ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

DHMT01

BÁO CÁO

Ứng Dụng Xử Lý Ảnh và Video Số

Báo Cáo Nghiên Cứu JoJoGAN

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

UDXLA

BÁO CÁO

ĐỒ ÁN

Ứng Dụng Xử Lý Ảnh và Video Số

Giảng Viên Hướng Dẫn

Phạm Minh Hoàng

Võ Hoài Việt

Phạm Thanh Tùng

Sinh viên:

Phan Hữu Đoàn Anh - 20127110

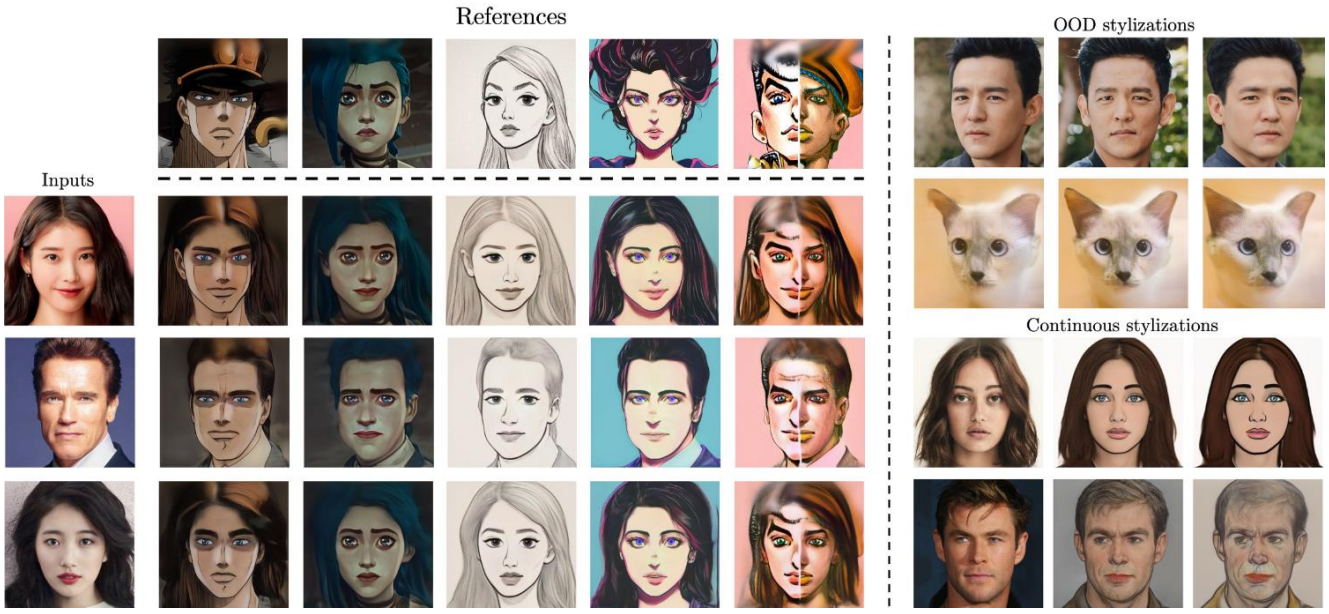
Bảng thể hiện tỉ lệ hoàn thành các nội dung tương ứng của project:

Công việc	Tiến độ
mô tả ứng dụng xử lý ảnh / video trong project theo kết quả tìm hiểu của bản thân	100%
minh chứng kết quả cài đặt, thực nghiệm, các vấn đề, khó khăn gặp phải (nếu có) trong quá trình thực hiện...	100%
đưa ra các nhận xét, đánh giá, diễn giải tương ứng	100%

Động Lực Nghiên Cứu:

Trong những năm gần đây, đã có những tiến bộ trong việc áp dụng phong cách hình ảnh bằng phương pháp few-shot, tuy nhiên những phương pháp này thường không thể tái hiện được những chi tiết phong cách mà con người dễ nhận biết. Những chi tiết như hình dáng của mắt, độ đậm của các đường nét, đặc biệt khó cho một mô hình học máy học được, đặc biệt là trong trường hợp có ít dữ liệu huấn luyện.

Trong nghiên cứu này, chúng tôi nhằm mục tiêu thực hiện phong cách hóa hình ảnh one-shot nhằm tái tạo đúng các chi tiết. Dựa trên một hình ảnh phong cách tham chiếu, chúng tôi tiếp cận gần đúng dữ liệu thực sử dụng GAN inversion và điều chỉnh lại một mô hình StyleGAN đã được huấn luyện trước bằng cách sử dụng dữ liệu gần đúng đó. Sau đó, chúng tôi khuyến khích StyleGAN tổng quát hóa để phong cách đã học có thể được áp dụng cho tất cả các hình ảnh khác.



Phát biểu bài toán:

Bộ ánh xạ phong cách áp dụng một phong cách cố định cho các hình ảnh đầu vào của nó. JoJoGAN (quy trình của chúng tôi) sử dụng một hình ảnh tham chiếu (hoặc nhiều hình ảnh - nhưng một hình ảnh là đủ) và tạo ra một tập dữ liệu cặp bằng cách sử dụng GAN inversion và tính chất style-mixing của StyleGAN. Tập dữ liệu cặp này được sử dụng để điều chỉnh lại StyleGAN bằng cách sử dụng một hàm mất mát trực tiếp cấp pixel mới. Cơ chế hoạt động rất đơn giản: chúng ta có thể thu được một bộ ánh xạ (và do đó là một nguồn cung cấp giàu có của các chân dung được phong cách hóa) từ một hình ảnh tham chiếu duy nhất trong vòng ít phút.

Mục đích bài toán

-Một quy trình để học một bộ ánh xạ phong cách nên:

-dễ sử dụng

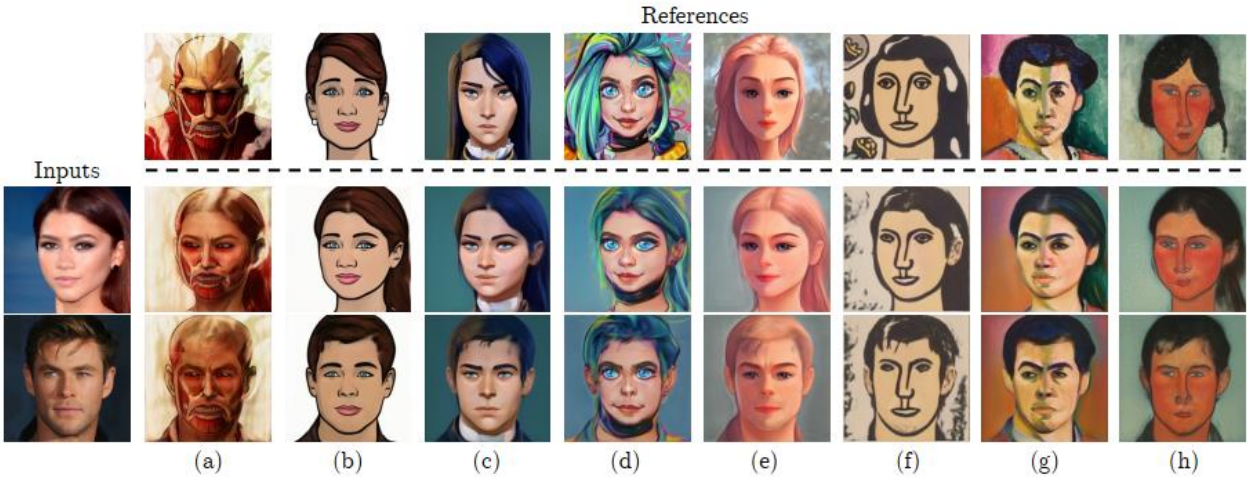
-tạo ra kết quả thuyết phục và chất lượng cao

