Report on The Task

Case Description:

Apart from the car's functionality, equipment availability, and healthiness, which can only be determined by test drive/manual inspection, car body external damages (scratch, dent, repaint, etc.) play an important role in determining the accurate pricing of the vehicle in the used car industry (both marketplace and brick and mortar dealers). During the car evaluation process, most of these damages are detected and assessed manually from the car images. However, the most recent computer vision frameworks can detect the location of the damage on the car body and assist pricers in quantifying the damage without requiring much manual intervention. This concept will also assist car insurers in automatically assessing damage and processing claims more quickly.

By removing the manual process of damage assessment, automated identification of car exterior damages and subsequent quantification (damage severity) would assist used car sellers (Marketplace) in pricing cars accurately and quickly. In terms of faster claim settlement and thus higher customer satisfaction, the concept is equally useful for property and casualty (P&C) insurance. In this article, I'll walk you through the process of detecting automobile scratches (the most common type of exterior damage) with CNN transfer learning and the Tensorflow backend.

We have a few options for object detection algorithms in a typical application - R-CNN, Fast R-CNN, Faster R-CNN, SSD, and so on. In summary, as with every object detection task, there are three subtasks:

1. **Extracting Regions of Interest (ROI)**: The image is passed to a ConvNet, which returns the region of interest using methods such as selective search (RCNN) or RPN (Region Proposal N/W for Faster RCNN), and then an ROI pooling layer is applied to the extracted ROI to ensure that all of the regions are the same size.
2. **Classification task**: Regions are sent to a fully connected network, which sorts them into several image classes. Scratch('damage') or backdrop will be the situation in our scenario (car body without damage).
3. **Regression task:** Finally, for tightening the bounding boxes, a bounding box (BB) regression is utilized to forecast the bounding boxes for each recognized region (getting exact BB defining relative coordinates)

However, because car scratches/damages are amorphous, getting at square/rectangular shaped BBs is insufficient in our scenario (without a clearly defined shape or form). The specific pixels in the bounding box that correspond to the class must be identified (damage). The precise pixel location of the scratch will only aid in correctly identifying the place and quantifying the damage. As a result, we'll need to add another stage to the pipeline: semantic segmentation (pixel-wise shading of the class of interest), for which we'll utilize the Masked Region-based CNN(Mask R-CNN) architecture.

Mask R-CNN Description:

Mask R-CNN is an instance segmentation model that identifies pixel-by-pixel demarcation for our target object class. So Mask R-CNN has two broad tasks: 1) BB based Object detection (also known as localization task) and 2) Semantic segmentation, which allows segmenting of distinct objects within a scene at the pixel level, regardless of their shapes.

Put these two assignments together. For a given image, Mask R-CNN does get Instance Segmentation.

Mask R-CNN is made up of two parts: **1) BB object identification and 2) semantic segmentation.** It uses a similar architecture to Faster R-CNN for object detection. The sole difference in Mask R-CNN is the ROI step, which employs ROI alignment instead of ROI pooling to preserve ROIs pixel by pixel and prevent information loss. It employs fully convolutional n/w for semantic segmentation (FCN). By producing a pixel-wise classification of each region, FCN builds masks (in our example, binary masks) around the BB objects (distinct objects of interest). As a result, Mask R-CNN reduces the total loss in Instance Segmentation by minimizing the following losses at each phase.

Procedure:

In the following section, I'll go over data preparation and how to apply this principle to real-world car photos using Mask R-CNN.
Step 1: Data Collection - Although we will use transfer learning from a suitable pre-trained(weights) CNN architecture, we must customize the network for our specific use to minimize application-specific loss (loss due to pixel level damage mismatch between ground truth and predicted damage).
Step 2: Data Annotation - Because the notion is supervised learning, we must label the data. This tagging is referred to as annotation in the context of computer vision object detection or object localisation. It is precisely identifying the region of damage in an image and accurately marking them along the boundary of the scratch for our application. We utilized the VGG Image Annotator for annotation (VIA)

Step 3: Setting up the environment - We will use the 'Matterport Mask R-CNN' repository to leverage a few pre-trained CNN n/w weight matrices built on different standard datasets like COCO dataset, ImageNet, and custom functions such as data processing and preparation, configuration setup, model training, creating log-file to save iteration wise weight-matrix objects and n/w losses, object detection, and more before training the model on collected images and annotations(labels). To use the custom training function on the photos and annotations, we must first clone the repository and then follow the repository's exact file-folder structure. The Tensorflow Object Detection API is used to build this Matterport Mask R-CNN.

Step 4: Loading datasets- In this step, we load training and validation photos and assign each one to the appropriate labelling or annotation. I modified the baloon.py code produced for Mask R-CNN to prepare custom 1.py, which loads images and annotations and adds them to a CustomDataset class, according to the application (class label, directory location, shape standardization, etc.).

