JOANNEUM RESEARCH DIGITAL

Institute for Information and Communication Technologies

MFA CycleGAN User Manual

MARS-DL/TRIP Projects

Graz, 22 June 2021

DOCUMENT DATA

Reference	Software-User-Manual-JR-Template-V1.0.docx
Projectnr.	
Document version	1.0
Status	**Draft**
Document date	22 June 2021
Keywords	software, manual, tutorial
Abstract	

Document Version	**Date**	**Name**	**Description**
V1.0	2021-06-05	Miguel Fernández	Initial version

Company / Organisation	**Name**	**Contact**
JOANNEUM RESEARCH	Kathrin Sander	Email: Miguel.Yuste.Fernandez.Alonso@joanneum.at Internet: http://www.joanneum.at/digital

TABLE OF CONTENTS

1 INTRODUCTION

This document is intended to serve as a manual to the neural style transfer systems set up in the context of the Mars-DL and TRIP projects. If you are unfamiliar with NST procedures, and more crucially, with the CycleGAN architecture, you can get a refresher here.

2 INSTALLATION

2.1 Prerequisites

- The operating system is not critical as long as it supports CUDA (and PyTorch)
- Python >= 3.7
- `conda` (`Anaconda` was used in the projects, `miniconda` works too)

2.2 Installation Process

The setup of the project includes the following:

- Clone the CycleGAN/Pix2Pix repository
- Clone the MARS-DL repository (private repo, ask Gerhard Paar or Daniel Kup for permissions)
    - The commit used in the project was `f13aab8148bd5f15b9eb47b690496df8dadbab0c`
- Use the "mfa_env_requirements.txt" file of the *cycleganutils* package to create a ready-to-use `conda` environment.
    - PyTorch will most likely need to be installed manually. Do a test run of the network, and if errors appear, uninstall PyTorch with `conda`.
    - The default installation command of PyTorch also installs the appropriate CUDA version.

All these steps have already been carried out on the DIGS203 machine (*E:\MFA* directory), which provides a ready environment for all tasks described in the present document.

2.3 Licensing

- All the listed external software is of free commercial use. You can find the CycleGAN/Pix2Pix license here

3 SOFTWARE OVERVIEW

The software consists of a generative adversarial neural architecture and a loose collection of scripts to usage of the network.

3.1 Software Purpose

3.1.1 CycleGAN/Pix2Pix repository

This repository is the official CycleGAN/Pix2Pix repository from the authors of both architectures.

3.1.1.1 CycleGAN

CycleGAN is a generative adversarial neural architecture for unpaired image style transfer: taking two input datasets A and B, it learns to make the images of A look like B, and viceversa. A good introduction to the architecture can be found here.

The core idea of the Mars-DL and TRIP projects was to use artificial images to train a conventional convolutional network for instance segmentation and object detection tasks. Naturally, such a network will not perform well in real-life tasks unless the synthetic training images are as realistic as possible. Unfortunately, this is not the case in most simulated datasets. The project teams thus extensively used the CycleGAN architecture in a preprocessing step to increase the realism of the synthetic training set.

For this, two datasets were always used: the synthetic dataset to adapt, and a real dataset containing the target style to copy. Choosing the second was always a sensitive step: the closest the real dataset is to the domain of the simulated dataset, the better the style conversion will work. Example: if we have a dataset of simulated images of kitchens, a real dataset containing rooms of all kinds might work decently, but it would be better to select only those images of kitchens.

Two final models were created after several training iterations:

- Martian model (training set image size: 1024x1024): it learned the appearance of classical rover imagery, and it's thus capable of applying the style of Mars landscapes. Originally, this was to be used with simulated rover images, a case in which it worked very well, but it could also be generally used to transform any kind of landscape into a Martian landscape. The greater the difference between the dataset and the Martian landscapes, the worse the network can be expected to perform.

