Technical University of Košice
Faculty of Electrical Engineering and Informatics

Generative adversarial network for augmenting insufficient medical datasets

System Guide

2024 Bc. Miroslav Plavec

Contents

List of Figures

1 Introduction

In our system guide, we will provide a more detailed explanation of the functionalities within the custom scripts we have written. Since all experiments follow the same structure, we will describe only one experiment in detail. This approach will first outline the overall structure of our experiments and then delve into each important script.

2 Folder organization

All scripts can be found in the folder named ”Code”. The image below shows the structure of this folder. This folder contains six subfolders and one file named ”requirements.txt”. The text file ”requirements.txt” lists all libraries necessary to run your experiments. Subfolders contain scripts for specific GAN.

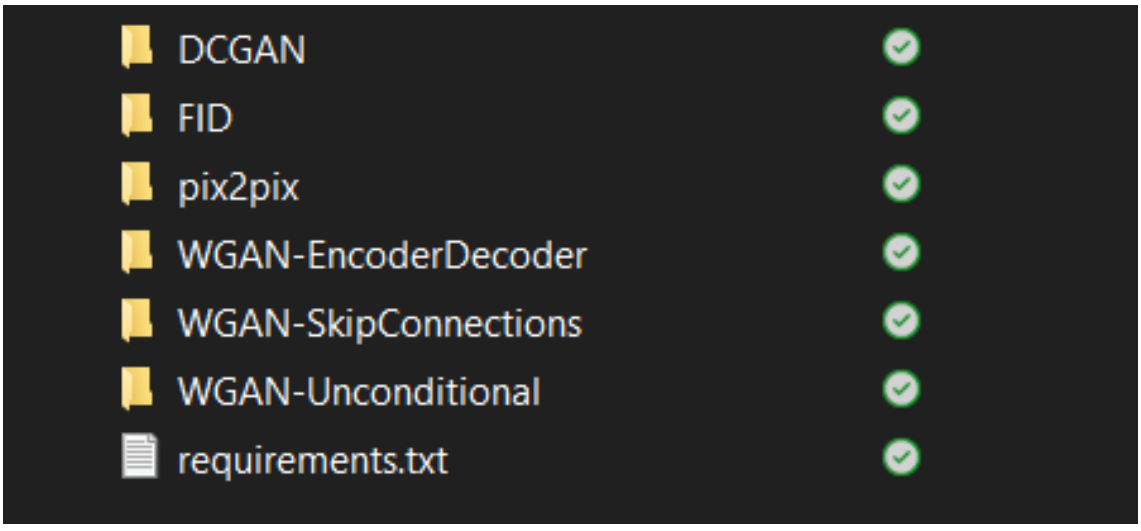

Figure 2 – 1 Folder structure

3 FID

In the folder named ”FID” is script for computing the FID score between generated and real images. These script utilize several functions.

normalize_batch: This function takes a batch of images (real or fake) and normalizes their pixel values to the range [0, 1].

gray_to_rgb - This function takes batches of real and fake images and transforms them from grayscale to RGB format. This is necessary because the torchmetrics function used to compute FID requires RGB images as input. If you're not working with grayscale images, you can safely delete the line where this function is called.

get_distance - This function takes batches of real and fake images as input. It first calls **normalize_batch** on both sets of images, then transforms them to RGB format if necessary (using **gray_to_rgb**). Finally, it utilizes a torchmetrics function to compute the FID score and returns the result.

4 Specific GAN

In our thesis, we experimented with several GAN types to generate images of A-lines and B-lines. Each subfolder within the "Code" folder (except for "FID") represents a specific GAN that utilizes a different architecture or loss function. To ensure consistency, we have structured each GAN folder to contain the same scripts with similar functionalities. While we will only describe one GAN in detail here, this description applies to all the other GANs as well. The image below shows the structure of specific folder for GAN.

Figure 4 – 1 Experiment structure

4.1 Dataset.py

This script defines a class named LungsDataset whose purpose is to load, transform, and return training images. The class contains several functions for data manipulation:

__**init**__ (constructor): This function takes the path to a CSV file as input and reads all image paths from the CSV into a pandas DataFrame.

__**len**__: This function returns the length of the training dataset (the total number of training images).

transform: This function takes an image and its corresponding mask as input and applies certain transformations. It first resizes both the image and mask to a size of 512x256. Next, it normalizes pixel values to the range [-1, 1] and, with a probability of 0.5, performs a horizontal flip on both the image and mask. This horizontal flip is used for data augmentation to slightly increase the size and diversity of the training data. The function returns the transformed image and mask. If you are using an unconditional experiment (without masks), the input and output will just be the image itself.