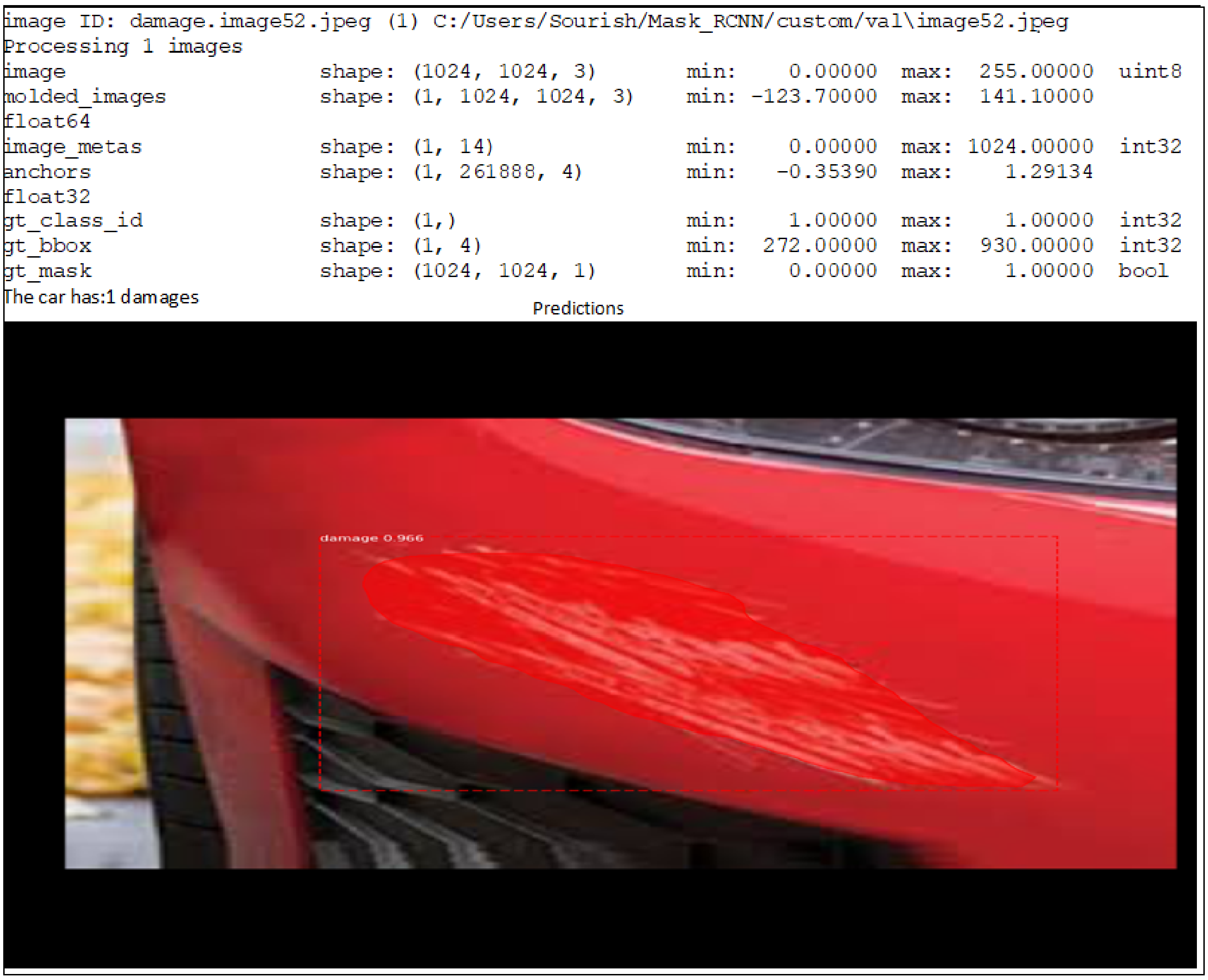
Step 5: Network training - We now need to enhance the base mask RCNN ‘coco.h5' using model training on real images, and the revised weight matrix is saved in 'log' after each iteration(epoch). In addition, iteration/epoch-specific loss information is kept in TensorBoard for monitoring.

Step 6: Model Validation- The revised weight matrix is saved in the 'log' after each iteration(epoch). In addition, iteration/epoch-specific loss information is kept in TensorBoard for monitoring.

Although the majority of the model training is standardized and beyond our control, we can use TensorBoard to see the various training and validation loss components (as stated in the previous section). We can also check the histogram of weights and biases using the saved callbacks (saved weight matrices).

Step 7: Model Prediction - After achieving sufficient and desirable loss monitoring - ideally, both training and validation loss should be monotonically decreasing - we can test the model object on randomly selected validation photos to see how accurate it is at predicting (vehicle damage masking).

The Output will look like the below Images:



```
image ID: damage.image52.jpeg (1) C:/Users/Sourish/Mask_RCNN/custom/val\image52.jpeg
Processing 1 images
image                    shape: (1024, 1024, 3)       min:    0.00000  max:  255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  141.10000
float64
image_metas              shape: (1, 14)               min:    0.00000  max: 1024.00000  int32
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:    1.29134
float32
gt_class_id              shape: (1,)                  min:    1.00000  max:    1.00000  int32
gt_bbox                  shape: (1, 4)                min:  272.00000  max:  930.00000  int32
gt_mask                  shape: (1024, 1024, 1)       min:    0.00000  max:    1.00000  bool
The car has:1 damages
```

Almost 97% Accuracy.

This concept can be implemented as a mobile app or as an API service to help with an automotive appraisal.

Following the identification and masking of automobile damage, the method can assist car evaluators/claim settlement staff in estimating the severity of the damage in terms of dimensions and approximate relative area (w.r.t. the car surface area) of damage. Most crucially, because we're using transfer learning, we don't need to gather as many images and then annotate them, and we don't need to train the model for very long because it starts from trained weights('coco'). This technique can also be applied to detect various forms of obvious automotive problems or faults.