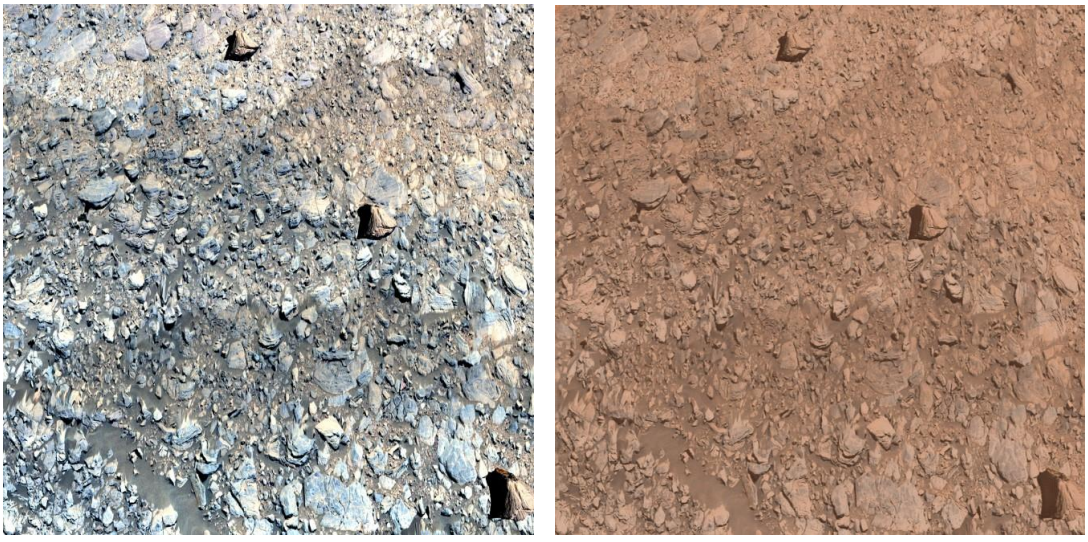

Figure 1: Left: Contrast-adapted screenshot before CycleGAN processing. Right: Contrast-adapted screenshot after CycleGAN processing

- Indoor spaces model (training set image size: 512x512): this model was trained on images of indoor spaces. Due to the specifity of the target domain, it did not perform as well as the CycleGAN model, but it is capable nonetheless of giving most surfaces a more realistic texture, as well as adapting the lighting and shading.

Figure 2: Left: Simulated indoor space before CycleGAN processing. Right: Simulated indoor space after CycleGAN processing

3.1.1.2 Pix2Pix

Another well-known style-transfer architecture, also included in the standard repository used in the project, Pix2Pix is however much less versatile than its evolved sibling CycleGAN, since it requires that the image pair be exactly aligned: images of the same scene, from the same perspective. This has applications in exact paired image translation; a good example would be the conversion of day pictures to night pictures.

No productive models were trained with Pix2Pix due to the limitations of the architecture.

3.1.2 Mars-DL repository: *cycleganutils* package

This repository contains a number of tools designed to aid the preparation of CycleGAN-ready datasets.

3.1.2.1 Dataset preparation utility

With the purpose of enabling the quick setup of training experiments, we developed a script to prepare two datasets in a training-ready folder structure. The script can generate appropriate training sets for both CycleGAN and Pix2Pix.

3.1.2.2 Batch histogram adaption script (located in ImproHistAdopt subfolder)

Making use of the existing Impro *HistAdopt* script, which adapts the histogram of a picture to that of a reference image, this script allows the efficient mass adaption of the individual histograms of a large

dataset using concurrent individual executions of the *HistAdopt* script. If a good picture example is available (appropriate colours, contrast), this script can be used to further increase the resemblance of the real training set to the desired conditions, (useful for example if the real dataset available is not completely adequate to the style transfer task).

add script and reference image to repo

3.1.2.3 LabelMars preparation script

A short script created to filter out all artificial pictures from the well-known *LabelMars* dataset.

3.1.3 Mars-DL repository: *pro3dutils* package/Random Pose Generator

Originally containing several utilities built around our project partner VRVis' *Pro3D* 3D rendering application, all of the contents of this repository have now been merged into a single script. Similar to Blender, *Pro3D* can render several types of tridimensional mesh files, allowing the user to navigate the environments and take snapshots of the views. Created with the purpose of enabling the mass collection of snapshots, the *Random Pose Generator* automatically creates a given amount of randomly generated snapshots of the shatter cones defined in the YAML config file centred at the camera centers (also specified in the YAML config) and writes out a *Pro3D*-ready JSON snapshot file; in this way, large synthetic image datasets can be quickly compiled using 3D meshes. The script was coded for an early version of *Pro3D* and it is no longer supported by the application; it would thus require adaption to function properly in later versions.

