HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO THỰC TẬP CƠ SỞ

ĐỀ TÀI:TỔNG HỢP VÀ CHỈNH SỬA ẢNH DỰA TRÊN MÔ HÌNH ANYCOST GAN

Giảng viên hướng dẫn : **Ths. Bùi Văn Kiên**
Sinh viên : **Nguyễn Thanh Tùng**
MSV : **B21DCCN771**

Hà Nội – 2025

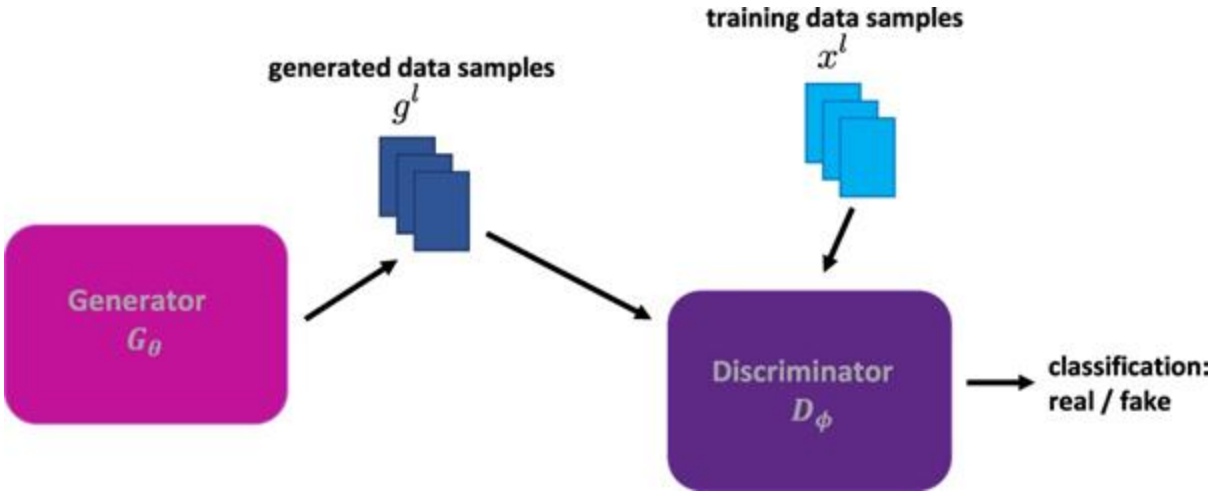
CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 3
1.1. Tổng quan về GAN và Anycost GAN 3
1.2. Cơ chế hoạt động 4
1.3. Ưu điểm và ứng dụng 4
1.4. Đánh giá chất lượng 5
CHƯƠNG 2. KIẾN TRÚC HỆ THỐNG 6
2.1. Kiến trúc tổng quan 6
2.2. Giao diện người dùng (PyQt5) 6
2.3. Luồng hoạt động tổng quát 11
CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG 12
3.1.Ngôn ngữ lập trình Python 12
3.2. PyTorch 12
3.3. CUDA/cuDNN 12
3.4. Horovod 13
3.5. NumPy 13
3.6. Torchvision 14
CHƯƠNG 4. TRIỂN KHAI MÔ HÌNH 15
4.1. Triển Khai Mô Hình GAN 15
4.1.1. Bộ Sinh Ảnh (Generator) 15
4.1.2. Bộ phân biệt (Discriminator) 16
4.1.3. Bộ Mã Hóa (Encoder) 18
4.1.4. Xử lý dữ liệu CelebA-HQ và trích xuất đặc trưng từ ảnh 19
4.2. Triển Khai Ứng Dụng Chỉnh Sửa Khuôn Mặt Sử Dụng Anycost GAN 23
TÀI LIỆU THAM KHẢO 25

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

Trong những năm gần đây, sự phát triển của trí tuệ nhân tạo (AI) và học sâu (deep learning) đã mang đến những bước tiến vượt bậc trong nhiều lĩnh vực, đặc biệt là trong xử lý và tạo dựng hình ảnh. Một trong những xu hướng nổi bật là việc sử dụng Mạng Sinh Generative Adversarial Networks (GANs) để tạo ra các hình ảnh mới hoặc chỉnh sửa các bức ảnh hiện có một cách tự động. Trong số các mô hình GAN tiên tiến, Anycost GAN đã trở thành một trong những giải pháp tiên phong, nổi bật với khả năng tổng hợp và chỉnh sửa ảnh hiệu quả ngay cả khi có tài nguyên tính toán hạn chế. Dự án này chúng em sẽ đi sâu vào việc áp dụng mô hình Anycost GAN để tổng hợp và chỉnh sửa ảnh cũng như là khám phá những lợi ích mà mô hình này mang lại

1.1. Tổng quan về GAN và Anycost GAN

Mạng đối nghịch tạo sinh (Generative Adversarial Networks -GAN) là một mô hình học sâu nổi bật, hoạt động dựa trên nguyên tắc đối kháng giữa 2 mạng nơ ron nhân tạo:



- Mạng tạo sinh (Generator - G): Có nhiệm vụ tạo ra dữ liệu giả nhưng trông giống dữ liệu thật.
- Mạng phân biệt (Discriminator - D): Có nhiệm vụ phân biệt dữ liệu thật từ dữ liệu giả do Generator tạo ra.

Hai mạng này được huấn luyện đồng thời trong một quá trình liên tục, Generator cố gắng đánh lừa Discriminator, trong khi Discriminator cố gắng phân biệt chính xác dữ liệu thật và dữ liệu giả. Qua thời gian, Generator sẽ cải thiện khả năng tạo dữ liệu sao cho ngày càng chân thực hơn.

GAN có thể tạo ra các hình ảnh cực kỳ chân thực từ dữ liệu đầu vào, điều này đã mở ra nhiều cơ hội mới trong các ứng dụng như tổng hợp ảnh, cải tiến chất lượng hình ảnh, và các tác vụ sáng tạo khác. Tuy nhiên, một trong những thách thức lớn khi áp dụng GAN trong thực tế là yêu cầu tài nguyên tính toán rất lớn, đặc biệt khi tạo ra các hình ảnh có độ phân giải cao.

Anycost GAN là một mô hình cải tiến từ GAN truyền thống, với mục tiêu tối ưu hóa chi phí tính toán trong quá trình huấn luyện và suy luận mà vẫn duy trì chất lượng hình ảnh ở mức cao. Bằng cách điều chỉnh và tối ưu hóa các tham số tính toán, Anycost GAN có thể hoạt động hiệu quả ngay cả trên các thiết bị với tài nguyên phần cứng hạn chế, từ đó mở rộng khả năng ứng dụng mô hình trong nhiều môi trường thực tế, như thiết bị di động hoặc các ứng dụng đám mây với chi phí tính toán tiết kiệm.

1.2. Cơ chế hoạt động

Anycost GAN sử dụng hai cơ chế chính để tối ưu hóa quá trình huấn luyện và tạo ảnh:

- **Huấn luyện đa độ phân giải (multi-resolution training)**: Cơ chế này cho phép mô hình học cách tạo ảnh ở nhiều mức độ phân giải khác nhau. Thay vì huấn luyện mô hình chỉ với một độ phân giải cố định, Anycost GAN có thể huấn luyện trên nhiều mức phân giải và linh hoạt thay đổi giữa các mức phân giải khi cần thiết. Điều này giúp giảm thời gian huấn luyện và tăng tốc độ tạo ảnh mà không làm giảm chất lượng hình ảnh.
- **Điều chỉnh kênh thích ứng (adaptive channel pruning):** Một trong những cải tiến quan trọng của Anycost GAN là khả năng tự động điều chỉnh số lượng kênh tính toán trong mạng dựa trên tài nguyên phần cứng hiện có. Điều này giúp mô hình hoạt động hiệu quả hơn trên các thiết bị có phần cứng hạn chế, đồng thời vẫn duy trì được chất lượng hình ảnh đầu ra ở mức chấp nhận được.

Các cơ chế này không chỉ giúp tối ưu hóa tài nguyên tính toán mà còn làm cho mô hình linh hoạt hơn khi triển khai trên các thiết bị có phần cứng khác nhau, từ các máy tính để bàn mạnh mẽ cho đến các thiết bị di động có cấu hình thấp.

1.3. Ưu điểm và ứng dụng

- **Xem trước nhanh và hiệu quả**:Người dùng có thể xem trước hình ảnh ở độ phân giải thấp mà không tốn nhiều tài nguyên tính toán. Sau đó, mô hình sẽ tạo phiên bản cuối cùng với độ phân giải cao nhất, giúp tiết kiệm thời gian và tài nguyên trong quá trình tạo ảnh.

- **Tích hợp dễ dàng vào các ứng dụng thực tế**:Mô hình này hỗ trợ nhiều kiến trúc huấn luyện sẵn, cho phép triển khai nhanh chóng vào các ứng dụng thực tế

như chỉnh sửa ảnh, tạo ảnh từ mô tả, và nâng cấp chất lượng ảnh. Điều này làm cho Anycost GAN trở thành một công cụ hữu ích cho các nhà phát triển trong việc tạo ra các sản phẩm ứng dụng mạnh mẽ.

- **Ứng dụng đa dạng**:
  Chỉnh sửa ảnh chân dung: Anycost GAN có thể sử dụng để chỉnh sửa các bức ảnh chân dung, ví dụ như thay đổi các đặc điểm như nụ cười, màu tóc, hoặc thêm các chi tiết trang điểm.

  Tạo ảnh từ dữ liệu đầu vào: Mô hình có thể sinh ra các bức ảnh từ dữ liệu đầu vào có sẵn, ví dụ như từ các mô tả văn bản hoặc dữ liệu đầu vào hình ảnh.

  Nâng cấp chất lượng ảnh: Anycost GAN giúp nâng cấp độ phân giải và chi tiết của các bức ảnh có độ phân giải thấp, cải thiện chất lượng ảnh trong các ứng dụng đòi hỏi hình ảnh rõ nét.

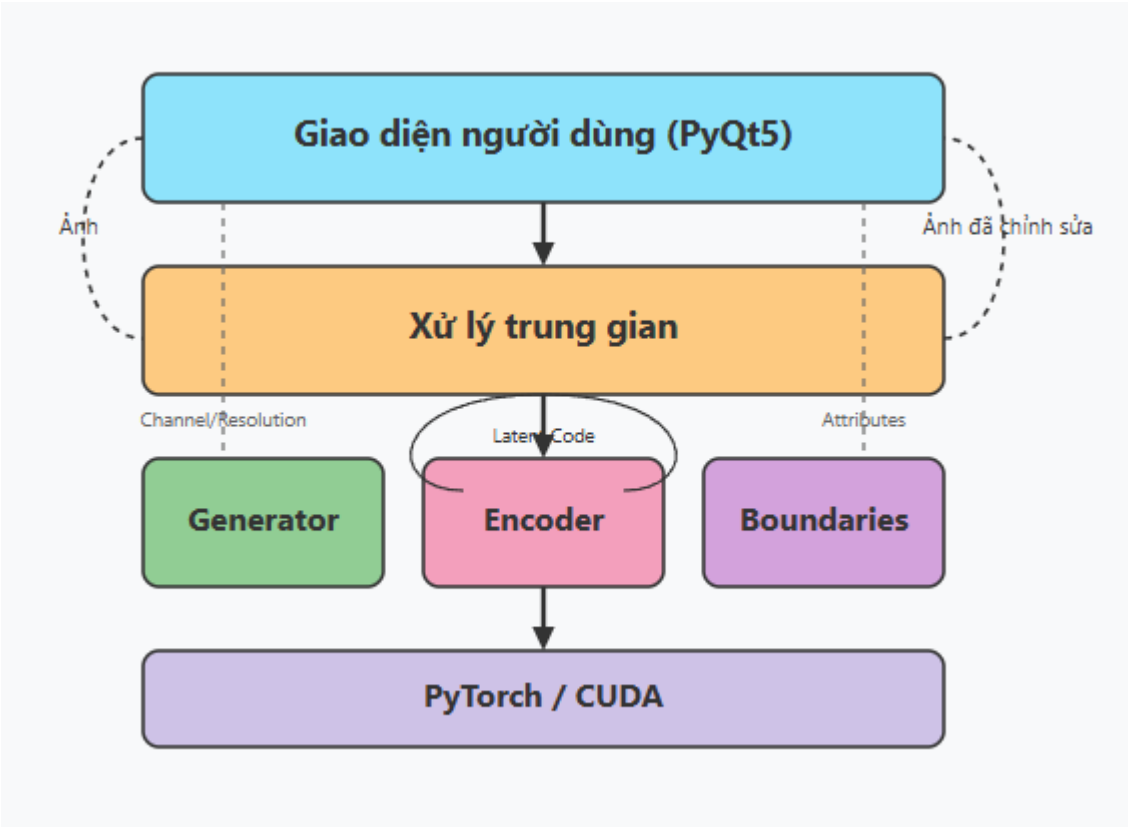
1.4. Đánh giá chất lượng

Để đánh giá chất lượng hình ảnh tạo ra từ Anycost GAN, hệ thống tích hợp các chỉ số đánh giá phổ biến như:

