ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

BÁO CÁO ĐỒ ÁN

Xây Dựng Mô Hình Chuyển Đổi Ảnh Selfie Thành Anime Style Sử Dụng CycleGAN

GIẢNG VIÊN HƯỚNG DẪN
NGUYỄN TRỊNH ĐÔNG

MSSV	HỌ VÀ TÊN
22521406	HUỲNH HỮU THỊNH
22520908	ĐOÀN PHƯƠNG NAM

Mục lục

1. Giới thiệu đề tài

1.1. Lý do chọn đề tài

Trong những năm gần đây, trí tuệ nhân tạo (AI) đã tạo ra bước đột phá trong lĩnh vực xử lý hình ảnh, đặc biệt là các mô hình chuyển đổi phong cách ảnh (style transfer). Trong đó, việc chuyển đổi ảnh chân dung thực tế (selfie) sang phong cách anime là một ứng dụng thu hút sự quan tâm lớn từ cộng đồng nghệ thuật số, game, và mạng xã hội. Đề tài này tập trung xây dựng mô hình **chuyển đổi ảnh selfie thành anime style** bằng kiến trúc **CycleGAN** (Cycle-Consistent Generative Adversarial Network), một phương pháp tiên tiến trong lĩnh vực chuyển đổi domain ảnh không cặp đôi (unpaired image-to-image translation).

Phong cách anime mang tính nghệ thuật cao và có nhu cầu ứng dụng rộng rãi, nhưng việc chuyển đổi thủ công đòi hỏi nhiều thời gian và kỹ năng. CycleGAN giải quyết bài toán này bằng cách tự động hóa quá trình chuyển đổi mà không yêu cầu dữ liệu cặp đôi (paired data), giúp tận dụng nguồn ảnh selfie và anime riêng biệt sẵn có. Đây là ưu điểm vượt trội so với các mô hình GAN truyền thống, đồng thời đảm bảo tính nhất quán thông qua cơ chế cycle consistency loss.

1.2. Mục tiêu và phương pháp tiếp cận

Mục tiêu chính của đồ án là xây dựng mô hình có khả năng:

1. **Học đặc trưng** của ảnh selfie (domain X) và anime (domain Y) từ hai tập dữ liệu không liên quan.
2. **Chuyển đổi** ảnh selfie sang phong cách anime mà vẫn giữ nguyên bố cục và chi tiết quan trọng.
3. **Tái tạo** ảnh gốc từ ảnh đã chuyển đổi để kiểm tra tính nhất quán.

CycleGAN được lựa chọn nhờ hai cặp **Generator** (G_XtoY, G_YtoX) và **Discriminator** (D_X, D_Y). Các generator sử dụng **Residual Blocks** để học phép biến đổi phức tạp, trong khi discriminator áp dụng tích chập (convolution) để đánh giá tính chân thực. Hàm loss bao gồm **adversarial loss** (đảm bảo ảnh đầu ra giống domain

đích) và **cycle consistency loss** (giảm thiểu sai lệch khi chuyển đổi qua lại giữa hai domain).

1.3. Dữ liệu và quy trình huấn luyện:

- **Dữ liệu đầu vào:** Hai tập ảnh riêng biệt: ảnh selfie (trainA/testA) và ảnh anime (trainB/testB).
- **Tiền xử lý:** Resize ảnh về kích thước 255x255, chuẩn hóa giá trị pixel về khoảng [-1, 1].
- **Huấn luyện:** Mô hình được huấn luyện trên 100 epoch với batch size 32, learning rate 0.0002, sử dụng optimizer Adam. Quá trình đồng thời cập nhật generator và discriminator để cân bằng giữa việc tạo ảnh chất lượng và phân loại chính xác.

1.4. Kết quả và ứng dụng:

Sau huấn luyện, mô hình thành công trong việc tạo ra ảnh anime từ ảnh selfie với các đặc trưng như đường nét cách điệu, màu sắc tương phản cao, đồng thời bảo toàn nội dung gốc. Ứng dụng tiềm năng bao gồm:

- Công cụ chỉnh sửa ảnh tự động cho ứng dụng di động.
- Hỗ trợ sáng tạo nội dung số trong game và phim hoạt hình.
- Nghiên cứu nâng cao về chuyển đổi đa domain trong thị giác máy tính.

1.5. Đóng góp của đồ án:

- Triển khai thành công CycleGAN bằng PyTorch, tối ưu hóa kiến trúc mạng và hyperparameter.
- Cung cấp bộ mã nguồn mở đầy đủ, dễ dàng tích hợp vào các dự án thực tế.
- Minh họa trực quan kết quả thông qua ảnh đầu vào/đầu ra và biểu đồ loss.

Với hướng phát triển trong tương lai như tăng độ phân giải ảnh, cải thiện chất lượng chi tiết, đề tài mở ra nhiều cơ hội ứng dụng thực tiễn, kết hợp giữa nghệ thuật và công nghệ AI.

