CarModelClassifier

Release 0.0.1

Manieri, Pericoli, Parietti, Cioffi

Nov 26, 2019

CONTENTS:

CHAPTER

ONE

CARMODELCLASSIFIER

This is a project of the *Deep Learning for Computer Vision* course held by Professor **Gaia Rubera** and **Francesco Grossetti** at Bocconi university.

1.1 Getting Started

This project allows to predict the model, the brand and the year of a car based on an image of that. It is organized in 3 main files to be run by the user, 5 folders for the code and 1 folder for the data. All the detailed info about the code and the functions can be found in the folder *"docs"*.

1.2 Prerequisites

The main requirement to satisfy is to have a folder called *"data"* inside the main folder. In this folder some of the data are necessary for all the project while some others are created by the scripts. The required root data are the following:

1. **raw_data**:
    - **train_cars_new**: All the train images must be in this folder
    - **YOLO_weights**: This contains the weights for object detection
2. **labels**:
    - **all_labels_new.csv**: This file should contain for each image name the real class
    - **models_info_new.csv**: This file combines each class with the model features, that is: *model*, *brand* and *year*.

1.3 Project Organization

The code is organized in the following 5 folders:

1. **CarModelClassifier**: This folder contains all the modules and the related functions to perform the main tasks of the project.
    - **estimation.py**: In this file there are the functions for the model prediction and evaluation.
    - **models.py**: This file contains an abstract class which identifies how the class of each new model should be set. Furthermore it contains two main models used to do image classification.
    - **pipeline.py**: This file contains the main function *"run"* which defines the flow of the train.

- **splitter.py**: This file contains the function for the splitting of the data in train, test and validation.
- **utils.py**: Here are all the functions required in the different scripts.
- **yolo.py**: This script is necessary in the case an object location of the car and a subsequent cropping of the car is required.

2. **config**: This folder contains the configuration files necessary to modify the main global parameters for each model. Having configuration files allows not to modify the scripts directly every time it is necessary to change some global parameters like the number of epochs or the size of the images.
3. **docs**: This folder contains all the useful files necessary to create the documentation of the code
4. **guess_make**: Here you can find all the material needed to create a graphical interface for the model prediction.
5. **tests**: Here are the tests of some functions of the project.

1.3.1 Flow of training explanation

Here the main process of the project is explained in details. The steps are the following:

1. Initialized the following parameters:
   - **–params_file**: This parameter lets the user specify the name of the configuration file, that must be placed in the folder **config**. The file to be passed must be a yaml file containing the following parameters:
     - **seed**
     - **train_batch_size**
     - **validation_batch_size**
     - **test_batch_size**
     - **epochs**
     - **IMG_HEIGHT**
     - **IMG_WIDTH**
     - **data_path**
   - **–username**: This will be the name of the folder in which important data, like the model weights and architecture, will be saved for future usage.
   - **–shows_only_summary**: This is a boolean parameter which allows the user to decide whether to train the model or just to show the summary of the architecture to see the number of the parameters that will be trained. If True, the script stops right after having shown the model summary.
   - **–net**: This allows the user to specify which model architectures among the ones present in *"CarModel-Classifier/models.py"*. The possible values are:
     - **effnetb1**
     - **effnetb7**
     - **prototype**
   - **–bounding_cpu**: A boolean parameter needed if the user wants to limit the cpu usage for the process of training.
   - **–split_data**: This parameter allows the user to decide whether to perform the splitting of the data
   - **–crop_images**: If True the model performs object detection on the images, first locating the car and then cropping the image in such a way the new image will be only the box containing the car.