__**getitem**__: This function takes an index from the dataloader and returns the corresponding image-mask pair. Based on the index, it first finds the path to the image and mask in the pandas DataFrame and load them. Then, it calls the **transform** function on both the image and mask. If you are using the unconditional approach, only the image is used.

getEvalMask (conditional only): This function takes an integer parameter n that specifies how many masks you want to use for evaluation during training. It randomly selects n masks from the dataset, and these masks are then excluded from the training process. The purpose of these evaluation masks is to monitor the progress of GAN training and observe how the generated images change throughout the training process. This function is only relevant for the conditional approach where masks

are used.

4.2 Utils.py

This script contains several helper functions:

seed_everything: This function initializes the pseudo-random number generators in NumPy and PyTorch libraries. This ensures reproducibility of your training results, meaning that if you run the experiment with the same seed value, you should get the same results (assuming no external randomness is introduced).

initialize_weights: This function takes the weights of a generator or discriminator model and initializes them with specific values before training starts.

gradient_penalty: This function is specifically used for experiments that employ the WGAN-GP (Wasserstein Generative Adversarial Network with Gradient Penalty) loss function. It calculates the gradient penalty, which is a term added to the loss function to enforce a 1-Lipschitz constraint on the critic network. This constraint helps to stabilize the training process of WGAN-GP.

4.3 Generator.py

This script defines a class named Generator whose purpose is to generate new images. The class contains several functions:

__init__ (constructor): This function takes two arguments: the number of channels in the input image and the desired feature counts for the convolutional layers. Its purpose is to initialize the model architecture, defining the specific types and order of the layers used.

_down: This function takes several arguments like input channels, output channels, kernel size, stride, and padding. It returns a block that consists of a convolutional layer with a specific normalization function and activation function.

_up: Similar to **_down**, this function takes input and output channels, kernel size, stride, and padding. It returns a block containing a transposed convolutional layer with a chosen normalization and activation function.

forward: This function takes the input to the generator (either noise or noise combined with a mask) and performs a forward pass through the neural network, ultimately computing the generated image.

4.4 Discriminator.py

This script defines a class named Discriminator whose purpose is to classify an input image as real or generated. The class contains several functions:

__init__ (constructor): This function takes two arguments: the number of channels in the input image and the desired feature counts for the convolutional layers. Its purpose is to initialize the model architecture, defining the specific types and order of the layers used.

_down: Similar to the Generator class, this function takes arguments like input channels, output channels, kernel size, stride, and padding. It returns a block that consists of a convolutional layer with a specific normalization function and activation function.

forward: This function takes the input to the discriminator (an image) and performs a forward pass through the neural network. The output of this function indicates the probability of the input image being real or generated.

4.5 main.py

This script train a specific GAN using your own data. It uses various functions to manage the training process and evaluation.

saveModel: This function takes the parameters of the generator, discriminator,

and optimizer as input and saves them to designated folders for future use.

saveImages: This function generates images for monitoring training progress. It uses either fixed noise or a combination of fixed noise and fixed masks, depending on whether you're using a conditional or unconditional approach. The generated images are then saved to a designated folder.

fidEval: This function takes the generator as input and generates the same number of images as the size of your training dataset. It then utilizes the previously described FID class to compute the FID score for evaluation.

saveToCsv: This function gathers information about the experiment, such as batch size and number of epochs, and saves it to a CSV file for later analysis.

trainModel: This is the core function responsible for training the model. It starts by creating a folder to store generated images, network parameters, and other relevant data. Then, it initializes the generator, discriminator, and optimizer. Following initialization, it starts training for a specified number of epochs. Every 10 epochs, the function calls **saveModel** to save the networks parameters and **saveImages** to generate and save images for progress visualization. After training completion, **fidEval** is called to calculate the FID score, and all experiment information is saved using **saveToCsv**.

Before calling the trainModel function, you can set up several training parameters like batch size and number of epochs. These parameters may vary depending on the specific type of GAN you choose to use. The provided image 4 – 2 shows an example of these parameters. The only element you need to modify is **CSV_PATH**, which should point to the CSV file containing the paths to your training data. FIXED_MASKS and FIXED_NOISE hold the same masks and noise which are used during a training to capture progress of generated images.

```
BATCH_SIZE = 16
EPOCHS = 1
Z_DIM = 500
LAMBDA_GP = 10
LEARNING_RATE = 1e-4
CRITTIC_ITERATIONS = 5
CSV_PATH = "PATH_TO_CSV"
FIXED_MASKS = dataset.getEvalMask(4).to(device)
FIXED_NOISE = torch.randn(4, Z_DIM, 1, 1).to(device)
```

Figure 4 – 2 Parameters