4 SOFTWARE USAGE

Note about the script paths of the commands listed below: these are set to the existing DIGS203 paths for a faster application (so that the user only needs to log into DIGS203 and can then directly copy paste the commands); in other machines, the paths would naturally need to be edited.

There are several common steps to both running and training a model:

If you are working from a powershell, you will need to switch to `cmd` to be able to run most commands:

```
cmd
```

1. We will also need to make sure all the GPUs are available for the CUDA jobs, which allows us to use higher batch sizes and speeding up the application of the models significantly.
   `set CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7`
2. We then proceed to activate the Mars-DL Python environment (change the environment name if you are not working with the preset environment of DIGS203):
   `conda activate E:\MFA\mfa`

4.1.1 Using the existing CycleGAN models

Applying the trained CycleGAN models is relatively straightforward. We will just need navigate to the CycleGAN/Pix2Pix directory (*E:\MFA\pytorch-CycleGAN-and-pix2pix* at DIGS203) and apply the model:

- Martian model:

  ```
  python test.py --dataroot {path to your data} --name mslmst_labelmars_cyclegan --model test --gpu_ids 0,1,2,3,4,5,6,7 --no_dropout --load_size {first load images with this size…} --crop_size {…then crop them to this size; see notes below} --results_dir {your desired output path}
  ```

- Indoor spaces model:
  To be added

Where the name parameter is not a directory but simply the name of the folder that contains the model in the *E:\MFA\pytorch-CycleGAN-and-pix2pix\checkpoints* directory.

Note about the model files: should the MFA folder not be available, you can also find the corresponding model files in the Mars-DL repository; you just need to copy the **folders** of the repo that contain the models to the *checkpoints* directory and then pass the name of the folder containing the desired model, as described in the previous paragraph.

Some remarks are relevant here:

- It is possible to apply a model trained on a given image size to bigger image sizes, but artifacts (distortions) may appear. The bigger the images are with respect to the original training set, the more distorted the images will appear.
- If the output should be of a different aspect ratio than that of the input pictures to be converted, it is recommended to set a `load_size` that keeps the aspect ratio of the input and then set the `crop_size` to the final desired resolution.
- If the results do not resemble the expected style, you might need to pass an additional parameter to the testing job inverting the conversion direction:

  ```
  --direction BtoA
  ```

Using the test CLI label allows us to use the selected GAN generator without the other 3 networks (the two discriminators and the opposite generator), which allows us to use much higher batch sizes. The recommended batch sizes to apply when using these models on the DIGS203 machine are the following:

- 256x256 and smaller images: 32.
- 512x512: 16.
- 1024x1024: 8.

Should there be memory allocation errors, simply reduce the batch size until the network fits in memory.

There is a wide array of testing options, which can be found in `test_options.py`, as well as `base_options.py`. The authors have also compiled a list of tips and tricks in this file.

4.1.2 Training new CycleGAN/Pix2Pix models

The process of training a new CycleGAN/Pix2Pix model is a bit longer but most of the preprocessing has been streamlined in the dataset preparation script. There are exhaustive documents in the CycleGAN/Pix2Pix repository, so if you have any questions, make sure to check those out.

4.1.2.1 Setup and dataset preparation

As an optional first step, and although not necessarily required, it is highly recommended to start our `visdom` server in order to be able to visualize the training progress:

```
start /B python -m visdom.server
```

Once it is up and running, we can navigate to http://localhost:8097 in our browser. However, bear in mind that this interface will not show anything until we start our training. It is also important to note that only one `visdom` instance can be active at a time, which means that, while it is possible to run several training jobs simultaneously (by making different CUDA devices visible to each `cmd` instance) you will not be able to visualize the progress but for one.

Before we combine our datasets, we can optionally use the *HistAdopt* script to adapt the histograms of our pictures to match that of a good example, as described in section 3.1.2.2. Important note: this needs to be run from the ImproHistAdopt folder.

```
cd "E:\MFA\mars_dl\cycleganutils\ImproHistAdopt"
```

```
python BatchAdaptHist.py -i {path to input folder} -r {path to reference image} -o {path to output folder}
```

Assuming we have our two datasets A and B clean and ready, we can then use the dataset preparation script to create our training set:

```
cd E:\MFA\mars_dl\cycleganutils
```

```
python PrepareDataset.py [cyclegan/pix2pix] {path to dataset A} {path to dataset B} {path to output folder} -r {resolution to use, 256 by default}
```

Where the resolution parameter is optional.

If we are running pix2pix, we need to run an extra command to merge our pictures (pix2pix works with a single image mosaic, containing both training pictures):

```
python datasets/combine_A_and_B.py --fold_A {path to the folder where you placed dataset A} --fold_B {path to the folder where you placed dataset B} --fold_AB {path to output folder}
```

4.1.2.2 Running the training job

And now we are finally ready to run the network, for which we will need to navigate to the *pytorch-CycleGAN-and-pix2pix* folder and then execute the following command.

```
cd "E:\MFA\pytorch-CycleGAN-and-pix2pix"
```

```
python train.py --dataroot {path to your prepared dataset} –name {name of your model, which will be placed under the corresponding subfolder in the pytorch-CycleGAN-and-pix2pix/checkpoints folder} --model [cycle_gan/pix2pix] --gpu_ids 0,1,3,4,5,6,7 --batch_size={batch size}
```

Unlike the test script, the training logically requires to load the full CycleGAN in memory (that is, four networks – two generators and two discriminators), which is extremely heavy and significantly reduces

the batch size we can use. The recommended batch sizes to apply when using these models on the DIGS203 machine are the following:

- 256x256 and smaller images: 16 or 8.
- 512x512: 2 or 1
- 1024x1024: 1. If even 1 is too much, see the method of the *Training/Testing with high res images* section of the author's tips and tricks. Bear in mind however that, while this method indeed makes it possible to train with high resolutions, it can sometimes lead to blurry results.

4.1.2.3 Using the results

Note about CycleGAN models: once the training is done, an important step remains in order to be able to use the resuls. The test script will look for a file called `latest_net_G.pth`, which is not created during training, and if it is missing an error will be thrown. Instead, a separate checkpoint file is generated for each generator: `latest_net_G_A.pth` and `latest_net_G_B.pth`. What we now need to do is copy the one corresponding to the network we want, paste it, and rename it to `latest_net_G.pth`. Example: I have two datasets: dataset A is the simulated dataset, and dataset B is the real dataset whose style we want to copy; once the training is done, I will copy-paste the checkpoint file of the B generator (`latest_net_G_B.pth`) and rename it to `latest_net_G.pth`. This does not need to be done for the two previously mentioned trained models: the correct network is already set to `latest_net_G.pth`.

4.1.2.4 CycleGAN tips and tricks

Here go some pieces of advice when training CycleGAN:

- You can stop a training and resume it later with the following flags:
  `--continue_train --epoch_count={epoch from which you want to continue training - this is only relevant for the labelling of the checkpoint files, the network will already load the latest model by default, so if this parameter is not set the only thing that will happen is that the checkpoints will be again labelled starting from 1} --n_epochs={how many epochs you still want to train for}`
  This is also useful if the training crashes, which will likely happen if other GPU-intensive applications are used while the training is running.
- The `--gan_mode` parameter is crucial, since it sets the GAN objective. The Wasserstein GAN objective (`wgangp` option) can be expected to perform best in most applications (especially in regards to the stability), followed by the Least Squares GAN (`lsgan` option), which is the default of the application. In any case, it is highly recommended to make experiments with both.
- Although it can be quite time-intensive, thorough dataset cleaning is crucial when training CycleGAN models. As an illustrative example, in our initial Martian trainings we left a couple of takes where some of the rover instruments where visible, and as a result the GAN tried to paint rover arms in the simulated landscapes. Should strange/unexplainable artifacts appear in the training results, this is the first candidate to check.
- Training GANs can be frustrating: they are universally known to be challenging and unstable networks to work with. Although CycleGAN is more functional than most GAN architectures, it still is quite likely that the first training attempts will not work, and sometimes repeating the training with the exact same parameters can achieve vastly superior results. Should the errors not disappear after a couple of trainings, here you can find a short list of the most common problems and ways to tackle them. This link provides a more in-depth guide to GAN failure cases and possible solutions.

Additionally, and as is the case with the testing of models, many different training options are available, which can be found in `train_options.py`, as well as `base_options.py`. The authors have also compiled a list of tips and tricks in this file.

JOANNEUM RESEARCH
Forschungsgesellschaft mbH

Leonhardstraße 59
8010 Graz

Tel. +43 316 876-0
Fax +43 316 876-1181

pr@joanneum.at
www.joanneum.at

oef mfg 12008