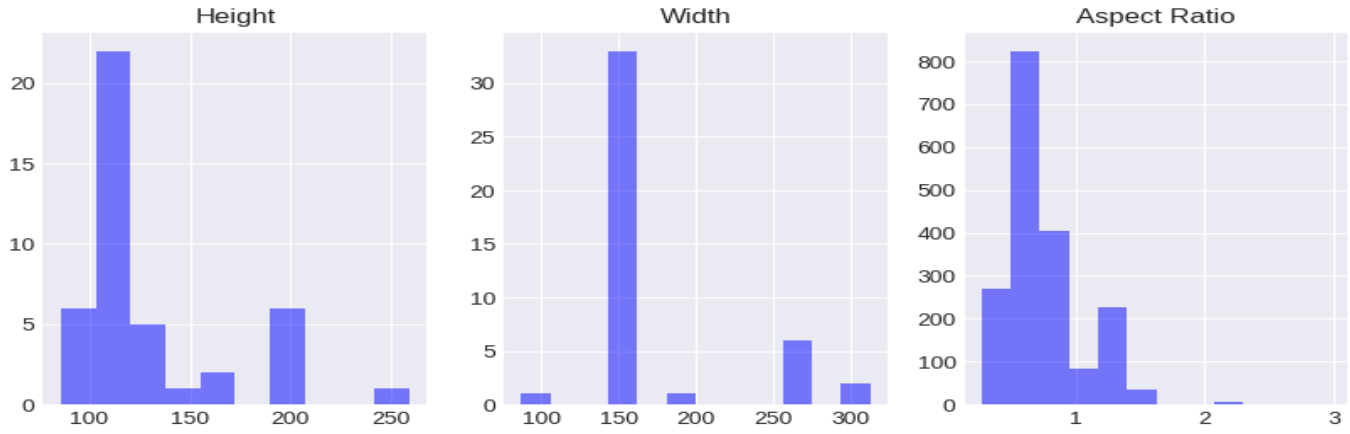
For the Web Scraped dataset:: Image properties for samples studied :

Total files taken into consideration = 43

Max Height, Min Height, Avg Height:: 259, 86, 128.39

Max Width, Min Width, Avg Width :: 313, 87, 181.30

Visualization::



2. Label (objects) sizes and dimensions::

The most important things to consider when it comes to box or mask dimensions are - ***Aspect ratios and Size (Area).*** Because -

- If the dataset contains only really big objects – it might be possible to simplify the model a lot

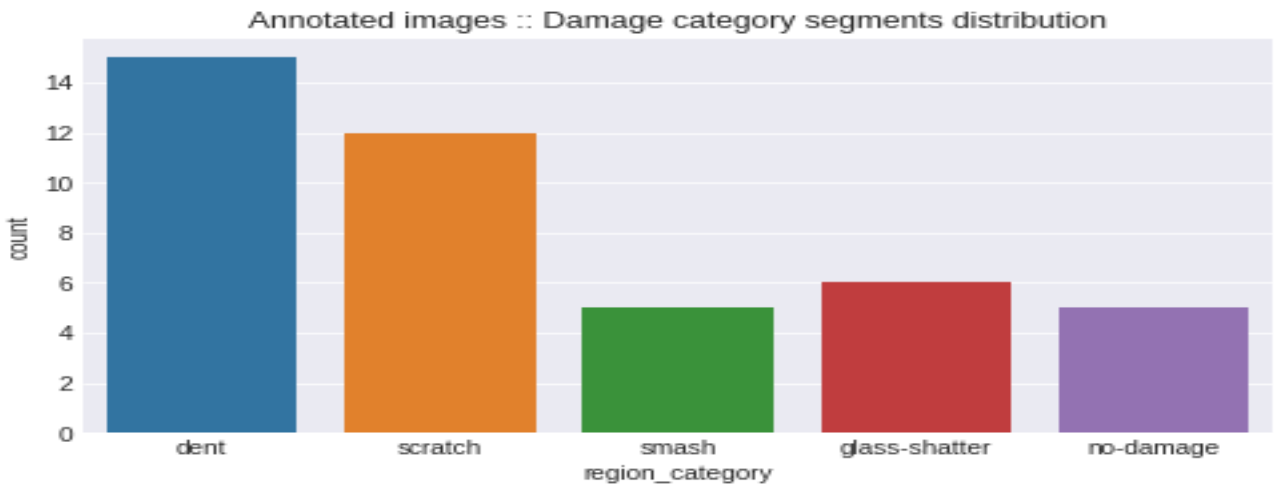
- If the dataset contains small images with small objects (for instance 10x10 px), it will not be able to train the model.

