SSD-Pytorch

Single-shot models are a popular, powerful, and especially nimble networks for both localization and detection tasks in a single forward sweep of the network, resulting in significantly faster detections while deployable on lighter hardware. (Quoted from the original repo)

In this README, details about how to prepare the dataset and how to run the Quantitative object detection benchmark is described.

For the dataset processing using object detection in case of the *source* in the object-context approach, refer to the main README.md file.

Latest Pytorch implementation taken from:
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

Setting up the SSD-Pytorch.

First install all the needed requirements by running,

```
pip install -r requirements.txt
```

Before running any object detection related processes, the checkpoint needs to be downloaded and saved in a convenient location. The checkpoint can be downloaded from:
https://drive.google.com/file/d/1bvJfF6r_zYl2xZEpYXxgb7jLQHFZ01Qe/view

Once this checkpoint is downloaded the `detect.py` script can be run to test if everything is in order. This script can be run with,

```
python detect.py --img_path <PATH_TO_TEST_IMAGE> --checkpoint <PATH_TO_CHECKPOINT>
```

Detailed usage,

```
usage: python detect.py [-h] [--img_path IMG_PATH] [--min_score MIN_SCORE] [--id ID]
                  [--checkpoint CHECKPOINT]

optional arguments:
  -h, --help            show this help message and exit
  --img_path IMG_PATH   Path to image to run the detection on
  --min_score MIN_SCORE
                        Minimum confidence score for prediction
  --id ID               Identifier to add to the image save name # TO BE IGNORED
  --checkpoint CHECKPOINT
                        Model checkpoint for eval
```

The detected image should be saved in this folder with a _detection suffix added to the original image filename.

A sample image is provided in the folder, to use this run,

```
python detect.py --img_path sample.png --checkpoint <PATH_TO_CHECKPOINT>
```

P.S. The sample image was an enhanced image using adapted style. (Also shown in report)

Running evaluation with SSD.

Before running the evaluations for VOC formatted datasets, the `create_data_lists.py` script is run which processes all the VOC annotations and images and stores them in multiple json files. Once these json files are obtained the `exdark_eval.py` file can be run. This file contains code specific to running ExDark formatted voc datasets and features to group the results by lighting conditions. The results are saved in a easy to use csv format. More details about changes are explained in code walkthrough.

The process followed to quantitatively evaluate different enhancement methods are as follows,

1. A copy of the original ExDark dataset folder (in VOC format) is created.
2. The original images in the copy are replaced by enhanced images using `copy_enlighten_generated_files()` in the `Tools/util_fns.py`.
3. The json files for this copy are generated at some location using `create_data_lists.py`.
4. The path to the json files is provided to the `exdark_eval.py` script and this calculates the quantitative metrics as presented in the report.

Testing sample detections with SSD

To test sample detections with SSD without evaluation, the `detect.py` script can used as mentioned above.

Code Walkthrough

This codebase was heavily modified to allow for different functionalities on the ExDark dataset.

1. The `exdark_eval.py` script was created from the `eval.py` script but with different AP calculation functions and additional features not present in `eval.py`.
2. The `eval.py` was also cleaned up for easier usage with argparse
3. The AP calculation used in `exdark_eval.py` is implemented from line 292 onwards in the `utils.py` script. Several modifications are made to allow the change in labels and object mappings programmed in the `utils.py` script.
4. Smaller changes are made throughout the script to parse annotations and create data lists.