-cho phép người dùng kiểm soát mức độ chuyển đổi phong cách

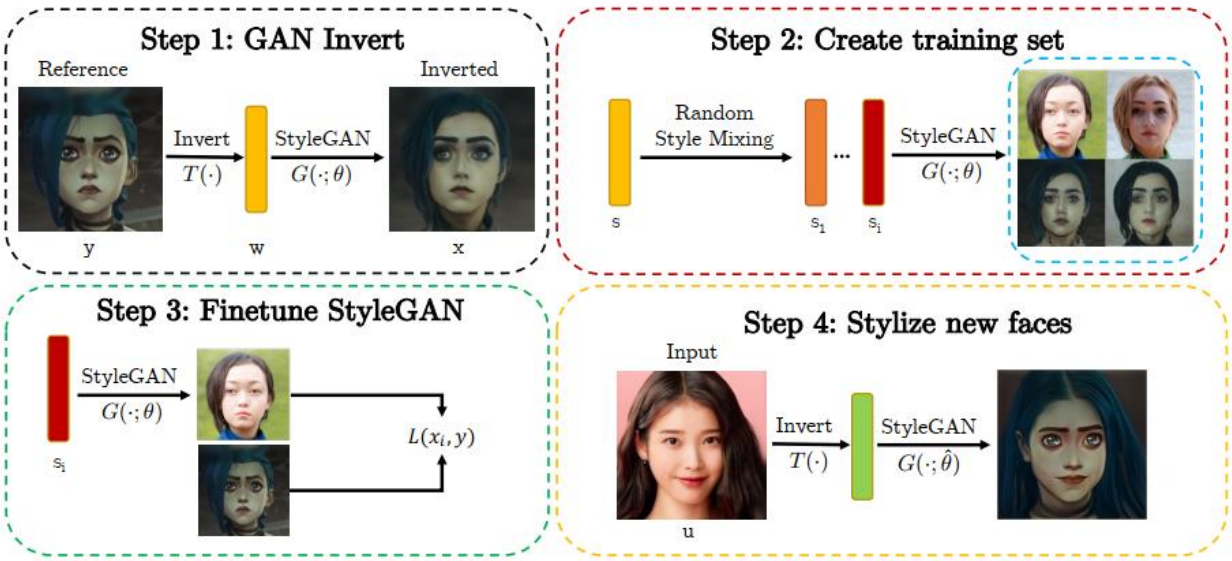
-cho phép người dùng có kiến thức chuyên sâu kiểm soát những khía cạnh nào của phong cách được chuyển đổi.

Đầu vào (Input): Hình ảnh cần tham chiếu phong cách và phong cách tham chiếu

Đầu ra (Output): Hình ảnh cần tham chiếu phong cách sau khi tham chiếu.



Hướng tiếp cận:



Các bước của JoJoGAN là:

Gọi T là GAN inversion, G là StyleGAN, s là các tham số phong cách trong không gian S của StyleGAN và θ là các tham số của StyleGAN gốc. JoJoGAN sử dụng bốn bước:

1. GAN Inversion: GAN invert hình ảnh references y để có được một style code w = T(y) và từ đó tập hợp một tham số s(w).
2. Tạo tập huấn luyện: Sử dụng s để tìm một tập hợp các mã code phong cách S gần với s. Các cặp (si,y) với si thuộc S sẽ là tập huấn luyện cặp của chúng tôi.
3. Điều chỉnh lại (Finetuning styleGAN): Điều chỉnh lại StyleGAN để thu được θˆ sao cho

$$G(s_i; \hat{\theta}) \approx y.$$

4. Suy luận (Inference): Đối với đầu vào u, hình ảnh khuôn mặt được phong cách hoá là

Chi tiết các bước thực hiện:

Bước 1: GAN Inversion

- Chúng ta sử dụng GAN Inversion để chuyển đổi hình ảnh phong cách tham chiếu thành một mã code phong cách (s). Điều đáng ngạc nhiên là khi chúng ta sử dụng mã code phong cách này để tạo ra một hình ảnh khuôn mặt bằng cách sử dụng StyleGAN (G), chúng ta thu được một hình ảnh khuôn mặt thực tế thay vì một hình ảnh được phong cách hóa.

- Điều này xảy ra vì GAN Inversion được huấn luyện để tạo ra mã code dẫn đến các hình ảnh khuôn mặt thực tế. Trong quá trình huấn luyện, GAN Inversion không được tiếp xúc với các hình ảnh khuôn mặt đã được phong cách hóa, do đó nó không thể tổng quát hóa đúng cách trong việc tạo ra các hình ảnh khuôn mặt đã được phong cách hóa. Thay vào đó, nó tạo ra các hình ảnh khuôn mặt trông thực tế. Điều này là một tính chất hữu ích trong việc xây dựng quy trình JoJoGAN, vì chúng ta muốn tạo ra

các hình ảnh khuôn mặt có chất lượng thực tế chứ không chỉ đơn thuần là hình ảnh được phong cách hóa.

Bước 2: Training set:

- Sau khi chúng ta đã thực hiện GAN Inversion và thu được mã code phong cách s(w) từ hình ảnh phong cách tham chiếu, chúng ta cần tìm một tập hợp các mã code phong cách S gần giống với s(w). Chúng tôi sử dụng một phiên bản StyleGAN2 với độ phân giải 1024 và 26 lớp điều chỉnh phong cách, do đó s ∈ R ^ 26×512.

- Gọi M ∈ {0, 1}^26 là một mặt nạ cố định, FC là lớp ánh xạ phong cách của StyleGAN và zi ∼ N(0, I). Chúng ta tạo ra các mã code phong cách mới bằng cách sử dụng công thức:

$$s_i = M \cdot s + (1 - M) \cdot s(FC(z_i))$$

Mỗi mặt nạ M sẽ tạo ra các hiệu ứng phong cách khác nhau.

Bước 3: Finetuning (Điều chỉnh lại):

Chúng ta cần tìm giá trị của θˆ để giảm thiểu hàm mất mát loss(θ), được tính bằng cách tính toán sự sai khác giữa đầu ra của StyleGAN khi áp dụng mã code phong cách si (từ tập hợp S) và hình ảnh tham chiếu y. Chúng ta sử dụng hàm mất mát perceptual loss L, được đề xuất mới, để đo lường sự khác biệt giữa hai hình ảnh.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \operatorname{loss}(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i}^{N} \mathcal{L}(G(s_i; \theta), y)$$

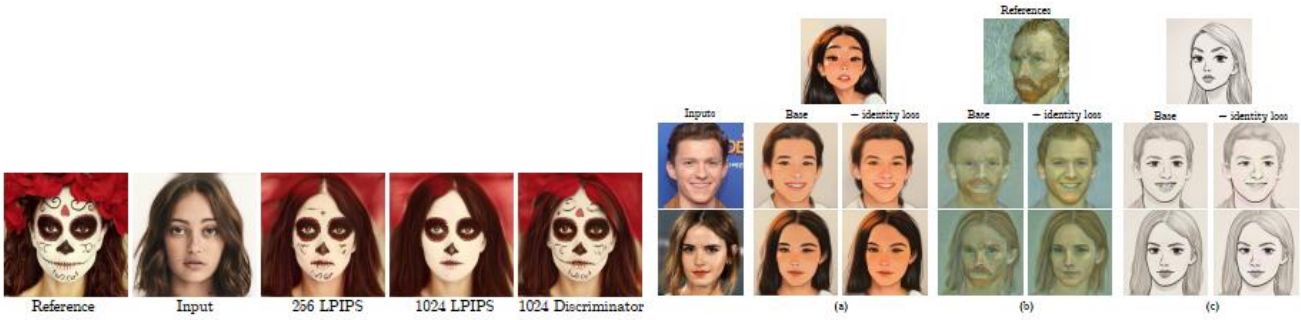
Trong đó $\mathcal{L}$ là một **perceptual loss**.

Bước 4: Inference(Suy luận):

- Đối với một đầu vào u bất kỳ, chúng ta áp dụng quá trình stylized lên đó để tạo ra khuôn mặt đã được phong cách hóa. Quá trình này được thực hiện bằng cách áp dụng tuần tự các phép biến đổi: mã code phong cách u được đưa qua quá trình GAN inversion để thu được mã code phong cách s(T(u)), sau đó mã code phong cách này được áp dụng vào mô hình StyleGAN đã được fine-tuning để tạo ra kết quả stylized G(s(T(u)); θˆ).