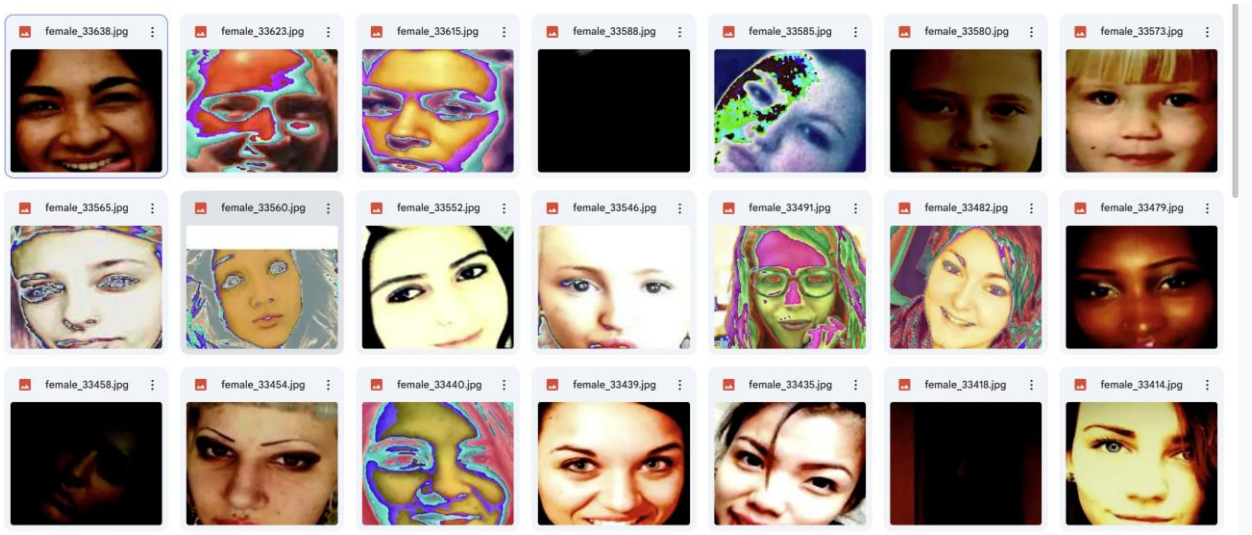
2. Quá trình tiền xử lý dữ liệu

2.1. Thu thập dữ liệu

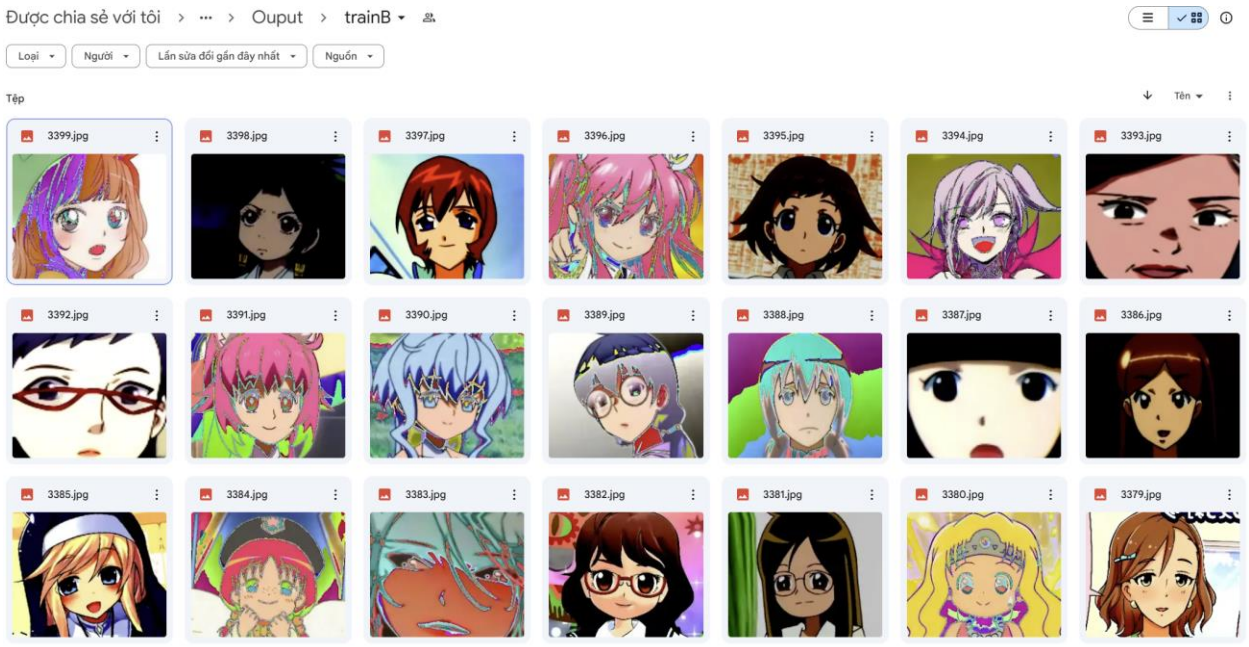
Nguồn : https://www.kaggle.com/datasets/arnaud58/selfie2anime

Dataset gồm có :

- 3400 khuôn mặt phụ nữ trong trainA



- 3400 khuôn mặt anime trong trainB



- 100 khuôn mặt phụ nữ trong testA
- 100 khuôn mặt anime trong testB

Code cài đặt:

2.1.1. Import các thư viện sử dụng:

```
[13] import os
     import cv2
     import numpy as np
     from PIL import Image
     from mtcnn import MTCNN
     from albumentations import (HorizontalFlip, Rotate, RandomBrightnessContrast, GaussianBlur, Compose)
     from tqdm import tqdm
     from multiprocessing import Pool
```

- **os**: Tương tác với hệ điều hành (ví dụ: tạo thư mục, đọc file).
- **cv2**: Xử lý ảnh và video (ví dụ: đọc, ghi, chỉnh sửa ảnh).
- **numpy**: Tính toán số học, đặc biệt là với mảng (array) dữ liệu.
- **PIL**: Xử lý ảnh nâng cao (ví dụ: mở, chỉnh sửa, lưu ảnh).
- **mtcnn**: Nhận diện khuôn mặt trong ảnh.
- **albumentations**: Tạo biến thể ảnh để tăng cường dữ liệu huấn luyện.
- **tqdm**: Hiển thị thanh tiến trình khi xử lý dữ liệu.
- **multiprocessing**: Xử lý song song để tăng tốc độ chương trình.

2.1.2. Cài đặt trước cho các thuật toán

```
#-----setting
input_dirs={
    'trainA':'/content/drive/MyDrive/SE355/Đồ án/Datasets/trainA',
    'trainB':'/content/drive/MyDrive/SE355/Đồ án/Datasets/trainB',
    'testA' : '/content/drive/MyDrive/SE355/Đồ án/Datasets/testA',
    'testB': '/content/drive/MyDrive/SE355/Đồ án/Datasets/testB'

}

ouput_dirs={
    'trainA':'/content/drive/MyDrive/SE355/Đồ án/Datasets/Ouput/trainA',
    'trainB':'/content/drive/MyDrive/SE355/Đồ án/Datasets/Ouput/trainB',
    'testA' : '/content/drive/MyDrive/SE355/Đồ án/Datasets/Ouput/testA',
    'testB': '/content/drive/MyDrive/SE355/Đồ án/Datasets/Ouput/testB'
}
resize_size = (256, 256)
pixel_range = [-1, 1]
augmentations = Compose([
    HorizontalFlip(p=0.5),
    Rotate(limit=10, p=0.5),
    RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    GaussianBlur(blur_limit=(3, 3), p=0.3)
])
```

1. **Định nghĩa đường dẫn:**
    - input_dirs: Lưu trữ đường dẫn đến các thư mục chứa ảnh gốc, được chia thành các tập trainA, trainB, testA, testB.
    - ouput_dirs: Lưu trữ đường dẫn đến các thư mục sẽ chứa ảnh sau khi xử lý, tương ứng với các tập trong input_dirs.
2. **Thiết lập kích thước và chuẩn hóa:**
    - resize_size: Tất cả ảnh sẽ được resize về kích thước 256x256 pixel.
    - pixel_range: Giá trị pixel của ảnh sẽ được chuẩn hóa về khoảng [-1, 1].
3. **Tăng cường dữ liệu (augmentation):**
    - augmentations: Định nghĩa một chuỗi các phép biến đổi ngẫu nhiên áp dụng lên ảnh, bao gồm lật ngang, xoay, thay đổi độ sáng/tương phản, làm mờ. Mục đích là tạo ra nhiều dữ liệu hơn cho việc huấn luyện mô hình.

2.2. Xử lý dữ liệu

2.2.1. Đảm bảo thư mục đích tồn tại

```
def ensure_dir_exists(dir_path):
    if not os.path.exists(dir_path):
        os.makedirs(dir_path)
```

- Hàm này đảm bảo rằng thư mục được chỉ định bởi dir_path tồn tại.
- Nếu thư mục chưa tồn tại, hàm sẽ tạo ra thư mục đó.
- Điều này rất hữu ích để đảm bảo rằng chương trình có thể lưu trữ kết quả xử lý ảnh mà không gặp lỗi.

2.2.2. Chuẩn hóa ảnh

```
def normalize_pixels(image):
    image = image / 255.0  # Chuyển về phạm vi [0, 1]
    if pixel_range == [-1, 1]:
        image = image * 2 - 1  # Chuyển về phạm vi [-1, 1]
    return image
```

- Hàm này chuẩn hóa giá trị pixel của ảnh.
- Đầu tiên, nó chia tất cả giá trị pixel cho 255.0 để đưa chúng về phạm vi [0, 1].
- Sau đó, nếu pixel_range được đặt là [-1, 1] (như trong đoạn code chính), hàm sẽ chuyển đổi giá trị pixel về phạm vi [-1, 1] bằng cách nhân với 2 và trừ đi 1.
- Việc chuẩn hóa này thường được thực hiện để cải thiện hiệu suất của các mô hình học máy khi xử lý ảnh.

2.2.3. Hàm xử lý ảnh

```
def process_image(file_path, output_dir):
    try:
        image = cv2.imread(file_path)
        if image is None:
            return

        # Detect and align face
        detector = MTCNN()
        results = detector.detect_faces(image)
        if results:
            x, y, w, h = results[0]['box']
            face = image[max(0, y):y + h, max(0, x):x + w]
        else:
            face = image

        # Resize and normalize
        face = cv2.resize(face, resize_size)
        face = normalize_pixels(face)

        # Apply augmentations
        augmented = augmentations(image=face)
        face = augmented["image"]

        # Save processed image
        file_name = os.path.basename(file_path)
        output_path = os.path.join(output_dir, file_name)
        cv2.imwrite(output_path, (face * 255).astype(np.uint8))
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
```

- Đây là hàm chính để xử lý một ảnh.
- Nó nhận đường dẫn đến ảnh (file_path) và thư mục lưu trữ kết quả (output_dir) làm đầu vào.

