TrashBot

Release 0.0.1

Jul 27, 2020

Source

CleanRobotics TrashBot is a smart assisted waste sorrting bin that uses AI and Computer Vision to segregate different kinds of trash.

This project is my work for the Insight Artificial Intelligence Fellowship, New York in collaboration with CleanRobotics. The idea behind the project is to develop and object detection solution for the TrashBot. Find out more about TrashBot.

CHAPTER 1

Dataset

1.1 data

1.1.1 augment_data

1.1.2 create_coco

1.1.3 generate_json

Helper function to generate COCO style annotation records one by one.

data.generate_json.**get_json**(*current_idx: int*, *image: str*, *annotation_idx: int*, *annotations: list*, *output: str*)

Generate COCO style record for each input image.

Parameters

- **current_idx** (*int*) – Current image id
- **image** (*str*) – Filename of image file
- **annotation_idx** (*int*) – Current annotation index
- **annotations** (*list*) – List of bounding box co-ordinates
- **output** (*str*) – Path to output json file to append records to

Returns latest annotation index

Return type int

1.1.4 supervisely2coco

1.1.5 utils

Utils for converting from Supervisely to MS-COCO.

class data.utils.**NpEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

Helper class for JSON dumping.

default(*obj*)

Return serializable object

data.utils.**annotation_split**(*annotations: list*, *train: list*, *valid: list*, *test: list*) → Tuple[list, list, list]

Create annotations splits.

Parameters

- **annotations** (*list*) – List of dictionaries describing each annotation
- **train** (*list*) – Train set images
- **valid** (*list*) – Valid set images
- **test** (*list*) – Test set images

Returns tuple(list, list, list) containing train, valid and test annotations after splitting

data.utils.**convert_image**(*id: int*, *name: str*, *jsons*, *category: dict*, *base_dir: str*, *image_name=False*, *start_idx=0*) → Tuple[dict, list]

Convert single image annotations to COCO representation

Parameters

- **id** (*int*) – image id
- **name** (*str*) – image filename
- **jsons** (*str*) – json object containing annotations in supervise.ly format
- **base_dir** (*str*) – path to base directory for annotations
- **image_name** (*bool*) – flag indicating whether to save filenames with full path or not
- **start_idx** (*int*) – annotation index

Returns tuple(dictionary, list) containing annotation for image info, objects

data.utils.**create_json**(*base_json: dict*, *images: list*, *annotations: list*, *output: str*, *filename: str*) → None

Create COCO json annotation record.

Parameters

- **base_json** (*dict*) – Base JSON dictionary containing metadatas
- **images** (*list*) – List of images
- **annotations** (*list*) – List of annotations
- **output** (*str*) – Path of output directory
- **filename** (*str*) – Filename to save generated JSON

Returns None

data.utils.**dataset_split**(*images: list*, *train_split: int*, *valid_split: int*, *test_split: int*) → Tuple[list, list, list]
Split dataset into train, test and valid sets.

Parameters

- **images** (*list*) – List of all images
- **train_split** (*int*) – Integer indicating proportion of images as train set
- **valid_split** (*int*) – Integer indicating proportion of images as valid set
- **test_split** (*int*) – Integer indicating proportion of images as test set

Returns tuple(list, list, list) with train, valid and test dataset images as lists after splitting

data.utils.**get_all_annotation_files**(*base_dir: str*) → Tuple[list, list]
Get all annotation filenames and corresponding jsons

Parameters **base_dir** (*str*) – path to base directory for annotations

Returns tuple(list, list) containing filenames(sans extension) and associated annotation json files

data.utils.**get_categories**(*meta: str*) → dict
Get all categories for the given dataset

Parameters **meta** (*str*) – path to meta.json file

Returns dict with categories as key, order index as value

CHAPTER 2

Augmentations

2.1 augmentation

2.1.1 augment

2.1.2 horizontal

Flipping transformation for data augmentation.

class data.augmentation.horizontal.**RandomHorizontalFlip**(*prob: float = 0.5*)

Bases: object

Randomly flip an image with probability p.

Parameters **p** (*float*) – Probability of flipping an image

Returns Flipped image as a numpy array numpy.ndarray: Transformed bounding boxes

Return type numpy.ndarray

2.1.3 rotate

2.1.4 scale

2.1.5 translate

2.1.6 utils

CHAPTER 3

TrashNet

3.1 EfficientDet

3.1.1 effnet

Initialize EfficientNet backbone.

class trashnet.effdet.effnet.**EfficientNet**
Bases: torch.nn.modules.module.Module

Create a nn.Module from pretrained EfficientNet

forward(*x*)
Forward pass through EfficientNet to generate feature maps.