- Điều đáng chú ý là quá trình stylized này có thể được áp dụng cho bất kỳ đầu vào u nào, cho phép chúng ta áp dụng phong cách đã học được từ hình ảnh tham chiếu y lên các hình ảnh mới. Ngoài ra, chúng ta cũng có thể tạo ra các mẫu phong cách ngẫu nhiên bằng cách lấy mẫu nhiễu ngẫu nhiên và tạo hình ảnh stylized tương ứng bằng cách sử dụng mô hình StyleGAN đã được fine-tuning.

Perceptual Loss:



- Sự lựa chọn của hàm mất mát là một phần quan trọng trong quá trình huấn luyện JoJoGAN. Hàm mất mát LPIPS, dựa trên mô hình VGG, không đảm bảo giữ được chi tiết khi áp dụng đối với độ phân giải khác nhau. Đối với độ phân giải 256, việc giảm độ phân giải dẫn đến mất mát chi tiết. Còn đối với độ phân giải 1024, các bộ lọc VGG không được huấn luyện cho tỷ lệ này, dẫn đến việc không kiểm soát được chi tiết. Thay vào đó, chúng tôi sử dụng khớp hoạt động tại các lớp của bộ phân biệt đã được huấn luyện trước từ StyleGAN huấn luyện trên bộ dữ liệu FFHQ để tính toán hàm mất mát nhằm bảo tồn chi tiết tốt hơn. Bên cạnh đó, chúng tôi cũng sử dụng một hàm mất mát đơn giản về định danh để kiểm soát hiệu ứng mất đi đặc điểm cá nhân trong một số kiểu đầu vào phong cách.
- Bộ phân biệt StyleGAN được huấn luyện trước cùng độ phân giải với bộ tạo hình ảnh để đảm bảo tính chi tiết. Chúng tôi sử dụng sự khác biệt trong hoạt động của bộ phân biệt để tính toán mất mát sự tương đồng giữa các hình ảnh stylized và hình ảnh tham chiếu. Phương pháp này cho phép chúng tôi duy trì chi tiết trong quá trình tạo hình ảnh stylized. Mặc dù một phiên bản của hàm mất mát này đã được sử dụng trước đây, chúng tôi đã so sánh và chứng minh hiệu quả của nó so với các phương pháp khác.

Variants:

Controlling Identity (Kiểm soát định danh):

- Để đảm bảo rằng định danh không bị thay đổi quá nhiều, chúng tôi sử dụng một hàm mất mát được ký hiệu là Lid. Hàm mất mát này tính toán độ tương đồng cosin giữa các nhúng khuôn mặt của hình ảnh được tạo ra bằng mạng tạo ra phong cách (G(si; θ)) và các nhúng khuôn mặt của hình ảnh gốc sau khi qua mạng nhúng khuôn mặt đã được huấn luyện trước (F(G(si; θˆ))). Khi giá trị của hàm mất mát Lid là thấp, tức là độ tương đồng giữa các nhúng khuôn mặt là cao, điều này cho thấy rằng định

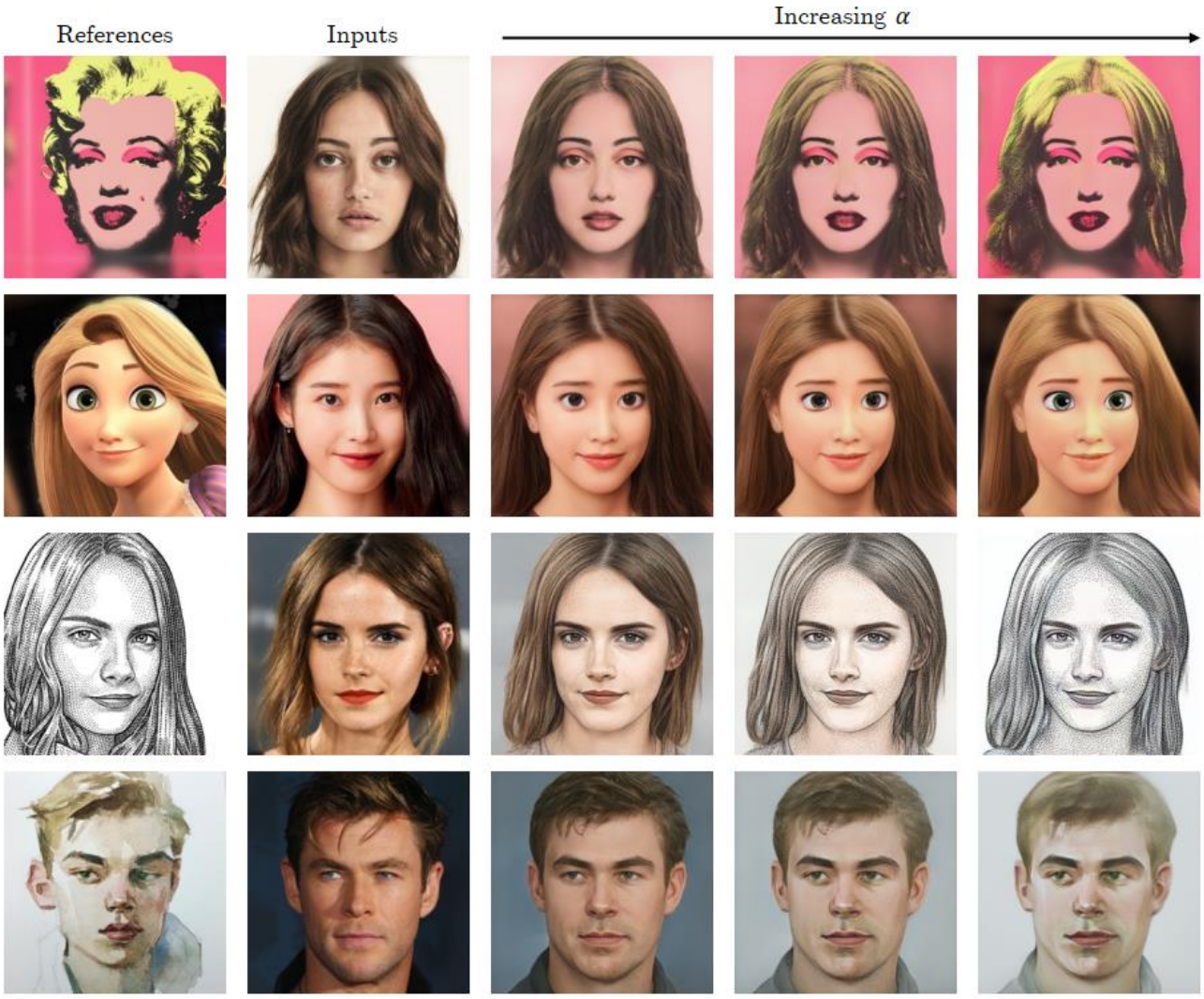
danh của hình ảnh được bảo tồn tốt.

$$\mathcal{L}_{id} = 1 - sim(F(G(s_i;\theta)), F(G(s_i;\hat{\theta})))$$

Controlling Style Intensity by Feature Interpolation (Kiểm soát độ mạnh của phong cách thông qua nội suy đặc trưng):