2. Once the parameters are defined the file **train_main.py** calls the function *"run"* in the module **pipeline.py**.
3. Checking whether to crop images using the script **yolo.py**. If True, the function *"crop"* gets the yolo weights from the folder raw_data and performs object detection on the images. Then it crops the images and put them in a folder called *object_detection_data* inside the data folder.
4. Checking whether to split the data using the script **splitter.py**. If True the splitting is performed using the function *"split"*. If the parameter **crop_images** is True, then the split is performed using the images in the folder *"data/object_detection_data/output_images_cropped"*, otherwise using the folder *"data/raw_data/cars_train_new"*. The splitting is performed in the following way:
    1. The splitting is performed in a stratified fashion using the file *"data/labels/all_labels_new.csv"*. From this file 3 new csv files are created in the same folder: *train_labels.csv*, *test_labels.csv*, *validation_labels.csv*. Then, according to these 3 files the images in the initial folder are splitted in 3 new folders inside the **data** folders: **train**, **test**, **validation**.
5. Checking whether to limit the cpu usage.
6. Initializing the images using the keras generators.
7. Initializing and training the pre-specified model.
8. Saving the results for feature prediction, evaluation and to have an History of the models run. The function used to save the results uses the *username* pre-specified to create a folder having the same name inside the folder *"data/models"*. The name of the new created folder will be a combination of the username and number, so that if you have more models having the same username, they will be saved with the same name and a chronological number at the end to avoid the risk of overwriting existing folders. Inside this folder the function creates 4 files:
    - **model.h5**: These are the weights after the training of the model.
    - **architecture.yml**: This is the architecture of the model.
    - **evaluation.csv**: This is a csv file containing the loss and the accuracy in train and in validation for each epoch. This file is essential to crate plots from which one can see at which epoch the model starts overfitting.
    - **initial_parameters.yml**: This is the configuration file passed as a parameter. This allows the user to understand which were the values of the global parameters for each model.

1.4 Using the model

1.4.1 Training

To perform the train of the model:

1. Make sure you have satisfied the prerequisites.
2. Launch from a terminal the script **train_main.py** specifying the following parameters:
    - **–params_file**: This parameter lets the user specify the name of the configuration file, that must be placed in the folder **config**. The file to be passed must be a yaml file containing the following parameters:
        - **seed**
        - **train_batch_size**
        - **validation_batch_size**
        - **test_batch_size**
        - **epochs**

  - **IMG_HEIGHT**
  - **IMG_WIDTH**
  - **data_path**
- **–username**: This will be the name of the folder in which important data, like the model weights and architecture, will be saved for future usage.
- **–shows_only_summary**: This is a boolean parameter which allows the user to decide whether to train the model or just to show the summary of the architecture to see the number of the parameters that will be trained. If True, the script stops right after having shown the model summary.
- **–net**: This allows the user to specify which model architectures among the ones present in *"CarModelClassifier/models.py"*. The possible values are:
  - **effnetb1**
  - **effnetb7**
  - **prototype**
- **–bounding_cpu**: A boolean parameter needed if the user wants to limit the cpu usage for the process of training.
- **–split_data**: This parameter allows the user to decide whether to perform the splitting of the data
- **–crop_images**: If True the model performs object detection on the images, first locating the car and then cropping the image in such a way the new image will be only the box containing the car.

1.4.2 Evaluation

To perform the evaluation of the model, that is to evaluate the accuracy on new images:

1. Make sure to have the following data:
   - The folder *"data/models/final_model"* and inside the files:
     - **model.h5**
     - **initial_parameters.yml**
   - If evaluating on test_data:
     - *"data/test"* containing the images.
     - *"data/labels/test_labels.csv"* containing for each image name the related class.
   - If evaluating on new_data:
     - The folder *"custom_evaluaton"* containing:
       * *"images"* containing the new images.
       * *"test_labels.csv"* containing for each new image name the related class
2. Launch from a terminal the script **evaluation_main.py** specifying the following parameters: * **–custom_images**: If True the evaluation will be performed using new images in the folder *"custom_evaluation"*, otherwise the test data will be used. * **–test**: This parameter is True only when performing a test with pytest. Hence it should be False when performing prediction on new images.