- **Các bước xử lý:**
  - Đọc ảnh bằng cv2.imread.
  - Nếu không đọc được ảnh, hàm sẽ thoát.
  - **Phát hiện và căn chỉnh khuôn mặt:** Sử dụng thư viện MTCNN để phát hiện khuôn mặt trong ảnh. Nếu tìm thấy khuôn mặt, nó sẽ cắt ảnh để chỉ giữ lại vùng khuôn mặt. Nếu không tìm thấy, nó sẽ sử dụng toàn bộ ảnh.
  - **Thay đổi kích thước và chuẩn hóa:** Thay đổi kích thước ảnh khuôn mặt (hoặc toàn bộ ảnh) về kích thước được chỉ định bởi resize_size và chuẩn hóa giá trị pixel bằng hàm normalize_pixels.
  - **Áp dụng các kỹ thuật tăng cường dữ liệu:** Sử dụng augmentations để thực hiện các biến đổi ngẫu nhiên trên ảnh, chẳng hạn như lật ngang, xoay, thay đổi độ sáng/độ tương phản và làm mờ. Việc này giúp tăng cường dữ liệu huấn luyện cho mô hình học máy.
  - **Lưu ảnh đã xử lý:** Lưu ảnh đã xử lý vào thư mục output_dir với tên tệp gốc.

2.2.4. Xử lý và lưu kết quả

```python
def process_directory(input_dir, output_dir):
    ensure_dir_exists(output_dir)
    files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(('.jpg', '.png'))]

    with Pool() as pool:
        list(tqdm(pool.starmap(process_image, [(file, output_dir) for file in files]), total=len(files)))

if __name__ == "__main__":
    for key in input_dirs:
        print(f"Processing {key}...")
        process_directory(input_dirs[key], ouput_dirs[key])

    print("Preprocessing completed.")
```

❖ **process_directory(input_dir, output_dir)**:

- Kiểm tra và tạo thư mục đầu ra nếu chưa tồn tại (ensure_dir_exists).
- Lấy danh sách các tệp ảnh .jpg và .png trong thư mục input_dir.
- Dùng **đa luồng** (Pool) để xử lý ảnh song song qua hàm process_image, hiển thị tiến trình bằng tqdm.

❖ **Phần chính (if __name__ == "__main__")**:

- Lặp qua các thư mục đầu vào (input_dirs) và đầu ra (ouput_dirs).
- Gọi process_directory để xử lý ảnh cho từng cặp thư mục.
- Thông báo khi hoàn thành.

3. Xây dựng và huấn luyện mô hình

3.1. Các thư viện sử dụng

```
from google.colab import drive
drive.mount('/content/drive')

import os

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image

import numpy as np
import pickle as pkl
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
```

- **Google Colab Drive:** Dòng lệnh drive.mount('/content/drive') dùng để kết nối với Google Drive, cho phép truy cập dữ liệu lưu trữ trên Drive.
- **Thư viện PyTorch và các module:** Các module của PyTorch (như nn, optim, DataLoader) được sử dụng để xây dựng, huấn luyện và đánh giá các mô hình học sâu.
- **torchvision và PIL:** Dùng để xử lý ảnh (đọc ảnh, chuyển đổi, resize,…).
- **NumPy và Matplotlib:** Hỗ trợ xử lý mảng và trực quan hóa kết quả (đồ thị, hiển thị ảnh).

3.2. Định nghĩa custom dataset

```python

class Dataset(torch.utils.data.Dataset):

    def __init__(self, img_dir):
        img_dir = BASE_DATASET_PATH + "/" + img_dir + "/"

        path_list = os.listdir(img_dir)
        abspath = os.path.abspath(img_dir)

        self.img_dir = img_dir
        self.img_list = [os.path.join(abspath, path) for path in path_list]

        self.transform = transforms.Compose([
            transforms.Resize(IMG_SIZE),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5]),
# normalize image between -1 and 1
        ])

    def __len__(self):
        return len(self.img_list)

    def __getitem__(self, idx):
        path = self.img_list[idx]
        img = Image.open(path).convert('RGB')

        img_tensor = self.transform(img)
        return img_tensor
```

- **Mục đích:** Tạo ra một lớp dataset tùy chỉnh để tải và xử lý ảnh từ thư mục.
- **__init__ :**
    - Xây dựng đường dẫn đến thư mục chứa ảnh dựa trên biến BASE_DATASET_PATH và tên thư mục (vd: "trainA" hoặc "trainB").

    - Lấy danh sách các file ảnh và lưu dưới dạng danh sách các đường dẫn tuyệt đối.
    - Định nghĩa một chuỗi các biến đổi (transform) gồm: Resize (đổi kích thước ảnh), ToTensor (chuyển đổi ảnh về tensor) và Normalize (chuẩn hóa giá trị pixel về khoảng [-1, 1]).
- **__len__:** Trả về số ảnh trong dataset
- **__getitem__**: Đọc ảnh tại vị trí idx, chuyển đổi sang RGB, áp dụng biến đổi và trả về tensor ảnh

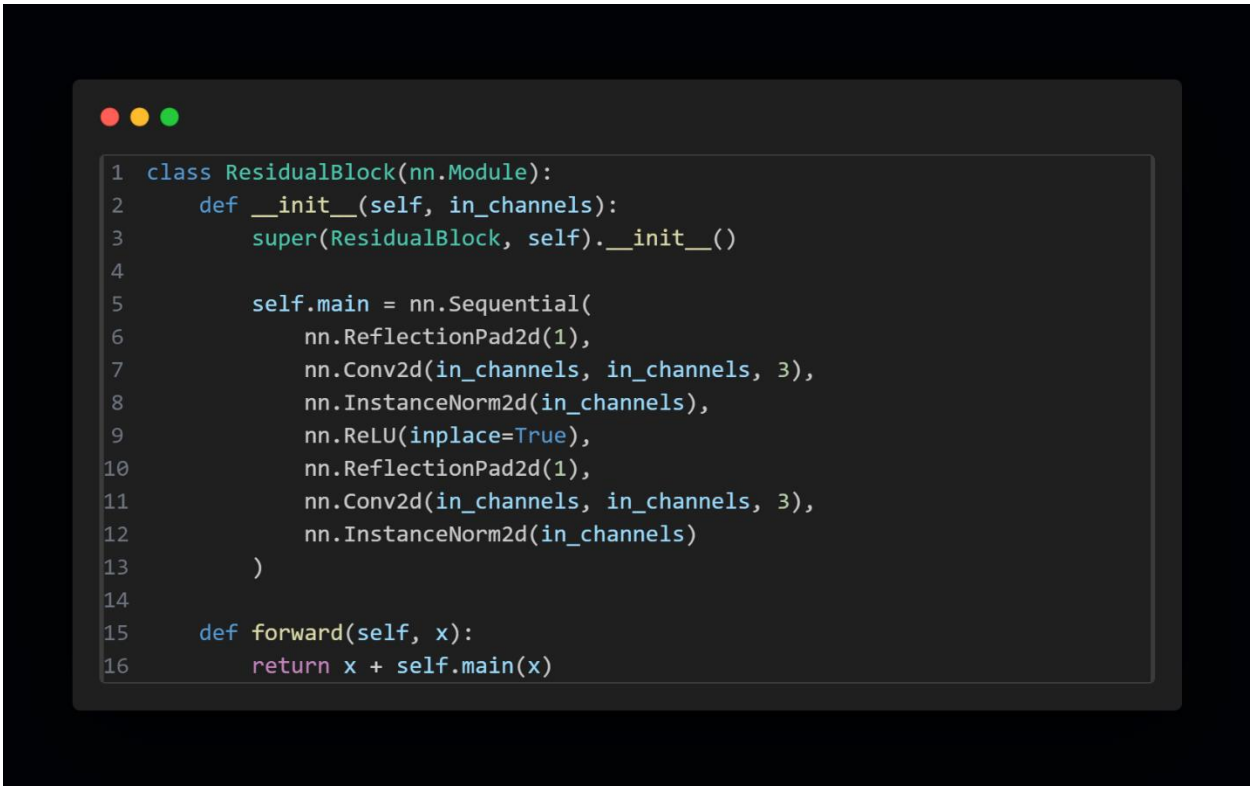
3.3. Định Nghĩa Mạng Phân Biệt (Discriminator)

```python

class Discriminator(nn.Module):
    def __init__(self, conv_dim=32):
        super(Discriminator, self).__init__()

        self.main = nn.Sequential(
            nn.Conv2d(3, conv_dim, 4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(conv_dim, conv_dim*2, 4, stride=2, padding=1),
            nn.InstanceNorm2d(conv_dim*2),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(conv_dim*2, conv_dim*4, 4, stride=2, padding=1),
            nn.InstanceNorm2d(conv_dim*4),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(conv_dim*4, conv_dim*8, 4, padding=1),
            nn.InstanceNorm2d(conv_dim*8),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(conv_dim*8, 1, 4, padding=1),
        )

    def forward(self, x):
        x = self.main(x)
        x = F.avg_pool2d(x, x.size()[2:])
        x = torch.flatten(x, 1)
        return x
```

- **Mục đích**: Mô hình phân biệt (discriminator) xác định xem một ảnh là thật (real) hay giả (fake).
- **Kiến trúc**: Gồm chuỗi các lớp convolution với kích thước bộ lọc tăng dần (từ conv_dim lên conv_dim*8), xen kẽ với InstanceNorm2d và hàm kích hoạt LeakyReLU.

- **Lớp cuối cùng:** Một lớp convolution giảm số kênh xuống 1, theo sau là trung bình pooling trên toàn bộ kích thước không gian (avg_pool2d) và flatten để đưa về dạng vector.
- **Đầu ra:** Một giá trị (hoặc vector) dùng để tính toán loss so sánh với nhãn 1 (cho ảnh thật) hoặc 0 (cho ảnh giả).

3.4. Định Nghĩa Residual Block



```python
class ResidualBlock(nn.Module):
    def __init__(self, in_channels):
        super(ResidualBlock, self).__init__()

        self.main = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels, in_channels, 3),
            nn.InstanceNorm2d(in_channels),
            nn.ReLU(inplace=True),
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels, in_channels, 3),
            nn.InstanceNorm2d(in_channels)
        )

    def forward(self, x):
        return x + self.main(x)
```

- **Mục đích:** Xây dựng một "residual block" – một khối học sâu giúp mô hình dễ dàng học các hàm nhận dạng phức tạp bằng cách truyền thông tin đầu vào qua các lớp (skip connection).
- **Chi tiết:**
    - **ReflectionPad2d:** Thêm padding với cách "reflection" để giảm biên (boundary) hiệu ứng khi sử dụng convolution.
    - **2 lớp Convolution:** Mỗi lớp đi kèm với Instance Normalization và ReLU (chỉ có ReLU sau lớp đầu tiên).
    - **Skip Connection:** Đầu ra của block được tính bằng tổng của đầu vào và đầu ra của các lớp bên trong, giúp gradient lưu thông tốt hơn.

3.5. Định Nghĩa Mạng Sinh (Generator)

```python
class Generator(nn.Module):
    def __init__(self, conv_dim=64, n_res_block=9):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ReflectionPad2d(3),
            nn.Conv2d(3, conv_dim, 7),
            nn.InstanceNorm2d(64),
            nn.ReLU(inplace=True),

            nn.Conv2d(conv_dim, conv_dim*2, 3, stride=2, padding=1),
            nn.InstanceNorm2d(conv_dim*2),
            nn.ReLU(inplace=True),
            nn.Conv2d(conv_dim*2, conv_dim*4, 3, stride=2, padding=1
),
            nn.InstanceNorm2d(conv_dim*4),
            nn.ReLU(inplace=True),

            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),
            ResidualBlock(conv_dim*4),

            nn.ConvTranspose2d(conv_dim*4, conv_dim*2, 3, stride=2,
padding=1, output_padding=1),
            nn.InstanceNorm2d(conv_dim*2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(conv_dim*2, conv_dim, 3, stride=2,
padding=1, output_padding=1),
            nn.InstanceNorm2d(conv_dim),
            nn.ReLU(inplace=True),

            nn.ReflectionPad2d(3),
            nn.Conv2d(conv_dim, 3, 7),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)

```

- **Mục đích:** Chuyển đổi ảnh từ miền này sang miền khác (vd: từ ảnh selfie sang ảnh anime).
- **Các phần chính:**
    - **Phần Encoding (Downsampling):**
        - Sử dụng ReflectionPad2d và Convolution với kích thước kernel 7 để bắt đầu.
        - Sau đó là các lớp convolution có stride=2 để giảm kích thước (downsampling).
    - **Phần Body – Các Residual Block:**
        - Sử dụng nhiều khối residual (ở đây là 9 khối) để trích xuất đặc trưng và giúp mạng học các đặc trưng phức tạp.
    - **Phần Decoding (Upsampling):**
        - Sử dụng các lớp ConvTranspose2d để tăng kích thước ảnh (upsampling) dần dần.
        - Cuối cùng sử dụng ReflectionPad2d và một lớp convolution để đưa số kênh trở lại 3, và **Tanh** để giới hạn đầu ra trong khoảng [-1, 1].
- **Đầu ra:** Ảnh được chuyển đổi sang phong cách của miền đích (anime hoặc selfie tùy theo hướng chuyển đổi).

3.6. Lớp CycleGan

```python
class CycleGAN:

    def __init__(self, g_conv_dim=64, d_conv_dim=64, n_res_block=6):
        self.device = torch.device('cuda') if
 torch.cuda.is_available() else torch.device("cpu")

        self.G_XtoY = Generator(conv_dim=g_conv_dim, n_res_block=
n_res_block).to(self.device)
        self.G_YtoX = Generator(conv_dim=g_conv_dim, n_res_block=
n_res_block).to(self.device)

        self.D_X = Discriminator(conv_dim=d_conv_dim).to(self.device)
        self.D_Y = Discriminator(conv_dim=d_conv_dim).to(self.device)

        print(f"Models running of {self.device}")

    def load_model(self, filename):
        save_filename = os.path.splitext(os.path.basename(filename))[
0] + '.pt'
        return torch.load(save_filename)

    def real_mse_loss(self, D_out):
        return torch.mean((D_out-1)**2)

    def fake_mse_loss(self, D_out):
        return torch.mean(D_out**2)

    def cycle_consistency_loss(self, real_img, reconstructed_img,
lambda_weight):
        reconstr_loss = torch.mean(torch.abs(real_img -
reconstructed_img))
        return lambda_weight*reconstr_loss
```

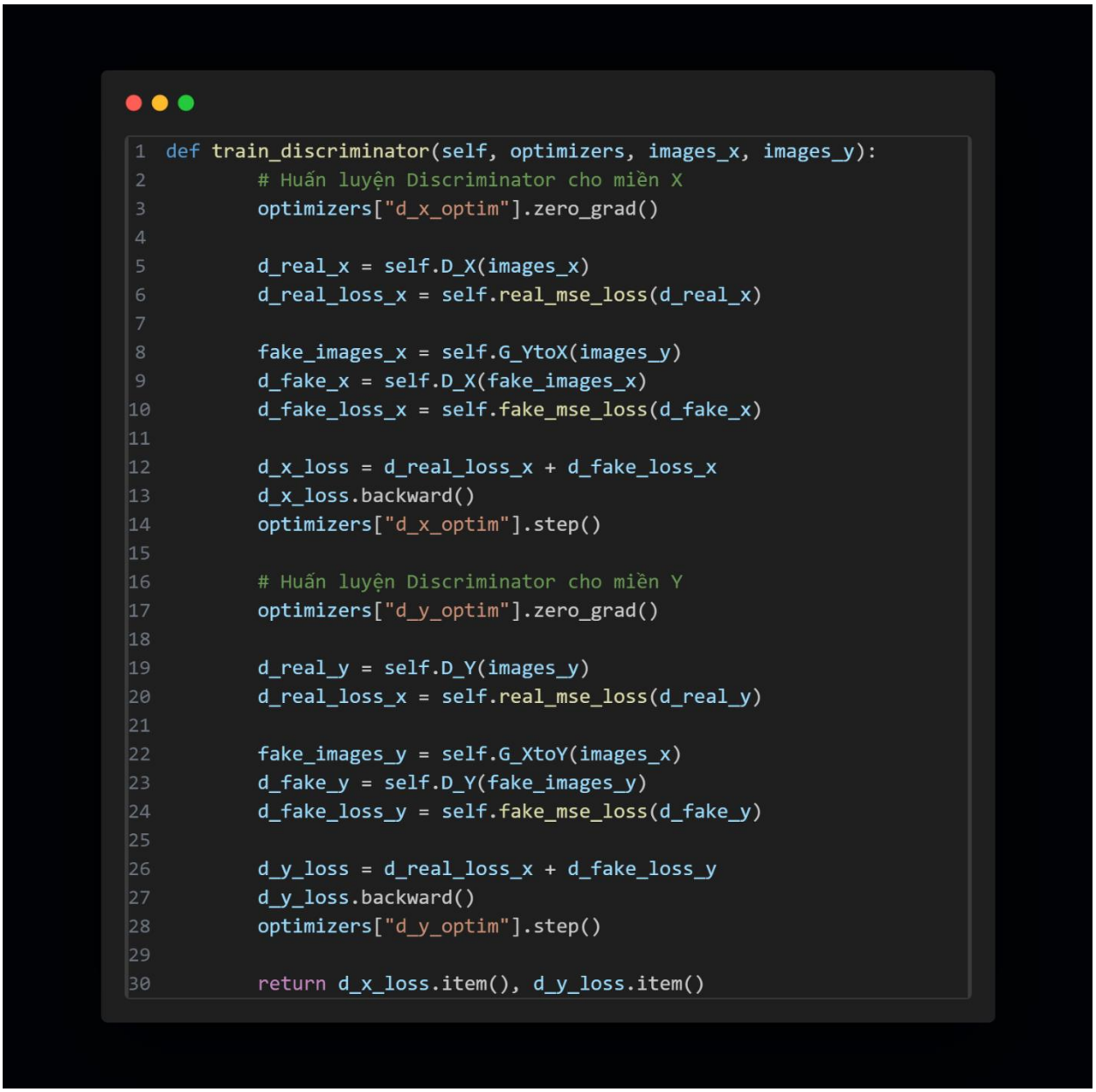
- **Khởi tạo mô hình:** Tạo 2 generator (`G_XtoY` và `G_YtoX`) và 2 discriminator (`D_X` và `D_Y`), đồng thời chuyển chúng sang thiết bị tính toán (GPU nếu có, ngược lại CPU).
- **Hàm loss:**
    - `real_mse_loss`: Tính MSE loss giữa output của discriminator với giá trị 1 (cho ảnh thật).
    - `fake_mse_loss`: Tính MSE loss giữa output của discriminator với giá trị 0 (cho ảnh giả).

- `cycle_consistency_loss`: Tính loss giữa ảnh gốc và ảnh được tái tạo sau khi chuyển qua generator hai chiều, nhân với hệ số lambda (ở đây lambda = 10) để đảm bảo tính nhất quán chu trình.

```python
def train_generator(self, optimizers, images_x, images_y):
        # Huấn luyện Generator từ Y sang X
        optimizers["g_optim"].zero_grad()

        fake_images_x = self.G_YtoX(images_y)
        d_real_x = self.D_X(fake_images_x)
        g_YtoX_loss = self.real_mse_loss(d_real_x)

        recon_y = self.G_XtoY(fake_images_x)
        recon_y_loss = self.cycle_consistency_loss(images_y, recon_y
, lambda_weight=10)

        # Huấn luyện Generator từ X sang Y
        fake_images_y = self.G_XtoY(images_x)
        d_real_y = self.D_Y(fake_images_y)
        g_XtoY_loss = self.real_mse_loss(d_real_y)

        recon_x = self.G_YtoX(fake_images_y)
        recon_x_loss = self.cycle_consistency_loss(images_x, recon_x
, lambda_weight=10)

        # Tổng hợp loss cho các generator
        g_total_loss = g_YtoX_loss + g_XtoY_loss + recon_y_loss +
recon_x_loss
        g_total_loss.backward()
        optimizers["g_optim"].step()

        return g_total_loss.item()
```

Phương pháp huấn luyện Generator (train_generator):

- Sinh ảnh giả từ một miền (vd: chuyển ảnh từ Y sang X) và tính loss dựa trên khả năng "lừa" được discriminator.
- Sau đó, thực hiện "cycle consistency" bằng cách chuyển ảnh giả trở lại miền ban đầu và so sánh với ảnh thật.
- Tương tự cho hướng chuyển từ X sang Y.
- Tổng hợp các loss lại và cập nhật tham số của generator.



```python
def train_discriminator(self, optimizers, images_x, images_y):
        # Huấn luyện Discriminator cho miền X
        optimizers["d_x_optim"].zero_grad()

        d_real_x = self.D_X(images_x)
        d_real_loss_x = self.real_mse_loss(d_real_x)

        fake_images_x = self.G_YtoX(images_y)
        d_fake_x = self.D_X(fake_images_x)
        d_fake_loss_x = self.fake_mse_loss(d_fake_x)

        d_x_loss = d_real_loss_x + d_fake_loss_x
        d_x_loss.backward()
        optimizers["d_x_optim"].step()

        # Huấn luyện Discriminator cho miền Y
        optimizers["d_y_optim"].zero_grad()

        d_real_y = self.D_Y(images_y)
        d_real_loss_x = self.real_mse_loss(d_real_y)

        fake_images_y = self.G_XtoY(images_x)
        d_fake_y = self.D_Y(fake_images_y)
        d_fake_loss_y = self.fake_mse_loss(d_fake_y)

        d_y_loss = d_real_loss_x + d_fake_loss_y
        d_y_loss.backward()
        optimizers["d_y_optim"].step()

        return d_x_loss.item(), d_y_loss.item()
```

Phương pháp huấn luyện Discriminator (train_discriminator):

- Đối với mỗi discriminator, tính loss trên ảnh thật (so sánh với giá trị 1) và ảnh giả (so sánh với giá trị 0), sau đó cập nhật tham số.

```python
def train(self, optimizers, data_loader_x, data_loader_y, print_every=10, sample_every=100):
        losses = []
        g_total_loss_min = np.Inf

        fixed_x = next(iter(data_loader_x))[1].to(self.device)
        fixed_y = next(iter(data_loader_y))[1].to(self.device)

        print(f'Running on {self.device}')
        for epoch in range(EPOCHS):
            for (images_x, images_y) in zip(data_loader_x, data_loader_y):
                images_x, images_y = images_x.to(self.device), images_y.to(self.device)

                # Cập nhật Generator: tính loss đối với các generator và cập nhật tham số
                g_total_loss = self.train_generator(optimizers, images_x, images_y)
                # Cập nhật Discriminator: tính loss cho từng discriminator và cập nhật tham số
                d_x_loss, d_y_loss = self.train_discriminator(optimizers, images_x, images_y)

            if epoch % print_every == 0:
                losses.append((d_x_loss, d_y_loss, g_total_loss))
                print('Epoch [{:5d}/{:5d}] | d_X_loss: {:6.4f} | d_Y_loss: {:6.4f} | g_total_loss: {:6.4f}'
                .format(
                    epoch,
                    EPOCHS,
                    d_x_loss,
                    d_y_loss,
                    g_total_loss
                ))

            # Lưu model khi loss của generator giảm xuống mức thấp nhất đạt được
            if g_total_loss < g_total_loss_min:
                g_total_loss_min = g_total_loss

                torch.save(self.G_XtoY.state_dict(), "G_X2Y")
                torch.save(self.G_YtoX.state_dict(), "G_Y2X")

                torch.save(self.D_X.state_dict(), "D_X")
                torch.save(self.D_Y.state_dict(), "D_Y")

                print("Models Saved")

        return losses

```

Hàm train:

- Lặp qua số lượng epoch (EPOCHS) và qua các batch dữ liệu của 2 miền.
- Trong mỗi epoch, cập nhật lần lượt generator và discriminator.
- In ra thông tin loss định kỳ (mỗi print_every epoch) và lưu mô hình nếu loss của generator giảm xuống mức thấp nhất.

3.7. Cấu Hình và Thiết Lập Tham Số

```python
BASE_DATASET_PATH = "/content/drive/MyDrive/SE355/Đồ án/Datasets/"
X_DATASET = "trainA"
Y_DATASET = "trainB"
BATCH_SIZE = 32
N_WORKERS = 0

IMG_SIZE = 128
LR = 0.0002
BETA1 = 0.5
BETA2 = 0.999

EPOCHS = 100
```

- **BASE_DATASET_PATH:** Đường dẫn gốc đến bộ dữ liệu trên Google Drive.
- **X_DATASET & Y_DATASET:** Tên các thư mục chứa ảnh của 2 miền (ví dụ: "trainA" cho ảnh selfie và "trainB" cho ảnh anime).
- **Hyperparameters:** Kích thước ảnh, learning rate, batch size, số epoch,… được thiết lập cho quá trình huấn luyện.

3.8. Thiết Lập Dữ Liệu và Huấn Luyện Mô Hình

3.8.1. Tạo Dataset và DataLoader

```
# Dataset
x_dataset = Dataset(X_DATASET)
y_dataset = Dataset(Y_DATASET)

data_loader_x = DataLoader(x_dataset, BATCH_SIZE, shuffle=True, num_workers=N_WORKERS)
data_loader_y = DataLoader(y_dataset, BATCH_SIZE, shuffle=True, num_workers=N_WORKERS)
```

- **Dataset:** Khởi tạo các đối tượng dataset cho 2 miền.
- **DataLoader:** Tạo DataLoader để chia dữ liệu thành các batch, giúp quá trình huấn luyện hiệu quả hơn.

3.8.2. Khởi Tạo Mô Hình CycleGAN và Optimizer

```python
# Model
cycleGan = CycleGAN()

# Optimizer
g_params = list(cycleGan.G_XtoY.parameters()) + list(cycleGan.G_YtoX.parameters())

optimizers = {
    "g_optim": optim.Adam(g_params, LR, [BETA1, BETA2]),
    "d_x_optim": optim.Adam(cycleGan.D_X.parameters(), LR, [BETA1, BETA2]),
    "d_y_optim": optim.Adam(cycleGan.D_Y.parameters(), LR, [BETA1, BETA2])
}

```

- **CycleGAN:** Khởi tạo mô hình, trong đó đã có 2 generator và 2 discriminator.
- **Optimizers:** Sử dụng thuật toán Adam để cập nhật tham số cho:
    - Các generator (cùng một optimizer với tập hợp tham số của cả 2 generator).
    - Mỗi discriminator có một optimizer riêng biệt.

3.8.3. Huấn Luyện Mô Hình

```python
# Train
losses = cycleGan.train(optimizers, data_loader_x, data_loader_y, print_every=1)
```

- Gọi hàm train của lớp CycleGAN để bắt đầu huấn luyện qua số epoch đã định (ở đây 100 epoch).
- Trong quá trình huấn luyện, loss được in ra và mô hình được lưu khi loss của generator giảm xuống mức thấp nhất.