3.1.2 layers

3.1.3 losses

Focal Loss for EfficientDet

The code is modified from a PyTorch implementation of RetinaNet where FocalLoss was first introduced

https://arxiv.org/abs/1708.02002 https://github.com/yhenon/pytorch-retinanet/blob/master/retinanet/losses.py

class trashnet.effdet.losses.**FocalLoss**
Bases: torch.nn.modules.module.Module

Focal Loss for EfficientDet. As defined in RetinaNet: https://arxiv.org/abs/1708.02002

forward(*classifications*, *regressions*, *anchors*, *annotations*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

> **Note:** Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

`trashnet.effdet.losses.`**`calculate_IoU`**$(a, b) \rightarrow$ float

Helper function to calculate Intersection over Union for ground truth bounding box and predicted bounding box

Returns float as the IoU score for the given inputs

3.1.4 models

3.1.5 utils

3.2 COCO Dataset

3.2.1 dataset

3.2.2 transforms

3.3 train

3.4 test

CHAPTER 4

Setup

Welcome! This tutorial will walk you through setting up the project, creating the necessary environments and installing all the dependencies in one go.

Start by cloning the project repository from GitHub. Once you've cloned the repository, enter the directory and visually inspect that the clone was successful

On a Linux machine, it would look something like this

```
git clone https://github.com/jsaurabh/TrashBot.git
cd Trashbot
ls
```

Once you've confirmed everything is the way it is supposed to be, go ahead and start installing all the dependencies

```
conda env update
```

The command above will setup a conda environment named *trashbot* that will contain all the dependencies needed for the project. After a couple of minutes, the environments will be ready. To start using the environment, activate it

```
conda activate trashbot
```

To change the name of the environment, edit the first line in the *environment.yml* file in the root of the project directory.

CHAPTER 5

Download Data

On the Supervise.ly platform, navigate to the concerned Workspace and select the project to work on. Now, download the data as a zipped archive, choosing to download the images along with the annotations.

Supervise.ly will then run a batch job, generating the necessary .zip archive. Once it's downloaded, extract using a standard archiver. To do so using the command line, open a terminal prompt and navigate to the download path. Now, type in the following command:

```
unzip -q DOWNLOADED_FILE.zip -d SAVE_PATH
```

where DOWNLOADED_FILE.zip is the name of the downloaded file and SAVE_PATH is the destination on the disk.

You're now ready to generate the COCO-style dataset records expected by the neural network.

CHAPTER 6

Generate COCO style dataset

After downloading and extracting the data, we need to convert it to a format that the network expects. We'll do that now.

Remember where you extracted your archive. For this tutorial, we'll consider **~/Desktop/TrashBot** as the extracted location.

Now, inside the root directory of the project, make the build_dataset bash file an executable. On Linux, you can do it like this:

```
chmod u+x build_dataset.sh
```

Now, go ahead and run the build_dataset.sh script as follows:

```
./build_dataset.sh --data_path ~/Desktop/TrashBot --output ~/Desktop/coco
```

where –output represents where I want the COCO dataset to be stored on disk and –data_path represents my Supervise.ly extracted archive

That's all! You're all set. Now, go ahead and train a model.

CHAPTER 7

Train

Now that you've got your data in a format the model expects, let's train an EfficientDet to do object detection on our dataset.

Within the root directory of the project, run the following command

```
python trashnet/train.py --num_epochs NUM_EPOCHS --path DATA_PATH/dataset
```

where NUM_EPOCHS is the number of epochs you want to train the network for and DATA_PATH is the path to the COCO style dataset that you set up previously.

For additional hyperparameter choices available during training, use help

```
python trashnet/train.py --help
```

The training loop comes with default hyperparameters that have been tested to work on the dataset, but feel free to try and experiment.

Depending on the underlying hardware and the number of epochs you're training for, it can take anywhere from a couple of minutes to a day for the network to finish training. Go ahead and grab a coffee while the network learns.

CHAPTER 8

Test

Now that you've got your data in a format the model expects, let's train an EfficientDet to do object detection on our dataset.

Within the root directory of the project, run the following command

```
python trashnet/test.py --path DATA_PATH --pretrained PRETRAINED_PATH --output OUTPUT_
↪PATH
```

where DATA_PATH is the path to the COCO style dataset that you set up previously, PRETRAINED_PATH is the path to the pretrained models from the training loop and OUTPUT_PATH is the path you want to save the predicted images to.

For additional hyperparameter choices available during training, use help

```
python trashnet/test.py --help
```

When altering the hyperparametrs for the testing loop, to get the best results, try to be consistent with the hyperparameter choices for the training loop.

Python Module Index

d

t

Index