1.4.3 Prediction

To perform the prediction on new images:

1. Make sure to have the following data:
   - The folder *"data/models/final_model"* and inside the files:
     - **model.h5**
     - **initial_parameters.yml**
   - The file *"models_info.csv"* inside the folder *"data/labels"*.
   - The folder *"custom_evaluaton"* containing: * *"images"* containing the new images. * *"test_labels.csv"* containing for each new image name the related class
2. Launch from a terminal the script **prediction_main.py** specifying the following parameters: * **–test**: This parameter is True only when performing a test with pytest. Hence it should be False when performing prediction on new images.

1.5 Running the tests

1.6 Authors

- **Martina Cioffi** - https://github.com/martinacioffi
- **Edoardo Manieri** - https://github.com/edoardomanieri
- **Valentina Parietti** - https://github.com/ValentinaParietti
- **Edoardo Pericoli** - https://github.com/Edoardopericoli

CHAPTER

TWO

CARMODELCLASSIFIER ESTIMATION

Estimation module.

2.1 List of functions

CarModelClassifier.estimation.**evaluation**(*custom_images=False*, *test=False*)

Perform evaluation of the model.

Images to evaluate the model are taken from the folder "custom_evaluation/images" while the labels of these images are in the file "/custom_evaluation/test_labels.csv"

Parameters

- **custom_images** (*bool, optional*) – if True images to evaluate the model are taken from the folder "custom_evaluation/images" while the labels of these images are those in the file "/custom_evaluation/test_labels.csv". if False images to evaluate the model are taken from the folder "data/test" while the labels of these images are those in the file "data/labels/test_labels.csv".
- **test** (*bool, optional*) – if True, test the correct working of the function, by default False

Returns accuracy of the model.

Return type float

CarModelClassifier.estimation.**prediction**(*test=False*)

Perform prediction of new images.

Images to evaluate the model are taken from the folder "custom_evaluation/images".

Parameters **test** (*bool, optional*) – if True, used to test the function, by default False

Returns The output dataframe.

Return type DataFrame

CHAPTER

THREE

CARMODELCLASSIFIER MODELS

class CarModelClassifier.models.**EffnetB1**(*train_generator*, *initial_parameters=None*)
Efficient Net Version B1 implementation

setup_model()
Build the complete Model.

Returns The complete Model.

Return type *Net*

class CarModelClassifier.models.**EffnetB7**(*train_generator*, *initial_parameters=None*)
Efficient Net Version B7 implementation

setup_model()
Build the complete Model.

Returns The complete Model.

Return type *Net*

class CarModelClassifier.models.**Net**
Abstract class that is the super class of all models

class CarModelClassifier.models.**Prototype**(*train_generator*, *initial_parameters=None*)
Basic model used for baseline and test purposes.

CHAPTER

FOUR

CARMODELCLASSIFIER PIPELINE

CarModelClassifier.pipeline.**run**(*initial_parameters_path='./config/initial_parameters.yml'*, *username='trial'*, *shows_only_summary=False*, *net=<class 'CarModelClassifier.models.EffnetB1'>*, *bounding_cpu=False*, *split_data=True*, *crop_images=False*)

Perform all the steps for training and saving the model.

Parameters

- **initial_parameters_path** (*string, optional*) – the name of the yaml file containing the initial parameters.
- **username** (*string, optional*) – the name used to save the model in the folder "data/models".
- **shows_only_summary** (*bool, optional*) – if True the script exits after having shown the summary.
- **net** (*net, optional*) – specify the architecture to be used
- **bounding_cpu** (*bool, optional*) – if True, it limits the cpu usage
- **split_data** (*bool, optional*) – if True perform splitting
- **crop_images** (*bool, optional*) – if True, perform object detection and cropping of images

CHAPTER

FIVE

CARMODELCLASSIFIER SPLITTER

CarModelClassifier.splitter.**split**(*initial_parameters*, *train_size=0.8*, *crop_images=False*)

Perform splitting of the data.

It splits the data starting from the csv file "data/labels/all_labels_new.csv" in a stratified fashion. The folder from which the images are taken dependes on the parameter "crop_images". Then it creates 3 new csv files in the same folder: "train_labels.csv", "test_labels.csv", "validation_labels.csv". Finally creates 3 folders in "data" called: "train", "test", and "validation" containing the images splitted.