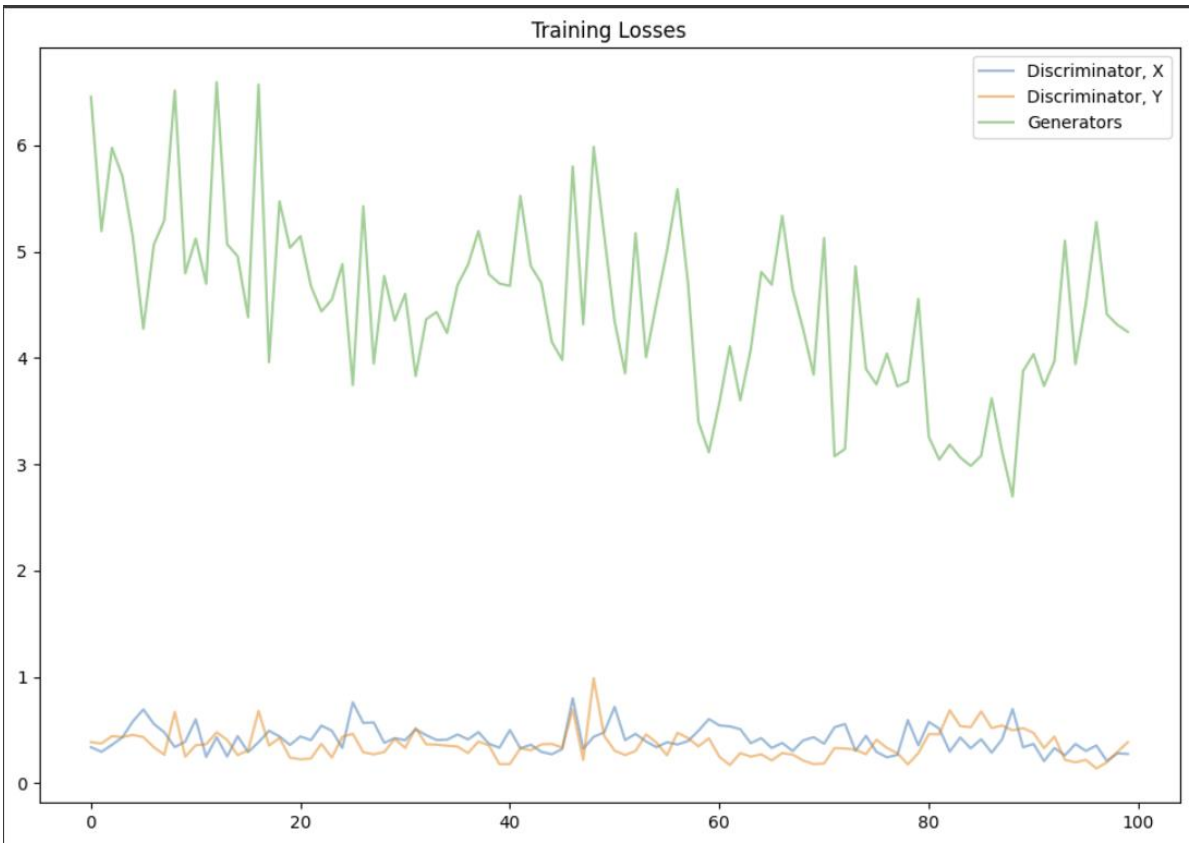
```
Models running of cuda
Running on cuda
Epoch [    0/  100] | d_X_loss: 0.3377 | d_Y_loss: 0.3850 | g_total_loss: 6.4559
Models Saved
Epoch [    1/  100] | d_X_loss: 0.2947 | d_Y_loss: 0.3728 | g_total_loss: 5.1948
Models Saved
Epoch [    2/  100] | d_X_loss: 0.3597 | d_Y_loss: 0.4416 | g_total_loss: 5.9769
Epoch [    3/  100] | d_X_loss: 0.4312 | d_Y_loss: 0.4333 | g_total_loss: 5.7102
Epoch [    4/  100] | d_X_loss: 0.5810 | d_Y_loss: 0.4534 | g_total_loss: 5.1332
Models Saved
Epoch [    5/  100] | d_X_loss: 0.6934 | d_Y_loss: 0.4327 | g_total_loss: 4.2753
Models Saved
Epoch [    6/  100] | d_X_loss: 0.5572 | d_Y_loss: 0.3376 | g_total_loss: 5.0667
Epoch [    7/  100] | d_X_loss: 0.4776 | d_Y_loss: 0.2667 | g_total_loss: 5.2957
Epoch [    8/  100] | d_X_loss: 0.3383 | d_Y_loss: 0.6695 | g_total_loss: 6.5167
Epoch [    9/  100] | d_X_loss: 0.3899 | d_Y_loss: 0.2484 | g_total_loss: 4.7969
Epoch [   10/  100] | d_X_loss: 0.6001 | d_Y_loss: 0.3563 | g_total_loss: 5.1237
Epoch [   11/  100] | d_X_loss: 0.2466 | d_Y_loss: 0.3661 | g_total_loss: 4.6988
Epoch [   12/  100] | d_X_loss: 0.4319 | d_Y_loss: 0.4770 | g_total_loss: 6.5955
Epoch [   13/  100] | d_X_loss: 0.2496 | d_Y_loss: 0.4067 | g_total_loss: 5.0703
Epoch [   14/  100] | d_X_loss: 0.4423 | d_Y_loss: 0.2635 | g_total_loss: 4.9570
Epoch [   15/  100] | d_X_loss: 0.2918 | d_Y_loss: 0.3079 | g_total_loss: 4.3838
Epoch [   16/  100] | d_X_loss: 0.3842 | d_Y_loss: 0.6789 | g_total_loss: 6.5724
Epoch [   17/  100] | d_X_loss: 0.4922 | d_Y_loss: 0.3533 | g_total_loss: 3.9610
Models Saved
Epoch [   18/  100] | d_X_loss: 0.4389 | d_Y_loss: 0.4247 | g_total_loss: 5.4734
Epoch [   19/  100] | d_X_loss: 0.3589 | d_Y_loss: 0.2400 | g_total_loss: 5.0375
Epoch [   20/  100] | d_X_loss: 0.4391 | d_Y_loss: 0.2239 | g_total_loss: 5.1460
Epoch [   21/  100] | d_X_loss: 0.4040 | d_Y_loss: 0.2313 | g_total_loss: 4.6778
Epoch [   22/  100] | d_X_loss: 0.5408 | d_Y_loss: 0.3692 | g_total_loss: 4.4380
Epoch [   23/  100] | d_X_loss: 0.4928 | d_Y_loss: 0.2398 | g_total_loss: 4.5486
Epoch [   24/  100] | d_X_loss: 0.3310 | d_Y_loss: 0.4356 | g_total_loss: 4.8835
Epoch [   25/  100] | d_X_loss: 0.7598 | d_Y_loss: 0.4640 | g_total_loss: 3.7453
Models Saved
Epoch [   26/  100] | d_X_loss: 0.5656 | d_Y_loss: 0.2899 | g_total_loss: 5.4278
Epoch [   27/  100] | d_X_loss: 0.5706 | d_Y_loss: 0.2709 | g_total_loss: 3.9478
Epoch [   28/  100] | d_X_loss: 0.3789 | d_Y_loss: 0.2924 | g_total_loss: 4.7717
Epoch [   29/  100] | d_X_loss: 0.4222 | d_Y_loss: 0.4161 | g_total_loss: 4.3515
Epoch [   30/  100] | d_X_loss: 0.4038 | d_Y_loss: 0.3306 | g_total_loss: 4.6029
Epoch [   31/  100] | d_X_loss: 0.5064 | d_Y_loss: 0.5167 | g_total_loss: 3.8302
Epoch [   32/  100] | d_X_loss: 0.4514 | d_Y_loss: 0.3658 | g_total_loss: 4.3635
Epoch [   33/  100] | d_X_loss: 0.4056 | d_Y_loss: 0.3620 | g_total_loss: 4.4325
Epoch [   34/  100] | d_X_loss: 0.4087 | d_Y_loss: 0.3518 | g_total_loss: 4.2357
Epoch [   35/  100] | d_X_loss: 0.4579 | d_Y_loss: 0.3434 | g_total_loss: 4.6854
Epoch [   36/  100] | d_X_loss: 0.4112 | d_Y_loss: 0.2823 | g_total_loss: 4.8778
Epoch [   37/  100] | d_X_loss: 0.4797 | d_Y_loss: 0.3915 | g_total_loss: 5.1941
Epoch [   38/  100] | d_X_loss: 0.3702 | d_Y_loss: 0.3539 | g_total_loss: 4.7876
Epoch [   39/  100] | d_X_loss: 0.3329 | d_Y_loss: 0.1784 | g_total_loss: 4.7006
```

3.9. Trực Quan Hóa Loss và Kết Quả Kiểm Tra

3.9.1. Vẽ Đồ Thị Loss

```
fig, ax = plt.subplots(figsize=(12,8))
losses = np.array(losses)
plt.plot(losses.T[0], label='Discriminator, X', alpha=0.5)
plt.plot(losses.T[1], label='Discriminator, Y', alpha=0.5)
plt.plot(losses.T[2], label='Generators', alpha=0.5)
plt.title("Training Losses")
plt.legend()
plt.show()
```

- Sử dụng matplotlib để vẽ đồ thị loss theo từng epoch của:
    - Discriminator miền X
    - Discriminator miền Y
    - Tổng loss của các generator
- Giúp theo dõi tiến trình huấn luyện và đánh giá chất lượng của mô hình.



- **Discriminator Loss (X và Y)**:
  - Giá trị dao động ở mức thấp (dưới 1), điều này cho thấy các Discriminator hoạt động ổn định và học được cách phân biệt giữa ảnh thật và ảnh giả do Generator tạo ra.
  - Có sự dao động nhỏ nhưng không có xu hướng tăng/giảm rõ rệt, cho thấy các Discriminator đã đạt trạng thái cân bằng.
- **Generator Loss**:
  - Loss của Generator cao hơn hẳn (dao động từ 3 đến 7), điều này phản ánh mức độ khó khăn trong việc tạo ra các mẫu ảnh đủ tốt để "đánh lừa" Discriminator.
  - Đường loss biến động lớn, cho thấy Generator vẫn đang cố gắng cải thiện chất lượng ảnh tạo ra, nhưng có thể chưa đạt được sự ổn định hoàn toàn.

- **Cân bằng giữa Generator và Discriminator**:

- Discriminator loss ổn định ở mức thấp trong khi Generator loss dao động cao cho thấy hệ thống có thể đang gặp tình trạng "cạnh tranh không cân bằng", nơi Discriminator đang hoạt động hiệu quả hơn Generator.

3.9.2. Kiểm Tra Mô Hình Trên Dữ Liệu Test

```python
# Test Dataset
x_dataset = Dataset("testA")
y_dataset = Dataset("testB")

data_loader_x = DataLoader(x_dataset, BATCH_SIZE, shuffle=True, num_workers=N_WORKERS)
data_loader_y = DataLoader(y_dataset, BATCH_SIZE, shuffle=True, num_workers=N_WORKERS)

samples = []

for i in range(12):
    fixed_x = next(iter(data_loader_x))[i].to(cycleGan.device)
    fake_y = cycleGan.G_XtoY(torch.unsqueeze(fixed_x, dim=0))
```

