Creating a custom object detection

These instructions can help you to implement a machine-learning algorithm that localizes a car scratch.

Labeling of the images

To be able to determine the position of a scratch on the backside of a car, labeling of the images is required. In this context, you are not only classifying between two pictures. Instead, you are implementing an object detection The following guides refer to the free software LabelIMG, but you can also work with other labeling software.
Start by importing your needed libraries and dependencies and put all images with scratches into the same folder. Start LabelIMG through Python, e.g., by using a jupyter notebook. In LabelIMG, open your folder as the directory and start labeling.
Valuable tips for labeling:

- Use “W” to draw the label and “Strg+S” to save the label.
- Always use the same name for your labels.
- Try to hit the edges of the scratches to achieve high accuracy afterward.

After labeling, you should find one XML document assigned to every picture in the folder. Have a look at it. The coordinates in the document might have something to do with your task in week two!

Developing the custom object detection

You will find many pre-trained models uploaded to “TensorFlow Model Zoo”. You can use those pre-trained models as a base for your endeavors by integrating the link to your desired pre-trained model. Try to find a good compromise between speed and accuracy when choosing the model.

After that, set up your needed paths and install the TensorFlow Object Detection (TFOD).

Try the verification script to check if all modules and libraries are installed. It will immediately return missing libraries or mistakes in your installation.

```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')

!python {VERIFICATION_SCRIPT}
```

After that, create the label map and TF records, copy your model config to the training folder, and update the config for transfer learning. After that, you are ready to train and evaluate your model.

It is recommended to plan enough time for the training of your model. Using a standard device, you should estimate 1.5 to 2 seconds per training step. Training your model overnight is recommended since the training uses much computational power. Try starting with at least 10.000 steps, which means a training time of 15.000 to 20.000 seconds. To check your training speed, you could also start with 100 steps. This will help you determine the approximate waiting time for more training steps.

Detecting from an image

Before detecting from an image, load your model from the last checkpoint (1.000 steps equal approximately 1 checkpoint; can be checked in model folder). After that, build your detection model function.

To detect from the image, it is recommended to work with cv2, matplotlib, and NumPy. You will find a lot of tutorials on how to integrate the link of the picture into your code and show the picture with the detections on your screen.

Performance tuning of your model

Unhappy with your results? To achieve higher performance, you can **try the following steps**:

- Higher quality of input data:
  - Did your split of training and validation/test data make sense? Make sure that all relevant scenarios are represented in your training set.
  - Can you copy and crop the pictures to get more input data?
- Improve labeling:
  - Did you draw the labels with high accuracy?
  - Are all pictures labeled correctly?
- Training and model selection:
  - Should you choose a slower pre-trained model with higher accuracy?
  - Did you conduct enough training steps? Try to raise the number step by step to prevent overfitting of your model.