Technical University of Košice
Faculty of Electrical Engineering and Informatics

Generative adversarial network for augmenting insufficient medical datasets

User Guide

2024 Bc. Miroslav Plavec

Contents

1 Introduction **4**

2 Setting up your environment **4**

3 Training a specific GAN **5**

List of Figures

3 – 1 Training parameters 5
3 – 2 Folder where all experiments are saved 6
3 – 3 Structure of specific experiment‘s folder 7

1 Introduction

This user manual will guide you through running experiments with our pre-built Generative Adversarial Networks (GANs) codebase to generate images using your own custom dataset. While we offer various GANs architectures within the codebase, all experiments follow a consistent structure. Therefore, we'll describe the process for a single experiment in detail, and the instructions will be applicable to all the available GANs. First, we will describe the requirements for running the scripts. Then, we will detail how to train a specific GAN model and explain the parameters you can adjust.

2 Setting up your environment

All experiments were written in Python. To run the experiments on your own data, you'll need Python version 3.10 or higher. The required libraries are listed in a text file named requirements.txt. To install them, run the following command in your terminal: pip install -r requirements.txt. After installation, you can proceed with running experiments on your own data.

The folders containing the code are organized as follows:

FID: This folder contains the MyFID class, which computes the FID score between real and generated images.

DCGAN: This folder contains scripts for unconditional image generation using DCGAN.

WGAN-Unconditional: This folder contains scripts for unconditional image generation using WGAN-GP.

WGAN-EncoderDecoder: This folder contains scripts for conditional image generation with WGAN-GP, where the generator's input is random noise and a binary mask.

WGAN-SkipConnections: This folder contains scripts for conditional image generation with WGAN-GP, where skip connections are used in the Encoder-Decoder architecture.

Pix2pix: This folder contains scripts for conditional image generation using Pix2Pix.

All folders follow the same structure. The file named main.py initiates the training process for a specific GAN model on your data. Execute this file to train the model.

3 Training a specific GAN

As mentioned earlier, to run experiments on your own images, you simply need to choose a folder and execute the main.py script within the desired experiment directory. The main.py script allows you to modify various parameters that influence the training process. While changing these parameters is optional, they can be fine-tuned to potentially improve the results for your specific data. FIXED_NOISES holds the same random noises that are used during training to generate 8 images. These images allow you to visualize the progress of image generation. The following Figure 3 – 1 illustrates the configurable parameters within the main.py script.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
BATCH_SIZE = 16
EPOCHS = 300
Z_DIM = 500
LEARNING_RATE = 1e-4
FIXED_NOISE = torch.randn(8, Z_DIM, 1, 1).to(device)
CSV_PATH = "PATH_TO_CSV"
```

Figure 3 – 1 Training parameters

One crucial parameter you must modify is **CSV_PATH**. This parameter specifies the path to a CSV file containing information about your training images. The experiment relies on this file to sample image paths during training. The structure of the CSV file depends on whether you're performing unconditional or conditional

image generation.

Unconditional Generation: The CSV file should contain a single column listing the paths to your training images. Each row represents a single image.

Conditional Generation: The CSV file should have two columns. The first column lists the paths to your training images, and the second column specifies the corresponding segmentation mask paths. Each row represents an image-mask pair.

After setting up the path to the CSV file and other hyperparameters, the model starts training for a specified number of epochs. Each experiment you run is automatically saved in its own folder named "experiments" with a unique identifier "experiment_specificID". After every 10 epochs, the script prints a message to the terminal indicating the current epoch number. Also each 10 epochs, the generator creates 8 images, which are saved in the experiment folder. Additionally, the generator and discriminator parameters are saved. Once the training process is complete, the script calculates the final FID score, which measures the similarity between the generated images and the real images. This score is then printed to the terminal, evaluating the model's ability to generate realistic images. Additionally, a description of the specific experiment is saved to a CSV file named "experimentInfo.csv".

Figure 3 – 2 Folder where all experiments are saved

Figure 3 – 2 illustrates how the "experiments" folder will look after you run some experiments. Each experiment is saved in its own folder with a unique identifier.

Figure 3 – 3 Structure of specific experiment‘s folder

Figure 3 – 3 illustrates the structure of an individual experiment folder within the ”experiments” directory. Each unique experiment folder contains a subfolder where images generated during training are saved. These images serve as a visual record of the generator’s learning process. By observing the progression of these images, you can witness how the generator gradually improves its ability to create realistic images as the training progresses. Additionally, the experiment folder stores the final parameters of both the generator and discriminator models.