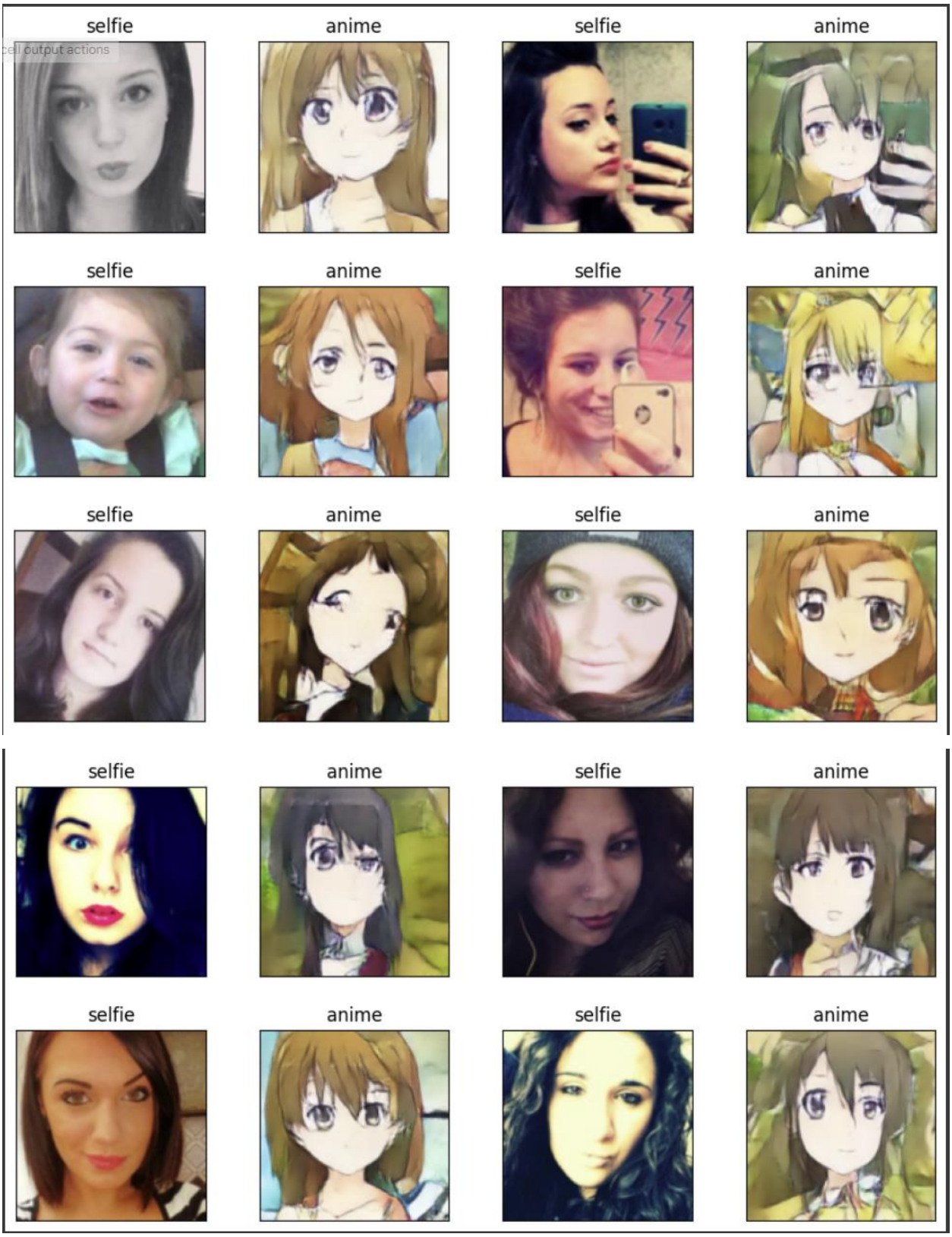
- **Chuẩn bị dữ liệu test**: Tương tự như huấn luyện nhưng dùng các thư mục "testA" và "testB".
- **Sinh mẫu**: Với mỗi ảnh trong batch, sử dụng generator *G_XtoY* để chuyển đổi ảnh từ miền X (selfie) sang miền Y (anime). Các ảnh gốc và ảnh chuyển đổi được lưu vào danh sách samples.

3.9.3. Hiển Thị Kết Quả

```python
fig = plt.figure(figsize=(18, 14))
grid = ImageGrid(fig, 111, nrows_ncols=(5, 4), axes_pad=0.5)

for i, (ax, im) in enumerate(zip(grid, samples)):
    _, w, h = im.size()
    im = im.detach().cpu().numpy()
    im = np.transpose(im, (1, 2, 0))
    im = ((im +1)*255 / (2)).astype(np.uint8)
    ax.imshow(im.reshape((w,h,3)))
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    if i % 2 == 0:
        title = "selfie"
    else:
        title = "anime"
    ax.set_title(title)

plt.show()
```

- **Mục đích:** Hiển thị kết quả so sánh giữa ảnh gốc (selfie) và ảnh được chuyển đổi (anime) theo dạng lưới (grid).
- **Các bước xử lý:**
  - Chuyển đổi tensor ảnh về dạng NumPy, chuyển trục từ (C, W, H) sang (W, H, C).
  - Giải chuẩn hóa (từ khoảng [-1, 1] về khoảng [0, 255]) để hiển thị dưới dạng ảnh RGB.
  - Vẽ ảnh lên lưới và đặt tiêu đề cho từng ảnh (ảnh gốc hay ảnh chuyển đổi).

3.9.4. Kết quả


selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime
selfie
anime

4. Tổng kết & hướng phát triển

4.1. Nhận xét output

❖ **Ưu điểm:**

1. **Định dạng khuôn mặt hợp lý**:
    - Các khuôn mặt anime đã được tạo đúng vị trí và kích thước tương ứng với khuôn mặt selfie gốc.
    - Tỉ lệ mắt, mũi, miệng trên ảnh anime phản ánh tương đối đúng cấu trúc khuôn mặt gốc.
2. **Phong cách anime rõ nét**:
    - Các đặc trưng của phong cách anime như mắt to, nét vẽ đơn giản và màu sắc sáng đã được áp dụng hiệu quả.
3. **Chuyển đổi màu sắc**:
    - Tông màu tóc và da từ ảnh gốc đã được mô phỏng tương đối tốt trong phong cách anime.

❖ **Nhược điểm:**

1. **Chi tiết hình ảnh chưa sắc nét**:
    - Ảnh anime trông hơi mờ và không rõ ràng ở một số vùng, đặc biệt là các chi tiết nhỏ như tóc hoặc quần áo.
    - Một số hình ảnh có hiện tượng bị "nhòe" hoặc các nét vẽ không rõ ràng, đặc biệt là hình thứ hai và thứ ba.
2. **Thiếu sáng tạo trong nền**:
    - Nền của ảnh anime dường như không mang tính nghệ thuật cao và khá giống nhau giữa các hình, không tạo cảm giác khác biệt hoặc độc đáo.
3. **Biểu cảm khuôn mặt**:
    - Biểu cảm khuôn mặt trong ảnh anime đôi khi chưa sát với ảnh gốc (ví dụ: ảnh selfie đầu tiên có biểu cảm rất rõ nhưng ảnh anime chỉ tái hiện một cách đơn giản).

4.2. Cải thiện & Hướng phát triển

4.2.1. Cải thiện

- Cải thiện dữ liệu: Làm sạch dataset, chuẩn hóa kích thước, tăng cường dữ liệu với augmentation.
- Tăng độ phân giải: Sử dụng SRGAN hoặc Real-ESRGAN để làm ảnh đầu ra sắc nét hơn.
- Điều chỉnh mô hình: Thêm attention mechanism, thử các mô hình nâng cấp như CUT, StyleGAN2 hoặc AnimeGAN2.
- Tối ưu hàm mất mát: Áp dụng Perceptual Loss, Style Loss, hoặc Adversarial Loss để cân bằng giữa chất lượng và phong cách.
- Hậu xử lý: Dùng kỹ thuật làm mượt, tăng chi tiết bằng OpenCV hoặc các công cụ xử lý ảnh khác.

4.2.2. Hướng phát triển

- Tạo ứng dụng chuyển đổi ảnh selfie thành phong cách anime trên nền tảng di động hoặc web.
- Phát triển API hoặc plugin tích hợp cho các mạng xã hội hoặc phần mềm chỉnh sửa ảnh.
- Thu thập và tạo dataset đa dạng hơn, bao gồm nhiều phong cách anime khác nhau (chibi, fantasy, action...).
- Tích hợp thêm các phong cách nghệ thuật khác ngoài anime, như tranh vẽ tay, phong cách truyện tranh Mỹ.