Functional specification

Ganics

Adrian Chmiel

4th June 2024

1 Project's goal

The aim of the project is to utilize GAN networks for generating images with translated style. The program offers two possible style conversion modes:

1. **Vincent Van Gogh Art** - images will resemble the artworks of Vincent Van Gogh
2. **Cartoon** - images will resemble pictures from cartoons (or comics)

2 General information

The program was written in Python 3.10. It uses the following libraries:

- **matplotlib** - for creating plots and visualizing data
- **numpy** - for matrix operations
- **opencv-python** - for image processing
- **pandas** - for data analysis
- **pillow** - for image processing
- **pydot** - for visualizing model structure
- **tensorflow** - for creating and training GAN models
- **tkinter** - for creating a simple graphical interface
- **tqdm** - for displaying progress in model training

3 Launching the program

The program can be launched in several ways. One of the options is to use the available ***ganics.exe*** file. In case we would like to run the application directly from the source code, it is recommended to create a new virtual environment and install the required libraries in it. Assuming that we are using *conda*, we can do this with the following commands:

```
conda create -n Ganics python=3.10 -y
conda activate Ganics
conda install jupyter
pip install -r requirements.txt
```

The installation of Jupyter can be skipped if we do not plan to use notebooks.

For NVIDIA graphics card owners, it is also worth running the following command, which will enable the use of CUDA:

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0 -y
```

The program can be launched by calling the ***main.py*** file:

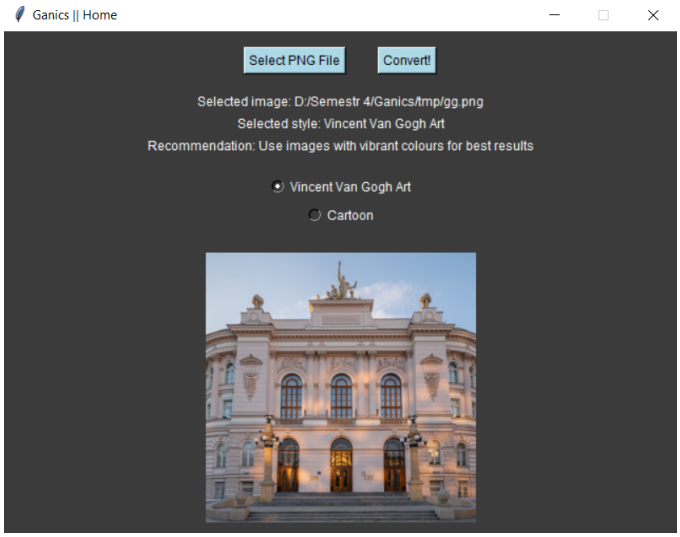
```
python src/main.py
```

4 Graphical interface

The graphical interface of the program was created using the *tkinter* library. It allows:

- Selection of the input file
- Selection of the conversion style
- Converting and saving the image
- Preview of the image before and after conversion

A sample screenshot of the graphical interface is shown below:



5 Input and output files

1. **Main application**
   The program allows selecting the path to the input file that will be converted. It is worth noting that the input file must be an image in the *.jpg*, *.jpeg*, or *.png* format.

   The output file, regardless of the initial format, is always an image in the *.png* format with a resolution of *512x512*.

2. **Notebooks**
   When it comes to notebooks, we provide entire sets of images as input (more information below), which will be used to train the model. As output, we receive visualizations illustrating how the model performs after training and an exported model weights file, which is later used as a component of the main application.

6 How the application works

The application's operation concept is visible below:

1. File selection and conversion style
2. Image style translation using **CycleGAN**
   The resulting image has a resolution of 256x256.
3. Image upscaling using **SRGAN** to a resolution of 512x512
4. Saving the image in the *.png* format

Important fact is that the application **will not work** if it does not find the appropriate model weights, which the user will be informed about by an error. In such a case, it is necessary to either run the notebooks to train the model, or download the ready-made files available on *Google Drive* under the link in the *how_to_get_models.md* file.

7 Theory

The program is based on two GAN network models:

- **CycleGAN**
- **SRGAN** (with *PatchGAN* discriminator)

GAN *(Generative Adversarial Network)* is a type of artificial neural network that consists of two models: **generator** and **discriminator**. The generator's task is to generate new images that are similar to those from the dataset, while the discriminator's task is to distinguish whether the image comes from the generator or the dataset. The training process involves a competition between both models, where the generator tries to deceive the discriminator, and the discriminator tries to learn to distinguish images. As a result, the generator is able to generate images very similar to those from the dataset.

More information about the operation of these models is available in the document: *Note_ about_ GANs.pdf*

8 Datasets

To obtain datasets for training models, you can extract the available *.zip* files, which contain images selected by me from the datasets listed below. An alternative option is to use the links to these datasets available in the *README.md* file. This approach will allow you to independently choose images that will be used for training.

The datasets used for training models are:

- **Familyguy** (from the Cartoon Classification dataset)
  Used to train CycleGAN responsible for converting the style to a cartoon (or comic).
- **VincentVanGogh** (from the Vincent Van Gogh Art dataset)
  Used to train another CycleGAN responsible for converting the style to Vincent Van Gogh's artworks.
- **natural_images**
  Sample images, the style of which was transformed into one of the two mentioned above during training.
- **mscoco**
  A dataset that additionally expanded the set of data with natural images used to train the models.

9 Limitations

Style conversion is a relatively complex task, so the training time in notebooks strongly depends on the computational power of the computer. For this reason, it is recommended to use an NVIDIA graphics card, which significantly speeds up the learning process. All notebooks are adapted to use CUDA if available, that is the required software has been installed.

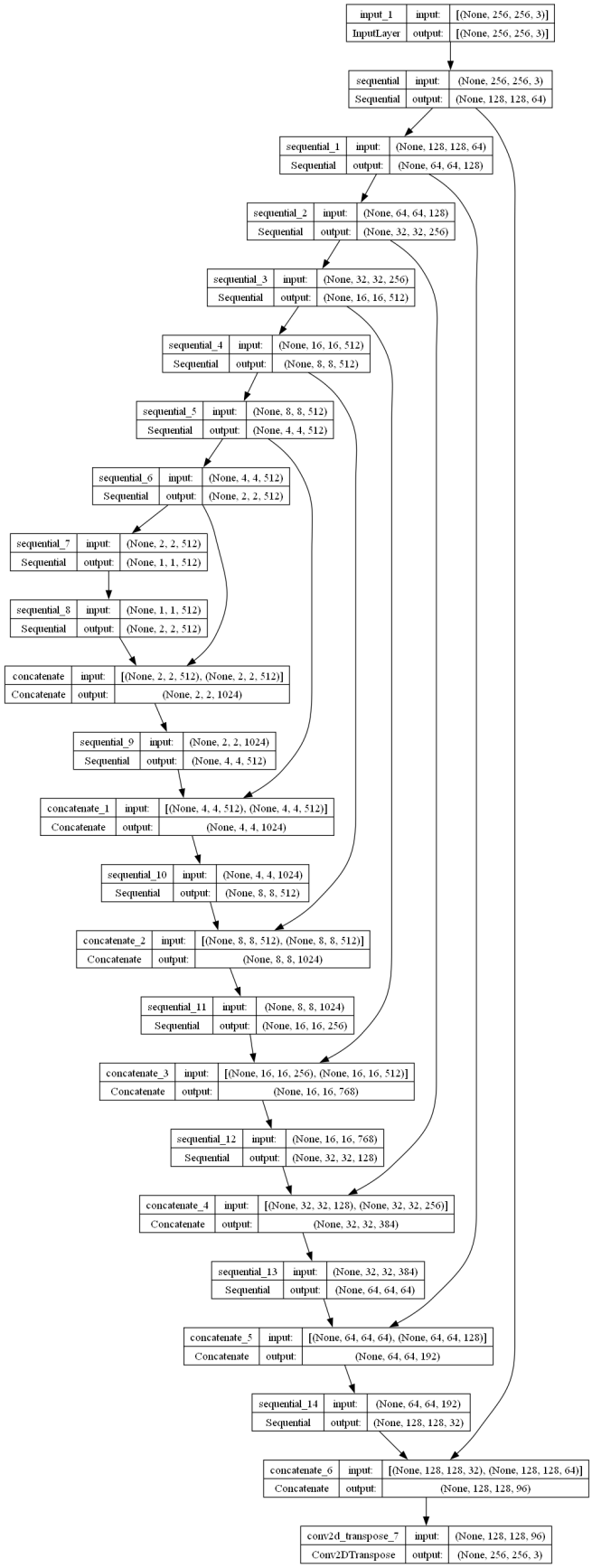
It is also worth noting that the cartoon style is relatively simple, so the model will cut off the details of the images during conversion. In the case of too complex images, the result may be unsatisfactory.

10 Visualization of featured models

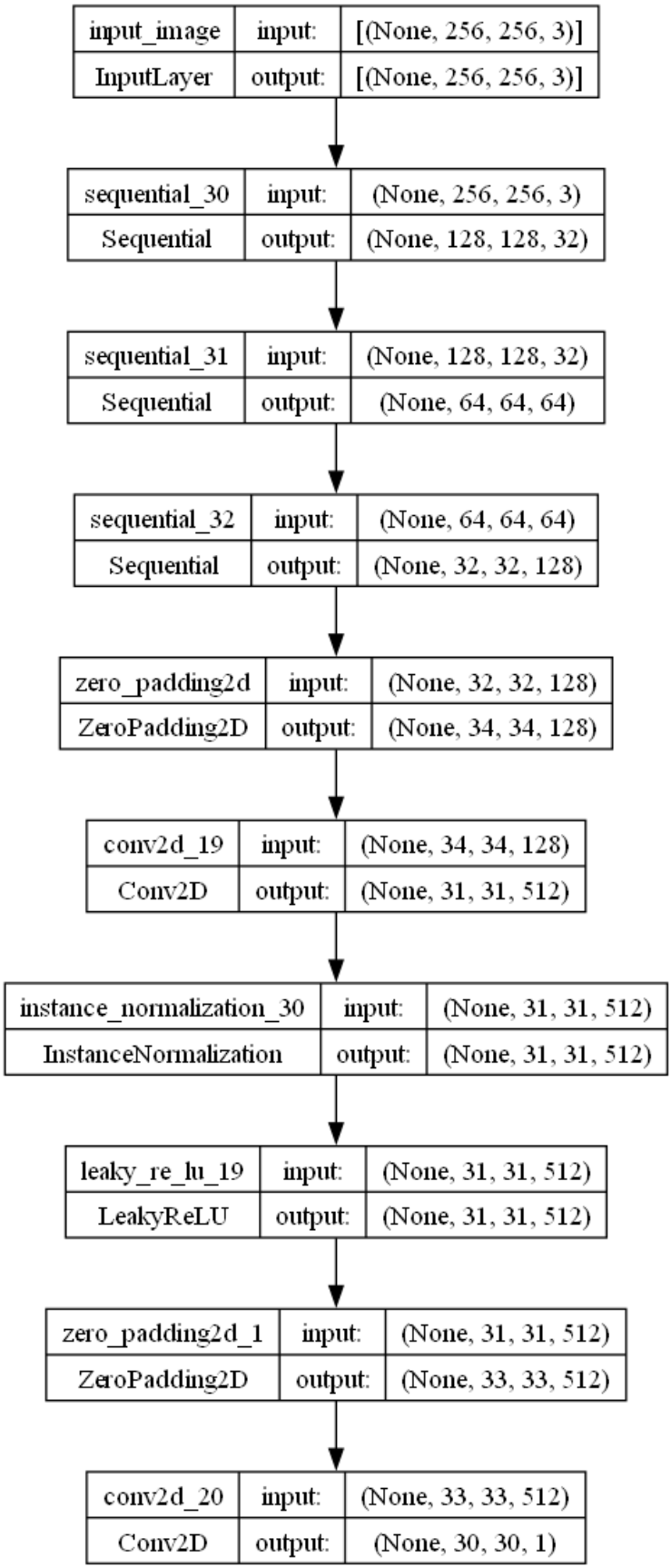
On the following pages, visualizations of the models used in the application are visible. These visualizations were generated using the *pydot* library. They show both the generator and the discriminator for both models, along with individual layers and their connections. More information is available in the documents:

- *Note_ about_ GANs.pdf*
- *Note_ about_ layers.pdf*

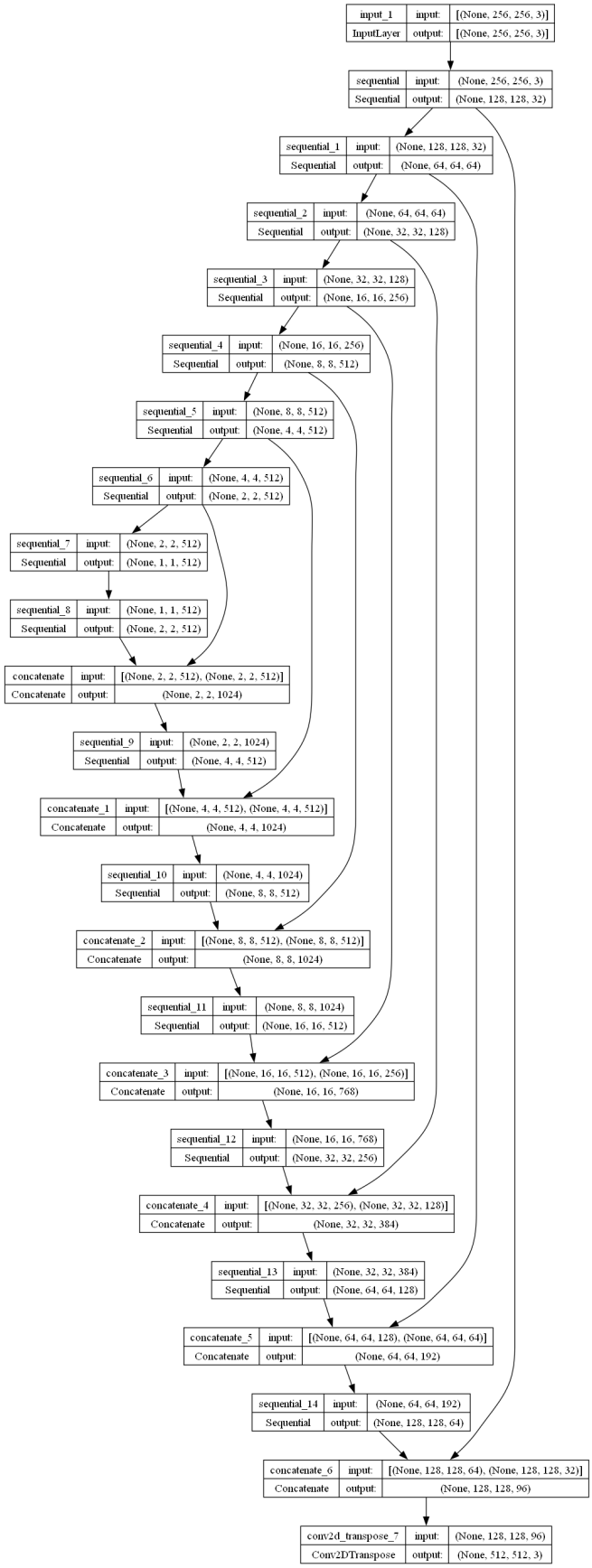
1. CycleGAN - Generator



2. **CycleGAN - Discriminator**



3. SRGAN - Generator



4. **SRGAN - Discriminator *(PatchGAN)***