Scenarios:: If the tail of this distribution is long then instances with extreme aspect ratios are present which needs to be taken care of. **If images are too large then it could be resolved by the Crop -> Resize approach**

In the case of object detection/image segmentation models, T**he bigger objects a class has the more likely it is to be underrepresented in the dataset.** Most of the time *the average area of objects in a given class will be inversely proportional to the (label) count.*

Approach:: Images will be resized uniformly

Visualization of the distribution of damage categories (43 samples from Web scraped data)::



3. *Partially labeled data & Imbalances::*

Missing annotations will hamper the learning process of the model. Hence, mixing datasets with non-overlapping classes is not advisable, even though the mixing of datasets could be facilitated by **soft labeling**.

Class imbalances can be a bit of a problem when it comes to object detection. Normally in image classification, for example, one can easily over a sample or downsample the dataset and control each class contribution to the loss.

Approach:: Data annotation process is still in continuation and will be applied to the complete training – validation set.

Scenario:: Partially Labelled data to be avoided. Instance segmentation is implemented

4. **Data augmentation::**

Data augmentation is by far the most important and widely used regularization technique (in image segmentation/object detection).

Applying it to object detection and segmentation problems is more challenging than in simple image classification because some transformations (like rotation, or crop) need to be applied not only to the source image but also to the target (masks or bounding boxes). Common transformations that require a target transform include:

a) Affine transformations b) Cropping c) Distortions d) Scaling e) Rotations

f) Lighting conditions - The lighting condition of the images are varied by adding Gaussian noise in the image.

It is crucial to do data exploration on batches of augmented images and targets to avoid costly mistakes (dropping bounding boxes, etc).

Approach:: Cropping, Scaling - Resizing, Rotations, Horizontal slip applied. To be done using Keras.preprocessing.image.ImageDataGenerator library. Other libraries for augmentation that could be tried are – Albumentations, Imgaug, Augmentor

Project demonstration strategy (tentative plans)

The following is the tentative project demonstration strategy.

1. **5 min** - Setting the context
    a. Brief on Domain information.
    b. Brief information on the problem statement.
2. **5 min** – EDA
    a. A walkthrough on data collection, labeling strategy, etc.
    b. Visualization and explanations of insights gained w.r.t collected data.
3. **15 min** – Demo and solution walkthrough
    a. Demo of the proposed method
    b. A detailed walkthrough on the proposed solution
    c. Explain the results – where it worked and where it didn't
4. **5 min** – Conclusion, Feedback, and QA

Proposed timeline of project stage executions

Weekly progress goals for each of the 4 Capstone Project Mentored Sessions.

Milestone	Planned end date	Expected Outcome
Progress Report 1	Apr 10, 2022	• Web scraping the car damage images. • Set up guidelines for labeling damaged regions. • Label polygon regions for damages and class label per polygon via the VGG Image Annotator tool. • Implement an intrinsic image estimation method as highlighted in stage 2 of the proposed method.
Progress Report 2	Apr 16, 2022	• Train and explore models mentioned in stage 3 of the pipeline. • Evaluate and fine-tune model w.r.t data augmentation, • Hyperparam tuning like model config in YOLO, LR scheduling, etc.
Progress Report 3	Apr 23, 2022	• Train and explore models mentioned in stage 3 of the pipeline. • Evaluate and fine-tune model w.r.t data augmentation, • Hyperparam tuning like model config in YOLO, LR scheduling, etc.
Progress Report 4	Apr 30, 2022	• Select the best approach • Final report w.r.t results and explanation of results w.r.t use-cases where model performed better and where it lacks.

Team members' names

Team 6

1. Pallavi Krishna
2. Tuhina Gupta
3. Prabuvel Venkatasamy Chennaiyan
4. Abhishek Shukla
5. Kishore Nandagiri
6. T Rithvik Kumar
7. Gaurav Batra

References:

1. "Deep Learning Based Car Damage" - Kalpesh Patil Mandar Kulkarni Shirish Karande, TCS Innovation Labs Pune
2. Srimal Jayawardena et al., Image based automatic vehicle damage detection, Ph.D. thesis, Australian National University, 2013.
3. F Samadzadegan and H Rastiveisi, "Automatic detection and classification of damaged buildings, using high-resolution satellite imagery and vector data," The International Archives of the photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 37, pp. 415–420, 2008.
4. Hashmat Shadab Malik, Mahavir Dwivedi, S. N. Omakar, Satya Ranjan Samal, Aditya Rathi, Edgar Bosco Monis, Bharat Khanna and Ayush Tiwari, "Deep Learning Based Car Damage Classification and Detection", EasyChair, Preprint no 3008.