EnlightenGAN

EnlightenGAN is a highly effective unsupervised generative adversarial network that can be trained without low/normal-light image pairs, yet proves to generalize very well on various real-world test images. A series of innovations for the low-light image enhancement problem, including a global-local discriminator structure, a self-regularized perceptual loss fusion, and attention mechanism are used.
Paper: https://arxiv.org/abs/1906.06972

Original README is saved as original_README.md.

Link to authors github repo: https://github.com/TAMU-VITA/EnlightenGAN

Link to modified github: https://github.com/surajpaib/EnlightenGAN

Installation and Dependencies

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

The code in this repository does not work without a GPU so please ensure that a GPU is present while running any of the code. Efforts were made to convert the code to be able to run on the CPU but most of the code writen by the authors is written with cuda runtime in mind so this turned out to be a difficult task and was abandoned in favor of investing time into other more important experiments.

Training

In this part of the README instructions on how the training data can be downloaded and training process can be initiated are described. This is specific to the project since it involves a different dataset and training parameters compared to the original work.

Downloading the data for training

The processed object-context paired data can be download from : https://drive.google.com/open?id=1ghNO6R8UNEoyWy9ugjEQ2pXgZbyVPe-y
After completing the download, unzip the dataset and place it in a convenient location. The folder should have `train_A, test_A, train_B, test_B` subfolders.

Details about how this data was processed and creating this processed data from scratch are described in the top-level README.md

Here it is assumed that the data is processed and ready to be used in the GAN training/prediction process.

Configuring and Running the training

To replicate the training process first the visdom server needs to be started which allows monitoring the training graphs and images across epochs.

`nohup python -m visdom.server -port=8097 &`

Instead of using `nohup` (for non-POSIX systems) this command can be run `pythonw.exe -m visdom.server -port=8097` (Not tested , the idea is to run the visdom server in the background, alternatively it can just be run in another terminal)

To run the training

```
python scripts/script.py --train --data_path <PATH>
```

Here `<PATH>` should be the location where the preprocessed data was placed.

The above script will train with the default configuration for 200 epochs with "car" objects and the Adapted loss. To change these configurations, look at the detailed usage below,

Detailed Usage:

```
usage: python script.py [-h] [--port PORT] [--train] [--object OBJECT] [--predict]
                 [--name NAME] [--gpu_ids GPU_IDS] [--data_path DATA_PATH]
                 [--loss_type LOSS_TYPE]

optional arguments:
  -h, --help            show this help message and exit
  --port PORT
  --train
  --object OBJECT       Mention object for Object specific GAN training
  --predict
  --name NAME           For training, the name of the directory where
                        checkpoints are created, for testing directory from
                        which it is loaded
  --gpu_ids GPU_IDS     GPU ids to use, to use 2 GPUs set to 0,1
  --data_path DATA_PATH
                        Path to the downloaded dataset
  --loss_type LOSS_TYPE
                        Choose between relu5_1 | relu5_3 | stylefeat
```

Change the loss_type to train with different losses and choose different object to train with other objects.

More objects were not trained with in this project due to each training process taking ~6 hours on 2x Tesla V100 on the Aachen Cluster. Limited jobs were accepted with long wait queues.

Once the script is run, the models will be saved in `checkpoints` under a folder created based on the `--name` argument provided. This is set to `enlightening` by default.

http://localhost:8097 can be visited to monitor the training process.

A small sample dataset is provided to test the training, to use this run,

```
python scripts/script.py --train --data_path SampleData/
```

Note: The batch size in `scripts/script.py` in line 33 has been set to 2 to allow testing on GPUs with smaller memory. The original batch size used was 32 across each GPU.

Testing/Running the Enhancement

To run only the image enhancement or test different pretrained models follow the section below,

Downloading the pretrained models.

Please download the models from the gdrive link below: (Please download the entire folder/ alternatively a sample model is provided with the code, skip the download and jump to Testing process section)
https://drive.google.com/open?id=1N4faPXW3OVfUnkSoQ2YLrGtut6aXMfq1

Once the folder is unzipped, place its contents in a folder called `checkpoints` in this directory.
At the end of this process the `checkpoints` folder should have the following subfolders,

```
style_loss_car
enlightening
final_style
car_object_trained
car_object_trained_deeper_vgg
```

Each of the above subfolders contains weight files needed to run the inference/enhancement. These also correspond to different experiments run through the project.

Testing process

To run the prediction process, first create a folder in any location. In this folder, create two subdirectories called

```
test_A
test_B
```

Place all the files that need to be converted/enhanced in `test_A` folder. Place a dummy image in the `test_B`. This is needed to avoid the dataloader crashing during testing.

Once this is done run the script
`python scripts/script.py --predict --data_path <PATH> --name <CHECKPOINT_SUBFOLDER_NAME>`

The `<PATH>` corresponds to the folder where images are contained in the `test_A` `test_B` format as shown above. The `<CHECKPOINT_SUBFOLDER_NAME>` is one of the subfolder names in the `checkpoints` folder as mentioned in the pretrained model download instructions. Note that, this needs to be just the name and not the path.

For testing purposes, a small sample dataset along with adapted style checkpoint is provided, to use this:
Run,

`python scripts/script.py --predict --data_path SampleData --name final_style`

Results

Once the above command is run, the images will be processed and placed in `ablation` folder in this directory under the name of the checkpoint provided. Run through the subdirectories to find a folder called images with all the enhanced images.

CLAHE

Evaluations using CLAHE are run with a custom implemented CLAHE runner from the OpenCV library. The code for this can be found under `util/clahe.py`.

Code Walkthrough!

The interesting part of the README!

As mentioned in the report and earlier through the README, multiple loss types can be selected and object-context based pairing is incorporated which was absent in the original repo.

The newly integrated losses are incorporated by modifying `models/networks.py`. Details about the modifications are as follows,

1. The forward pass of the VGG network from line 961 onwards was modified to save feature vectors into tensors at different layers of the network.
2. The features to be considered for style based loss were added in the forward pass and the actual calculation of the loss was added in the `compute_vgg_loss` function in the PerceptualLoss class from line 1051 onwards. This was done by integrating and modifying code from https://github.com/ashwindcruz/perceptual-loss-style-transfer.

The object-context pairing was integrated by modifying the `data/unaligned_dataset.py` to add object specific loading from the provided data_path. Using this new dataset caused the code to break at the

tranforms module due to an error by the authors in the code with improper reshaping. This was fixed by changing the default tranforms in the `data/base_dataset.py`

Apart from these specific changes, multiple changes were made to `options/base_options.py`, `scripts/script.py` during the experimentation process.

For a full log of changes made, git diff at this repo can be looked at:
https://github.com/surajpaib/EnlightenGAN/commits/master

Although these changes might seem minor in hindsight, they required a thourough understanding of the codebase and the original paper to ensure that naive changes that effect the performance adversely are not made. A lot of time was spent on understanding this codebase which was a very rewarding learning process indeed.

P.S. The codebase was also migrated from pytorch 0.4.0 to work with the latest version of pytorch.