Parameters

- **initial_parameters** (*string*) – the name of the yaml file containing the initial parameters.
- **train_size** (*float, optional*) – the size of the train when splitting is performed. By default, the test and the validation will have the same size.
- **crop_images** (*bool, optional*) – If True, the images to be splitted are taken from the folder "data/object_detection_data/output_images_cropped", otherwise they are taken from "data/raw_data/cars_train_new".

CHAPTER

SIX

CARMODELCLASSIFIER UTILS

Utils module.

6.1 List of functions

class CarModelClassifier.utils.**FixedDropout**(*rate*, *noise_shape=None*, *seed=None*, ***kwargs*)

Fixed Dropout Layer

CarModelClassifier.utils.**bound_cpu**(*n_threads=8*)

Limit the CPU usage to a limited number of threads.

Parameters **n_threads** (*int, optional*) – number of threads, by default 8

CarModelClassifier.utils.**get_generator**(*imagedatagenerator*, *labels_df*, *directory*, *initial_parameters*, *train=True*)

Build the generator from dataframe.

Parameters

- **imagedatagenerator** (*ImageDataGenerator*) – ImageDataGenerator for the generator
- **labels_df** (*DataFrame*) – DataFrame containing labels.
- **directory** (*string*) – directory containing the images
- **initial_parameters** (*dict*) – a dict containing parameters.
- **train** (*bool, optional*) – if True is used for the training dataset, by default True

Returns the Train Generator.

Return type TrainGenerator

CarModelClassifier.utils.**get_image_generators**()

Create image generators for train and validation.

Returns ImageDataGenerator for train and validation.

Return type ImageDataGenerators

CarModelClassifier.utils.**load_labels_dfs**(*initial_parameters*)

Load the labels files for training.

Parameters **initial_parameters** (*dict*) – a dict containing parameters.

Returns train and test dataframes.

Return type Dataframes

CarModelClassifier.utils.**load_parameters**(*parameters_path*)
Load the parameters from the config file.

Parameters **parameters_path** (*string*) – path of the config file

Returns a dict containing parameters.

Return type dict

CarModelClassifier.utils.**save_model_architecture**(*username*, *model*, *initial_parameters*)
Save the entire model in a .h5 file.

Save the model in a h5 file and the architecture in a yml file

Parameters

- **username** (*string*) – name of the folder in which put the saved the model
- **model** (Net) – the model used for training.
- **initial_parameters** (*dict*) – a dict containing parameters

CarModelClassifier.utils.**save_model_info**(*username*, *model*, *initial_parameters*, *history*)
Save model architecture and performance

Parameters

- **username** (*string*) – name of the folder in which put the saved the model
- **model** (Net) – the model used for training.
- **initial_parameters** (*dict*) – a dict containing parameters
- **history** (*DataFrame*) – History DataFrame

CarModelClassifier.utils.**save_model_performance**(*username*, *history*, *initial_parameters*)
Save model performance in a DataFrame.

Parameters

- **username** (*string*) – name of the folder in which put the saved the model
- **history** (*DataFrame*) – History DataFrame
- **initial_parameters** (*dict*) – a dict containing parameters

CarModelClassifier.utils.**setting_log**()
Set basic log configuration.

CarModelClassifier.utils.**swish**(*x*)
Custom activation function.

Parameters **x** (*float*) – input float

Returns the input number multiplied by the sigmoid function apply to it

Return type float

CarModelClassifier.utils.**train_model**(*train_generator*, *validation_generator*, *initial_parameters*, *train_df*, *model*)
Perform the model training

Parameters

- **train_generator** (*Generator*) – Train Generator
- **validation_generator** (*Generator*) – Validation Generator

- **initial_parameters** (*dict*) – a dict containing parameters
- **train_df** (*DataFrame*) – train DataFrame
- **model** (Net) – Model used for training

Returns history DataFrame

Return type DataFrame

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX