White Noise Reduction Using Style Based Generative Adversarial Networks

Description

Augmented Implementation of the pSp implementation to train Images on white noise reduction

Getting Started

Prerequisites

- Linux or macOS
- NVIDIA GPU + CUDA CuDNN (CPU may be possible with some modifications, but is not inherently supported)
- Python 2 or 3

Installation

- Clone this repo:

```
git clone https://github.com/rahuls02/Image-Noise-Reduction.git
cd Image-Noise-Reduction
```

- Dependencies:
  We recommend running this repository using Anaconda. All dependencies for defining the environment are provided in `environment/psp_env.yaml`.

Pretrained Models

If you wish to use one of the pretrained models for training or inference, you may do so using the flag `--checkpoint_path`.

Noise Level trained on	Link
55% Noise	link
70% Noise	link

In addition, we provide various auxiliary models needed for training your own pSp model from scratch as well as pretrained models needed for computing the ID metrics reported in the paper. | Path | Description | :— | :———- |FFHQ StyleGAN | StyleGAN model pretrained on FFHQ taken from rosinality with 1024x1024 output resolution. |IR-SE50 Model | Pretrained IR-SE50 model taken from TreB1eN for use in our ID loss during pSp training. |CurricularFace Backbone | Pretrained CurricularFace model taken from HuangYG123 for use in ID similarity metric computation. |MTCNN | Weights for MTCNN model taken from TreB1eN for use in ID similarity metric computation. (Unpack the

tar.gz to extract the 3 model weights.)

By default, we assume that all auxiliary models are downloaded and saved to the directory `pretrained_models`. However, you may use your own paths by changing the necessary values in `configs/path_configs.py`.

Training

Preparing your Data

- Currently, we provide support for numerous datasets and experiments (encoding, frontalization, etc.).
  - Refer to `configs/paths_config.py` to define the necessary data paths and model paths for training and evaluation.
  - Refer to `configs/transforms_config.py` for the transforms defined for each dataset/experiment.
  - Finally, refer to `configs/data_configs.py` for the source/target data paths for the train and test sets as well as the transforms.
- If you wish to experiment with your own dataset, you can simply make the necessary adjustments in
  1. `data_configs.py` to define your data paths.
  2. `transforms_configs.py` to define your own data transforms.

As an example, assume we wish to run encoding using ffhq (`dataset_type=ffhq_encode`). We first go to `configs/paths_config.py` and define:

```
dataset_paths = {
    'ffhq': '/path/to/ffhq/images256x256'
    'celeba_test': '/path/to/CelebAMask-HQ/test_img',
}
```

The transforms for the experiment are defined in the class `EncodeTransforms` in `configs/transforms_config.py`.
Finally, in `configs/data_configs.py`, we define:

```
DATASETS = {
   'ffhq_encode': {
        'transforms': transforms_config.EncodeTransforms,
        'train_source_root': dataset_paths['ffhq'],
        'train_target_root': dataset_paths['ffhq'],
        'test_source_root': dataset_paths['celeba_test'],
        'test_target_root': dataset_paths['celeba_test'],
    },
}
```

When defining our datasets, we will take the values in the above dictionary.

Training Noise decoder pSp

The main training script can be found in `scripts/train.py`.
Intermediate training results are saved to `opts.exp_dir`. This includes checkpoints, train outputs, and test outputs.
Additionally, if you have tensorboard installed, you can visualize tensorboard logs in `opts.exp_dir/logs`.

```
python scripts/train.py \
    --dataset_type=celebs_noise_reduction \
    --exp_dir=exp_name_1 \
    --workers=4 \
    --batch_size=1 \
    --test_batch_size=1 \
    --test_workers=8 \
    --val_interval=7500 \
    --save_interval=10000 \
    --encoder_type=GradualStyleEncoder \
    --start_from_latent_avg \
    --lpips_lambda=0.8 \
    --l2_lambda=1 \
    --id_lambda=0.4 \
    --optim_name=adam \
    --device=cuda \
    --noise_strength=0.5
```

Additional Notes

- See `options/train_options.py` for all training-specific flags.
- See `options/test_options.py` for all test-specific flags.
- If you wish to resume from a specific checkpoint (e.g. a pretrained pSp model), you may do so using `--checkpoint_path`.
- If you wish to generate images from segmentation maps, please specify `--label_nc=N` and `--input_nc=N` where `N` is the number of semantic categories.
- Similarly, for generating images from sketches, please specify `--label_nc=1` and `--input_nc=1`.
- Specifying `--label_nc=0` (the default value), will directly use the RGB colors as input.

Testing

Inference

Having trained your model, you can use `scripts/inference.py` to apply the model on a set of images.
For example,

```
python3 scripts/inference.py \
    --exp_dir=five_five_noise_contd \
    --checkpoint_path=five_five_noise_contd/checkpoints/iteration_current.pt \
    --data_path=datasets/val \
    --test_batch_size=10 \
    --resize_outputs \
    --couple_output
```

Repository structure

Path	Description
pixel2style2pixel	Repository root folder
configs	Folder containing configs defining model/data paths and data transforms
criteria	Folder containing various loss criterias for training
datasets	Folder with various dataset objects and augmentations
environment	Folder containing Anaconda environment used in our experiments
models	Folder containting all the models and training objects
encoders	Folder containing our pSp encoder architecture implementation and ArcFace encoder implementation from TreB1eN
mtcnn	MTCNN implementation from TreB1eN
stylegan2	StyleGAN2 model from rosinality
psp.py	Implementation of our pSp framework
notebook	Folder with jupyter notebook containing pSp inference playground
options	Folder with training and test command-line options
scripts	Folder with running scripts for training and inference
training	Folder with main training logic and Ranger implementation from lessw2020
utils	Folder with various utility functions

Credits

StyleGAN2 implementation:
https://github.com/rosinality/stylegan2-pytorch
Copyright (c) 2019 Kim Seonghyeon
License (MIT) https://github.com/rosinality/stylegan2-pytorch/blob/master/LICENSE

MTCNN, IR-SE50, and ArcFace models and implementations:
https://github.com/TreB1eN/InsightFace_Pytorch
Copyright (c) 2018 TreB1eN
License (MIT) https://github.com/TreB1eN/InsightFace_Pytorch/blob/master/LICENSE

CurricularFace model and implementation:
https://github.com/HuangYG123/CurricularFace
Copyright (c) 2020 HuangYG123
License (MIT) https://github.com/HuangYG123/CurricularFace/blob/master/LICENSE

Ranger optimizer implementation:
https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer
License (Apache License 2.0) https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer/blob/master/LICENSE

LPIPS implementation:
https://github.com/S-aiueo32/lpips-pytorch
Copyright (c) 2020, Sou Uchida
License (BSD 2-Clause) https://github.com/S-aiueo32/lpips-pytorch/blob/master/LICENSE

Please Note: The CUDA files under the StyleGAN2 ops directory are made available under the Nvidia Source Code License-NC

Citation

If you use this code for your research, please cite our paper Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation:

```
@article{richardson2020encoding,
  title={Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation},
  author={Richardson, Elad and Alaluf, Yuval and Patashnik, Or and Nitzan, Yotam and Azar, Y
  journal={arXiv preprint arXiv:2008.00951},
  year={2020}
}
```