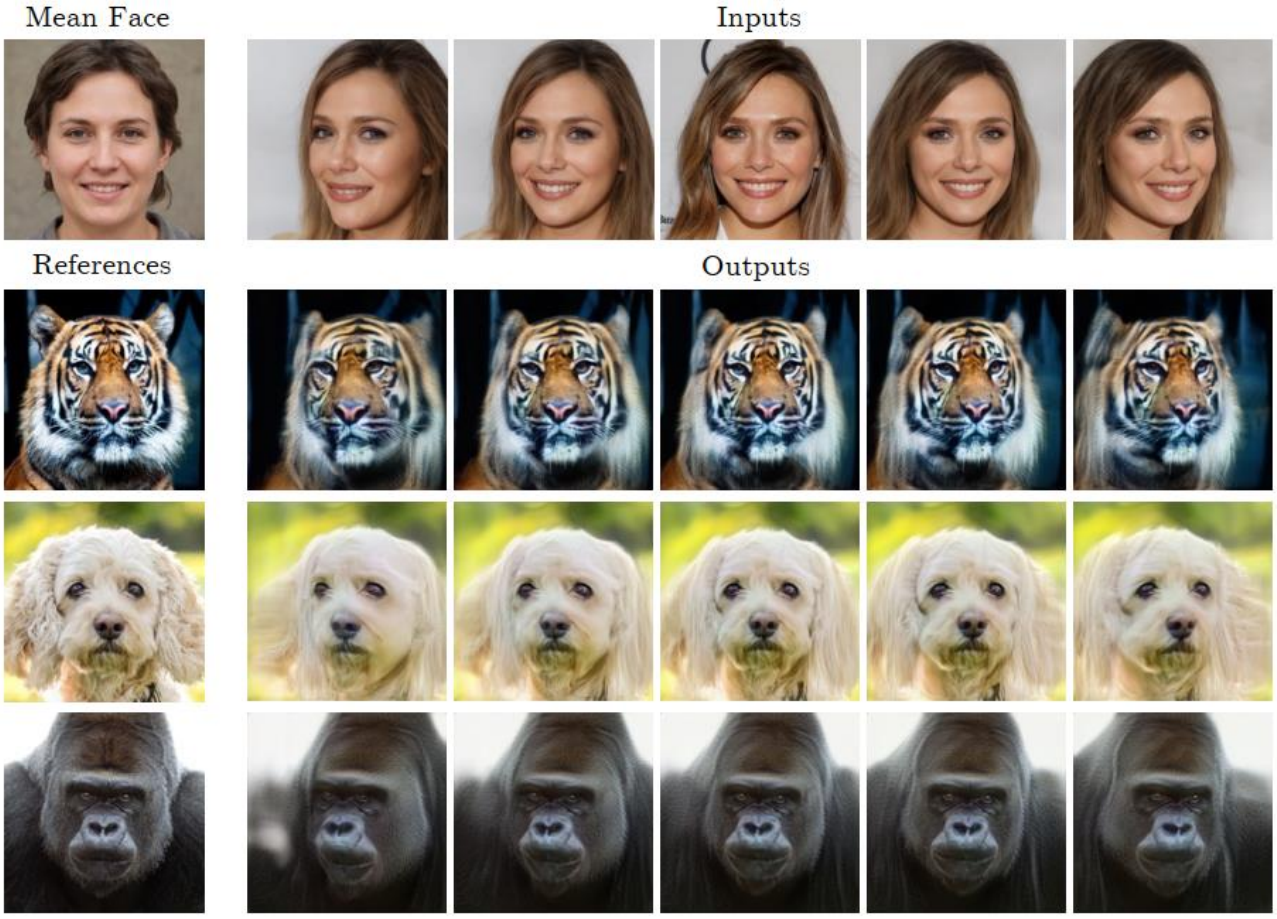
Giả sử các f_i^A là các bản đồ đặc trưng trung gian của lớp I từ StyleGAN ban đầu và f_i^B từ JoJoGAN. Chúng ta thực hiện việc tạo khuôn mặc bằng cách $f = (1-\alpha) f_i^A + \alpha f_i^B$

Trong đó alpha là yếu tố nội suy. Tăng giá trị của alpha sẽ làm tăng độ mạnh của phong cách.



Extreme Style Reference:

- Đối với JoJoGAN, để hoạt động tốt, chúng ta cần có tập S gồm các giá trị si mà StyleGAN có thể xử lý được. Tuy nhiên, đối với những hình ảnh tham chiếu phong cách cực đoan, ví dụ như hình ảnh khuôn mặt động vật, GAN inversion sẽ tạo ra giá trị s nằm ngoài phân phối của StyleGAN, dẫn đến việc truyền phong cách kém chất lượng. Để giải quyết vấn đề này, chúng tôi sử dụng một phương pháp khác cho tham chiếu phong cách cực đoan, bằng cách sử dụng mã phong cách trung bình s thay vì s(T(y)). Điều này giúp JoJoGAN hoạt động tốt trên các tham chiếu phong cách cực đoan và điều khiển được tư thế của đầu động vật trong kết quả truyền phong cách.



Multi-shot Stylization:

- JoJoGAN mở rộng đến việc thực hiện stylization đa bước một cách tự nhiên (sử dụng mỗi tham chiếu để xây dựng một Sk cho mỗi tham chiếu yk; sau đó, tiến hành fine-tune bằng cách sử dụng công thức:

$$\frac{1}{M * N} \sum_{j}^{M} \sum_{i}^{N} \mathcal{L}(G(s_{ij}; \theta), y_j).$$

Các bước cài đặt:

1. Cài đặt các thư viện và set up file:

```
#@title Setup. This will take a few minutes.
!git clone https://github.com/mchong6/JoJoGAN.git
%cd JoJoGAN
!pip install tqdm gdown scikit-learn==0.22 scipy lpips dlib opencv-python wandb
!wget https://github.com/ninja-build/ninja/releases/download/v1.8.2/ninja-linux.zip
!sudo unzip ninja-linux.zip -d /usr/local/bin/
!sudo update-alternatives --install /usr/bin/ninja ninja /usr/local/bin/ninja 1 --force

%load_ext autoreload
%autoreload 2

import torch
torch.backends.cudnn.benchmark = True
from torchvision import transforms, utils
from util import *
from PIL import Image
import math
import random
import os

import numpy as np
from torch import nn, autograd, optim
from torch.nn import functional as F
from tqdm import tqdm
import wandb
from model import *
from e4e_projection import projection as e4e_projection

from google.colab import files
from copy import deepcopy
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

os.makedirs('inversion_codes', exist_ok=True)
os.makedirs('style_images', exist_ok=True)
os.makedirs('style_images_aligned', exist_ok=True)
os.makedirs('models', exist_ok=True)
```

Các thư viện được cài đặt:

- tqdm: Một thư viện hỗ trợ hiển thị tiến trình trong quá trình chạy.
- gdown: Thư viện cho phép tải xuống tệp từ Google Drive.
- scikit-learn: Thư viện cho các công cụ học máy và phân tích dữ liệu.
- scipy: Thư viện cho các tính toán và phân tích khoa học.
- lpips: Thư viện cho đo lường độ tương đồng hình ảnh.
- dlib: Thư viện cho các công cụ nhận dạng khuôn mặt và xử lý hình ảnh.

- opencv-python: Thư viện xử lý hình ảnh và thao tác trên hình ảnh.
- wandb: Thư viện hỗ trợ ghi lại và theo dõi quá trình huấn luyện và đánh giá mô hình.

Các thư mục được tạo:

- inversion_codes: Thư mục để lưu trữ các mã nguồn liên quan đến quá trình nghịch đảo (inversion) trong JoJoGAN.
- style_images: Thư mục để lưu trữ các hình ảnh tham chiếu (style references).
- style_images_aligned: Thư mục để lưu trữ các hình ảnh tham chiếu đã được căn chỉnh (aligned).
- models: Thư mục để lưu trữ các mô hình của JoJoGAN.

2. Tải các model:

```
# from StyelGAN-NADA
class Downloader(object):
    def __init__(self, use_pydrive):
        self.use_pydrive = use_pydrive

        if self.use_pydrive:
            self.authenticate()

    def authenticate(self):
        auth.authenticate_user()
        gauth = GoogleAuth()
        gauth.credentials = GoogleCredentials.get_application_default()
        self.drive = GoogleDrive(gauth)

    def download_file(self, file_name):
        file_dst = os.path.join('models', file_name)
        file_id = drive_ids[file_name]
        if not os.path.exists(file_dst):
            print(f'Downloading {file_name}')
            if self.use_pydrive:
                downloaded = self.drive.CreateFile({'id':file_id})
                downloaded.FetchMetadata(fetch_all=True)
                downloaded.GetContentFile(file_dst)
            else:
                !gdown --id $file_id -O $file_dst
```

```
latent_dim = 512

# Load original generator
original_generator = Generator(1024, latent_dim, 8, 2).to(device)
ckpt = torch.load('models/stylegan2-ffhq-config-f.pt', map_location=lambda storage, loc: storage)
original_generator.load_state_dict(ckpt["g_ema"], strict=False)
mean_latent = original_generator.mean_latent(10000)

# to be finetuned generator
generator = deepcopy(original_generator)


transform = transforms.Compose(
    [
        transforms.Resize((1024, 1024)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ]
)
```

1. Tải xuống các mô hình từ Google Drive: Mã sử dụng downloader để tải xuống các mô hình từ Google Drive. Đầu tiên, mã khởi tạo một đối tượng downloader và xác thực với tài khoản Google Drive. Sau đó, mã sử dụng phương thức download file để tải xuống các mô hình từ Google Drive và lưu chúng vào thư mục "models".

2. Khởi tạo generator và tải mô hình gốc: Mã khởi tạo một generator ban đầu và tải mô hình generator gốc từ tệp tin "stylegan2-ffhq-config-f.pt". Mô hình generator gốc được lưu vào biến Original Generator, và generator được sao chép từ mô hình gốc vào biến Generator.
3. Chuẩn bị biến đổi hình ảnh: Mã định nghĩa một chuỗi các biến đổi hình ảnh được áp dụng trước khi sử dụng trong quá trình stylization. Các biến đổi bao gồm thay đổi kích thước hình ảnh thành (1024, 1024) và chuẩn hóa giá trị pixel của hình ảnh

Kết quả chạy:

```
--2023-05-17 19:38:08--  http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2
Resolving dlib.net (dlib.net)... 107.180.26.78
Connecting to dlib.net (dlib.net)|107.180.26.78|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 64040097 (61M)
Saving to: ‘shape_predictor_68_face_landmarks.dat.bz2.2’

shape_predictor_68_ 100%[===================>]  61.07M  26.8MB/s    in 2.3s

2023-05-17 19:38:11 (26.8 MB/s) - ‘shape_predictor_68_face_landmarks.dat.bz2.2’ saved [64040097/64040097]
```

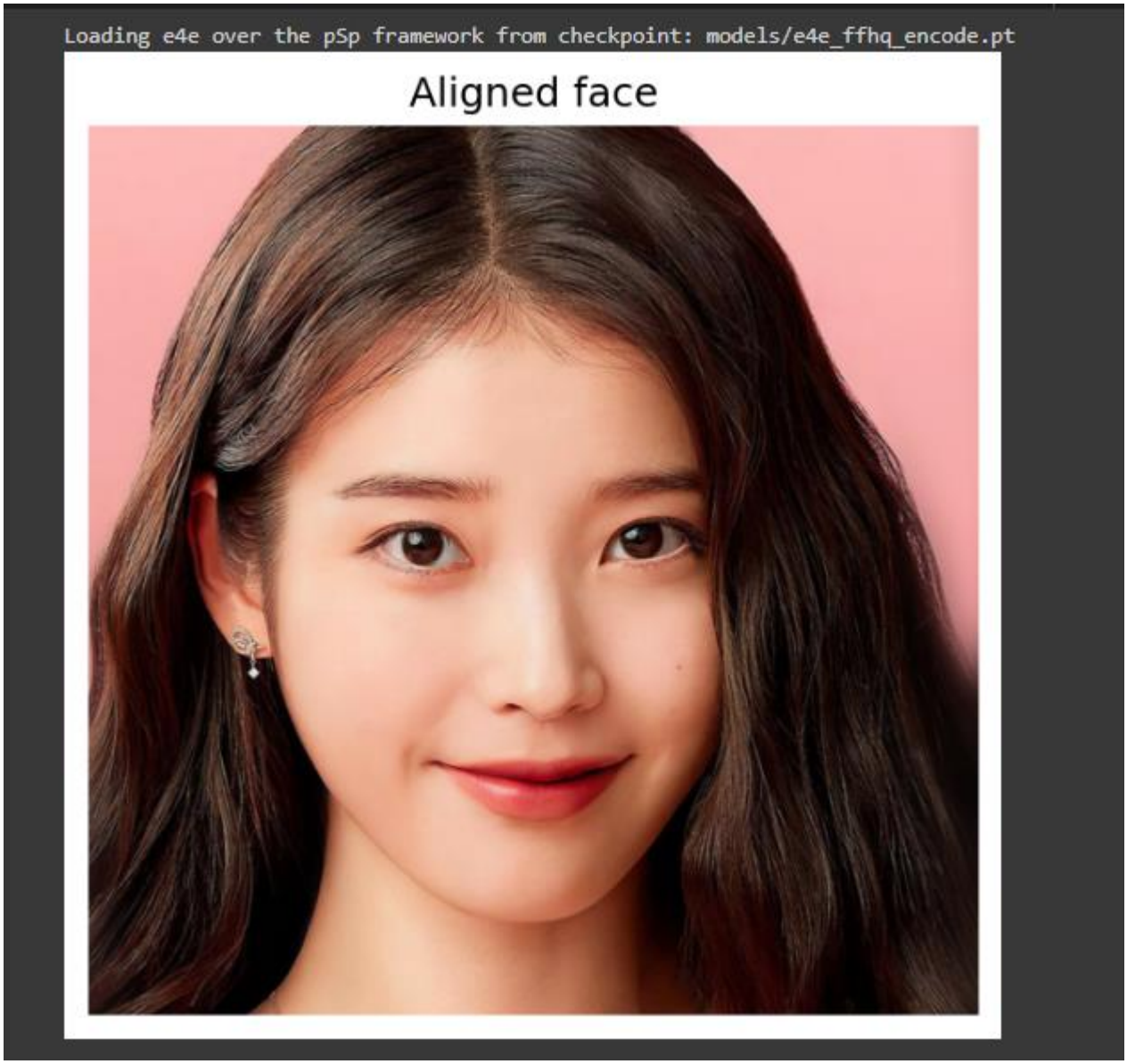
3. Chọn ảnh đầu vào:

```
plt.rcParams['figure.dpi'] = 150

#@title Choose input face
#@markdown Add your own image to the test_input directory and put the name here
filename = 'chris_hemsworth.jpeg' #@param {type:"string"}
filepath = f'test_input/{filename}'

# uploaded = files.upload()
# filepath = list(uploaded.keys())[0]
name = strip_path_extension(filepath)+'.pt'

# aligns and crops face
aligned_face = align_face(filepath)

# my_w = restyle_projection(aligned_face, name, device, n_iters=1).unsqueeze(0)
my_w = e4e_projection(aligned_face, name, device).unsqueeze(0)

display_image(aligned_face, title='Aligned face')
```

1. Thiết lập độ phân giải cho hình ảnh đầu ra: Mã sử dụng **`plt.rcParams['figure.dpi'] = 150`** để thiết lập độ phân giải cho hình ảnh đầu ra.
2. Chọn hình ảnh đầu vào: Mã yêu cầu người dùng cung cấp tên tệp tin hình ảnh đầu vào thông qua biến **`filename`**. Người dùng có thể thay đổi giá trị của **`filename`** để chỉ định tệp tin hình ảnh của riêng họ. Mã sẽ sử dụng đường dẫn tệp tin này để tải hình ảnh đầu vào từ thư mục "test_input".
3. Căn chỉnh và cắt hình ảnh khuôn mặt: Mã sử dụng hàm **`align_face`** để căn chỉnh và cắt hình ảnh khuôn mặt từ tệp tin hình ảnh đầu vào. Kết quả là một hình ảnh khuôn mặt đã được căn chỉnh và cắt gọn, được lưu trong biến **`aligned_face`**.

4. Xây dựng biểu diễn W cho hình ảnh đầu vào: Mã sử dụng phương thức **`e4e_projection`** để xây dựng biểu diễn W cho hình ảnh đã căn chỉnh. Biểu diễn W này được lưu trong biến **`my_w`**.
5. Hiển thị hình ảnh đã căn chỉnh: Mã sử dụng hàm **`display_image`** để hiển thị hình ảnh khuôn mặt đã được căn chỉnh (**`aligned_face`**) với tiêu đề "Aligned face".

Kết quả:



1. Thiết lập độ phân giải cho hình ảnh đầu ra: Mã sử dụng `plt.rcParams['figure.dpi'] = 150` để thiết lập độ phân giải cho hình ảnh đầu ra.

```
rcParams['figure.dpi'] = 150
```

2. Chọn mô hình được huấn luyện trước: Mã yêu cầu người dùng chọn một mô hình được huấn luyện trước từ danh sách các tùy chọn. Người dùng có thể chọn một trong các tùy chọn: 'art', 'arcane_multi', 'sketch_multi', 'arcane_jinx', 'arcane_caitlyn', 'jojo_yasuho', 'jojo', 'disney'. Giá trị của biến `pretrained` sẽ xác định mô hình được sử dụng.

```
pretrained = 'jojo' #@param ['art', 'arcane_multi', 'sketch_multi', 'arcane_jinx', 'arcane_caitlyn', 'jo
#@markdown Preserve color tries to preserve color of original image by limiting family of allowable tran
preserve_color = True #@param{type:"boolean"}
```

3. Lựa chọn bảo tồn màu sắc: Mã cho phép người dùng chọn xem liệu màu sắc của hình ảnh gốc có được bảo tồn hay không. Nếu `preserve_color` được chọn là True, thì màu sắc của hình ảnh gốc sẽ được bảo tồn trong quá trình áp dụng phong cách. Nếu `preserve_color` được chọn là False, thì màu sắc của hình ảnh đầu ra sẽ thừa hưởng từ hình ảnh tham chiếu, dẫn đến việc áp dụng phong cách nặng hơn.

```
if preserve_color:
    ckpt = f'{pretrained}_preserve_color.pt'
else:
    ckpt = f'{pretrained}.pt'
```

4. Tải mô hình đã được huấn luyện: Mã tải mô hình đã được huấn luyện dựa trên các thông số đã chọn. Nếu phiên bản bảo tồn màu sắc không khả dụng, mã sẽ tải phiên bản cơ bản của mô hình. Mã sử dụng phương thức `download_file` để tải xuống mô hình từ Google Drive.

```
# load base version if preserve_color version not available
try:
    downloader.download_file(ckpt)
except:
    ckpt = f'{pretrained}.pt'
    downloader.download_file(ckpt)

ckpt = torch.load(os.path.join('models', ckpt), map_location=lambda storage, loc: storage)
generator.load_state_dict(ckpt["g"], strict=False)
```

5. Tạo kết quả hình ảnh: Mã sử dụng mô hình được tải và các tham số cấu hình như số lượng mẫu (`n_sample`) và giá trị hạt giống (`seed`) để tạo ra các kết quả hình ảnh. Mã sử dụng hàm `torch.randn` để tạo ra các vectơ ngẫu nhiên (`z`) và sau đó sử dụng mô hình để tạo ra hình ảnh

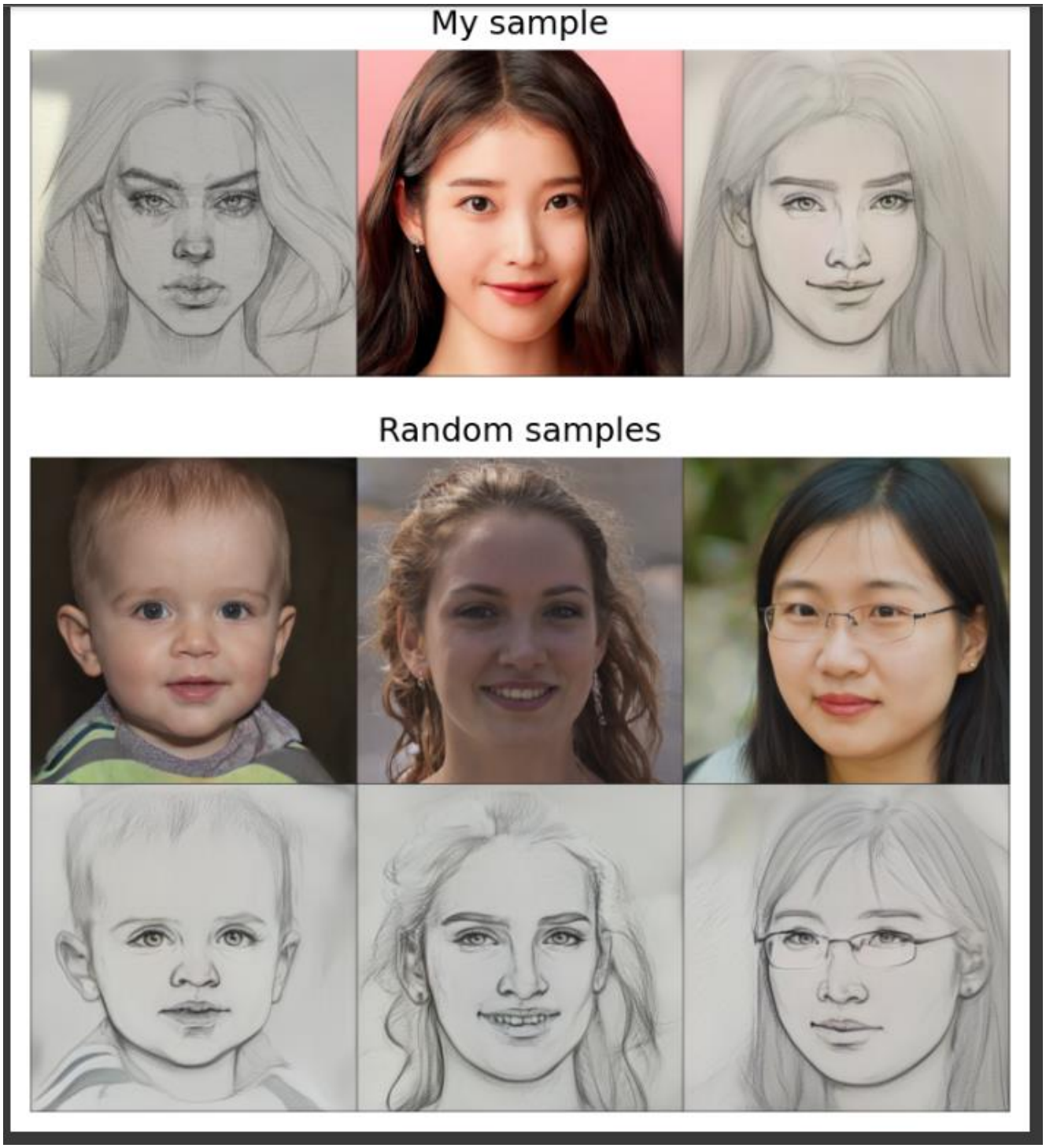
kết quả.

```
#@title Generate results
n_sample =  3#@param {type:"number"}
seed = 1000 #@param {type:"number"}

torch.manual_seed(seed)
with torch.no_grad():
    generator.eval()
    z = torch.randn(n_sample, latent_dim, device=device)

    original_sample = original_generator([z], truncation=0.7, truncation_latent=mean_latent)
    sample = generator([z], truncation=0.7, truncation_latent=mean_latent)

    original_my_sample = original_generator(my_w, input_is_latent=True)
    my_sample = generator(my_w, input_is_latent=True)
```

6. Hiển thị kết quả hình ảnh: Mã hiển thị các kết quả hình ảnh bao gồm mẫu ngẫu nhiên (**`output`**) và mẫu dựa trên hình ảnh đầu vào (**`my_output`**). Các hình ảnh được hiển thị bằng cách sử dụng hàm **`display_image`** và **`utils.make_grid`** để tạo lưới hình ảnh.

```
# display reference images
if pretrained == 'arcane_multi':
    style_path = f'style_images_aligned/arcane_jinx.png'
elif pretrained == 'sketch_multi':
    style_path = f'style_images_aligned/sketch.png'
else:
    style_path = f'style_images_aligned/{pretrained}.png'

style_image = transform(Image.open(style_path)).unsqueeze(0).to(device)
face = transform(aligned_face).unsqueeze(0).to(device)

my_output = torch.cat([style_image, face, my_sample], 0)
display_image(utils.make_grid(my_output, normalize=True, range=(-1, 1)), title='My sample')

output = torch.cat([original_sample, sample], 0)
display_image(utils.make_grid(output, normalize=True, range=(-1, 1), nrow=n_sample), title='Random sampl
```

Kết quả chạy:



4. Lựa chọn phong cách:

1. Lựa chọn hình ảnh phong cách: Mã yêu cầu người dùng tải lên các hình ảnh phong cách vào thư mục `style_images` và nhập tên của chúng vào trường tương ứng. Người dùng có thể tải

lên nhiều hình ảnh phong cách bằng cách nhập vào danh sách các tên hình ảnh.

```
#@markdown Upload your own style images into the style_images folder and type it into the field in the
names = ['sketch.jpeg', 'sketch2.jpeg', 'sketch3.jpeg', 'sketch4.jpeg'] #@param {type:"raw"}

targets = []
latents = []
```

2. Tiến hành xử lý hình ảnh phong cách: Mã lặp qua danh sách các tên hình ảnh và thực hiện các bước xử lý sau:
    a. Kiểm tra xem hình ảnh phong cách có tồn tại hay không. Nếu không tồn tại, mã sẽ đưa ra thông báo lỗi.
    b. Cắt và căn chỉnh khuôn mặt: Mã sử dụng hàm **`align_face`** để cắt và căn chỉnh khuôn mặt trong hình ảnh phong cách. Kết quả được lưu vào thư mục **`style_images_aligned`** với tên tương ứng.
    c. GAN invert: Mã sử dụng hàm **`e4e_projection`** để tạo mã nguồn ngẫu nhiên cho hình ảnh phong cách. Mã nguồn ngẫu nhiên được lưu vào thư mục **`inversion_codes`** với tên tương ứng.

```
for name in names:
    style_path = os.path.join('style_images', name)
    assert os.path.exists(style_path), f"{style_path} does not exist!"

    name = strip_path_extension(name)

    # crop and align the face
    style_aligned_path = os.path.join('style_images_aligned', f'{name}.png')
    if not os.path.exists(style_aligned_path):
        style_aligned = align_face(style_path)
        style_aligned.save(style_aligned_path)
    else:
        style_aligned = Image.open(style_aligned_path).convert('RGB')

    # GAN invert
    style_code_path = os.path.join('inversion_codes', f'{name}.pt')
    if not os.path.exists(style_code_path):
        latent = e4e_projection(style_aligned, style_code_path, device)
    else:
        latent = torch.load(style_code_path)['latent']

    targets.append(transform(style_aligned).to(device))
    latents.append(latent.to(device))
```

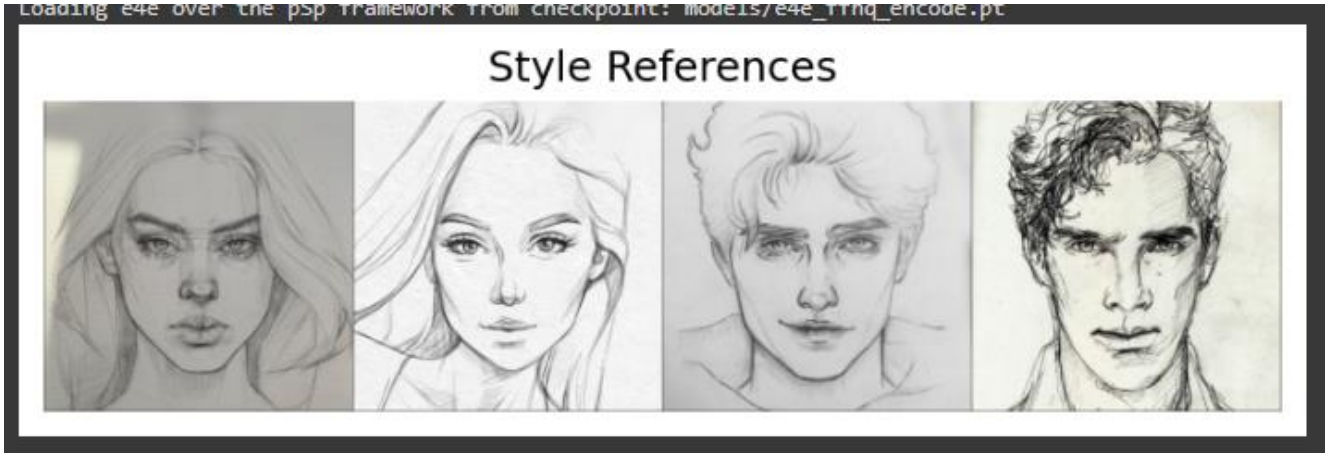
3. Tạo biểu diễn hình ảnh phong cách: Mã tạo các biểu diễn của các hình ảnh phong cách bằng cách chuyển đổi chúng thành dạng tensor và xếp chúng thành một tensor lớn. Các biểu diễn hình ảnh phong cách được lưu vào biến **`targets`** và **`latents`**.

```
targets = torch.stack(targets, 0)
latents = torch.stack(latents, 0)
```

4. Hiển thị hình ảnh phong cách: Mã sử dụng hàm **`utils.make_grid`** để tạo lưới hình ảnh của các hình ảnh phong cách và hiển thị nó bằng hàm **`display_image`**.

```
target_im = utils.make_grid(targets, normalize=True, range=(-1, 1))
display_image(target_im, title='Style References')
```

Kết quả:



5. Finetuning StyleGAN:

1. Điều chỉnh mức độ của phong cách: Mã sử dụng thanh trượt **`alpha`** để điều chỉnh mức độ của phong cách. Giá trị **`alpha`** được chọn từ 0 đến 1 và sau đó được chuyển đổi thành giá trị tương ứng để điều chỉnh mức độ phong cách.

```
#@title Finetune StyleGAN
#@markdown alpha controls the strength of the style
alpha = 1 #@param {type:"slider", min:0, max:1.01, step:0.1}
alpha = 1-alpha
```

2. Lựa chọn bảo tồn màu sắc: Mã cho phép người dùng chọn liệu có bảo tồn màu sắc từ hình ảnh gốc hay không bằng cách đặt giá trị **`preserve_color`**. Nếu **`preserve_color`** được đặt là True, màu sắc của hình ảnh gốc sẽ được bảo tồn trong quá trình tinh chỉnh, dẫn đến mức độ phong cách mạnh hơn. Nếu **`preserve_color`** được đặt là False, màu sắc của hình ảnh phong cách sẽ được chuyển từ hình ảnh tham chiếu.

3. Số lần tinh chỉnh: Mã yêu cầu người dùng chỉ định số lần tinh chỉnh **`num_iter`** để tạo ra kết quả tinh chỉnh. Số lần tinh chỉnh khác nhau có thể được yêu cầu cho các hình ảnh phong cách khác nhau.

```
#@markdown Tries to preserve color of original image by limiting family of allowable transformations. Se
preserve_color = True #@param{type:"boolean"}
#@markdown Number of finetuning steps. Different style reference may require different iterations. Try 2
num_iter = 300 #@param {type:"number"}
#@markdown Log training on wandb and interval for image logging
use_wandb = False #@param {type:"boolean"}
log_interval = 50 #@param {type:"number"}
```

4. Ghi nhật ký và hiển thị hình ảnh: Mã cung cấp tùy chọn sử dụng **wandb** để ghi nhật ký và hiển thị quá trình tinh chỉnh. Nếu **use_wandb** được đặt là True, mã sẽ khởi tạo **wandb** và ghi nhật ký các thông số cấu hình và hình ảnh tham chiếu ban đầu. Mã sẽ cũng ghi nhật ký giá trị **loss** sau mỗi vòng lặp và hiển thị hình ảnh kết quả tinh chỉnh hiện tại sau mỗi **log_interval** vòng lặp.

```
if use_wandb:
    wandb.init(project="JoJoGAN")
    config = wandb.config
    config.num_iter = num_iter
    config.preserve_color = preserve_color
    wandb.log(
    {"Style reference": [wandb.Image(transforms.ToPILImage()(target_im))]},
    step=0)

# load discriminator for perceptual loss
discriminator = Discriminator(1024, 2).eval().to(device)
ckpt = torch.load('models/stylegan2-ffhq-config-f.pt', map_location=lambda storage, loc: storage)
discriminator.load_state_dict(ckpt["d"], strict=False)
```

5. Tạo lại mô hình và tối ưu hóa: Mã tải lại mô hình StyleGAN và tạo bản sao của mô hình gốc. Mã sử dụng tối ưu hóa Adam với tốc độ học 2e-3 và các siêu tham số **betas=(0, 0.99)** để tối ưu hóa các tham số của mô hình. Mã lặp qua các vòng lặp và thực hiện các bước sau:
    a. Tạo biểu diễn ngẫu nhiên cho phong cách: Mã tạo biểu diễn ngẫu nhiên cho phong cách bằng cách kết hợp biểu diễn ngẫu nhiên của hình ảnh phong cách và một biểu diễn ngẫu nhiên trung bình. Mức độ phong cách được điều chỉnh bằng cách kết hợp hai biểu diễn này dựa trên giá trị **alpha**.
    b. Tạo hình ảnh tinh chỉnh: Mã sử dụng mô hình tinh chỉnh để tạo ra hình ảnh tinh chỉnh dựa trên biểu diễn phong cách đã được điều chỉnh.
    c. Tính toán hàm mất mát: Mã tính toán hàm mất mát bằng cách so sánh đặc trưng của hình ảnh tinh chỉnh với đặc trưng của hình ảnh phong cách tham chiếu. Hàm mất mát được tính bằng cách tính tổng các lỗi trung bình tuyệt đối giữa các đặc trưng.
    d. Tối ưu hóa mô hình: Mã sử dụng tối ưu hóa Adam để điều chỉnh các tham số của mô hình tinh chỉnh dựa trên hàm mất mát đã tính toán.
    e. Ghi nhật ký và hiển thị kết quả: Nếu sử dụng **wandb**, mã ghi nhật ký giá trị **loss** sau mỗi vòng lặp và hiển thị hình ảnh tinh chỉnh hiện tại.

```
# reset generator
del generator
generator = deepcopy(original_generator)

g_optim = optim.Adam(generator.parameters(), lr=2e-3, betas=(0, 0.99))

# Which layers to swap for generating a family of plausible real images -> fake image
if preserve_color:
    id_swap = [9,11,15,16,17]
else:
    id_swap = list(range(7, generator.n_latent))

for idx in tqdm(range(num_iter)):
    mean_w = generator.get_latent(torch.randn([latents.size(0), latent_dim]).to(device)).unsqueeze(1).re
    in_latent = latents.clone()
    in_latent[:, id_swap] = alpha*latents[:, id_swap] + (1-alpha)*mean_w[:, id_swap]

    img = generator(in_latent, input_is_latent=True)

    with torch.no_grad():
        real_feat = discriminator(targets)
    fake_feat = discriminator(img)

    loss = sum([F.l1_loss(a, b) for a, b in zip(fake_feat, real_feat)])/len(fake_feat)

    if use_wandb:
        wandb.log({"loss": loss}, step=idx)
        if idx % log_interval == 0:
            generator.eval()
            my_sample = generator(my_w, input_is_latent=True)
            generator.train()
            my_sample = transforms.ToPILImage()(utils.make_grid(my_sample, normalize=True, range=(-1, 1)
            wandb.log(
            {"Current stylization": [wandb.Image(my_sample)]},
            step=idx)

    g_optim.zero_grad()
    loss.backward()
    g_optim.step()
```

Tạo ra kết quả sau cùng:

1. Tạo ra các mẫu ngẫu nhiên: Mã sử dụng hàm ngẫu nhiên **`torch.randn`** để tạo ra **`n_sample`** mẫu ngẫu nhiên với độ lớn **`latent_dim`**. Mỗi mẫu ngẫu nhiên là một điểm trong không gian ngẫu nhiên của mô hình StyleGAN.

2. Tạo ra kết quả tinh chỉnh: Mã sử dụng mô hình tinh chỉnh để tạo ra kết quả tinh chỉnh từ các mẫu ngẫu nhiên đã tạo ra. Kết quả tinh chỉnh được tạo ra bằng cách đưa các mẫu ngẫu nhiên vào mô hình và sử dụng tham số **`truncation`** và **`truncation_latent`** để kiểm soát mức độ của

kết quả.

```
n_sample =  5#@param {type:"number"}
seed = 3000 #@param {type:"number"}

torch.manual_seed(seed)
with torch.no_grad():
    generator.eval()
    z = torch.randn(n_sample, latent_dim, device=device)

    original_sample = original_generator([z], truncation=0.7, truncation_latent=mean_latent)
    sample = generator([z], truncation=0.7, truncation_latent=mean_latent)

    original_my_sample = original_generator(my_w, input_is_latent=True)
    my_sample = generator(my_w, input_is_latent=True)
```

3. Hiển thị hình ảnh tham chiếu: Mã hiển thị các hình ảnh phong cách tham chiếu mà người dùng đã tải lên. Các hình ảnh được hiển thị dưới dạng lưới sử dụng hàm **`make_grid`** trong thư viện **`torchvision.utils`**.
4. Hiển thị kết quả tinh chỉnh của mình: Mã hiển thị kết quả tinh chỉnh của mình bằng cách ghép các hình ảnh cùng một khuôn mặt (hình ảnh khuôn mặt được điều chỉnh và kết quả tinh chỉnh) thành một lưới và hiển thị nó.
5. Hiển thị các mẫu ngẫu nhiên: Mã hiển thị các mẫu ngẫu nhiên được tạo ra từ mô hình. Các mẫu ngẫu nhiên được hiển thị dưới dạng lưới sử dụng hàm **`make_grid`**. Các kết quả này đại diện cho các biến thể ngẫu nhiên của phong cách dựa trên mẫu ngẫu nhiên ban đầu.

```
# display reference images
style_images = []
for name in names:
    style_path = f'style_images_aligned/{strip_path_extension(name)}.png'
    style_image = transform(Image.open(style_path))
    style_images.append(style_image)

face = transform(aligned_face).to(device).unsqueeze(0)
style_images = torch.stack(style_images, 0).to(device)
display_image(utils.make_grid(style_images, normalize=True, range=(-1, 1)), title='References')

my_output = torch.cat([face, my_sample], 0)
display_image(utils.make_grid(my_output, normalize=True, range=(-1, 1)), title='My sample')

output = torch.cat([original_sample, sample], 0)
display_image(utils.make_grid(output, normalize=True, range=(-1, 1), nrow=n_sample), title='Random samp
```

Kết quả sau cùng:

References

My sample

Random samples

Các khó khăn và thử thách khi chạy cài đặt:

- Thời gian để tải cái set up file và tải các model là khá lâu.

- Cần phải chạy bằng cuda trong bước download model mới có thể thực hiện các bước sau. Nếu chạy bằng CPU ở bước download models thì ở bước Finetuning sẽ bị lỗi ở hàm discriminator không chạy được.

- Dung lượng cần để chạy file là khá nhiều.

Nhận xét các kết quả:
- Khi chạy được các kết quả cho ra là khả quan trong từng bước và khá chính xác trong các kết quả.

- Song với đó là thời gian chạy lâu có thể dẫn đến giảm sự kiên nhẫn của người dùng không chuyên.

References:

https://github.com/mchong6/JoJoGAN

https://arxiv.org/abs/2112.11641

ChatGPT2023