- **Fréchet Inception Distance (FID):** Đánh giá độ chân thực của hình ảnh.
- **Perceptual Path Length (PPL**): Kiểm tra mức độ ổn định và khả năng điều khiển mô hình

CHƯƠNG 2. KIẾN TRÚC HỆ THỐNG

2.1. Kiến trúc tổng quan



2.2. Giao diện người dùng (PyQt5)

- Hiển thị ảnh gốc và ảnh sau chỉnh sửa.
- Cung cấp thanh trượt để điều chỉnh các thuộc tính khuôn mặt như biểu cảm, màu tóc, độ rộng mắt,...
- Gồm các nút chức năng như Reset, Finalize, giúp người dùng kiểm soát quá trình chỉnh sửa.

2.3. Lớp xử lý trung gian (Middleware)

- Điều phối toàn bộ luồng xử lý giữa giao diện và mô hình GAN.
- Sử dụng các lớp như FaceEditor và Worker để mã hóa ảnh đầu vào, cập nhật latent code theo điều chỉnh của người dùng.
- Trích xuất hoặc xử lý các thuộc tính cụ thể từ dữ liệu, có thể dùng để phân tích hoặc hiển thị thông tin chi tiết.

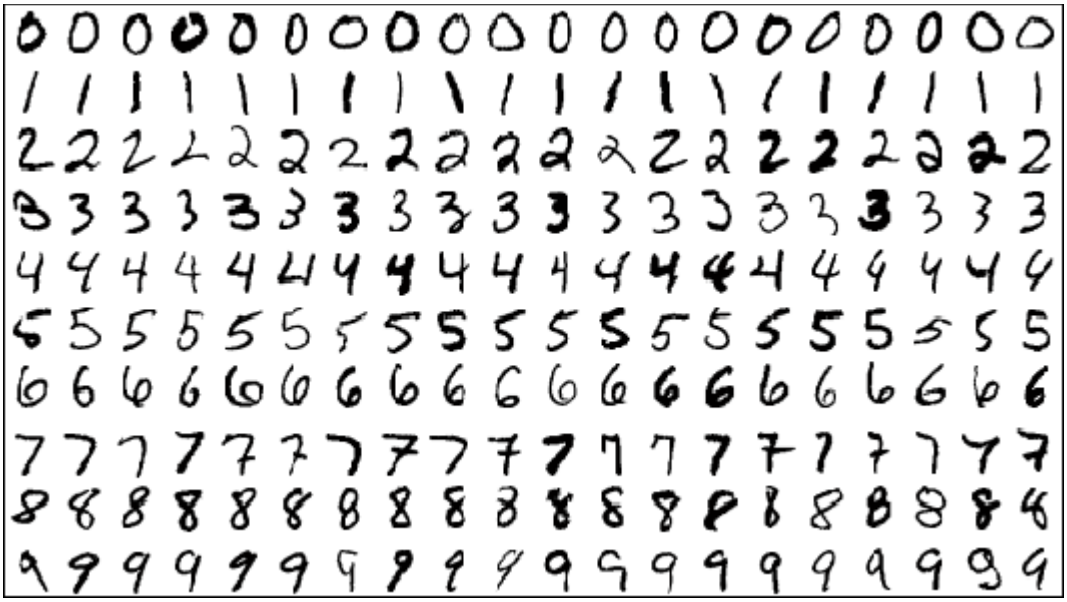
- **Channel/Resolution**: Xử lý thông tin về kênh dữ liệu (ví dụ: RGB, grayscale) và độ phân giải (resolution) của dữ liệu đầu vào.
- Quản lý luồng đa nhiệm thông qua QThreadPool, đảm bảo ứng dụng luôn phản hồi nhanh.

❖ Mô hình học sâu

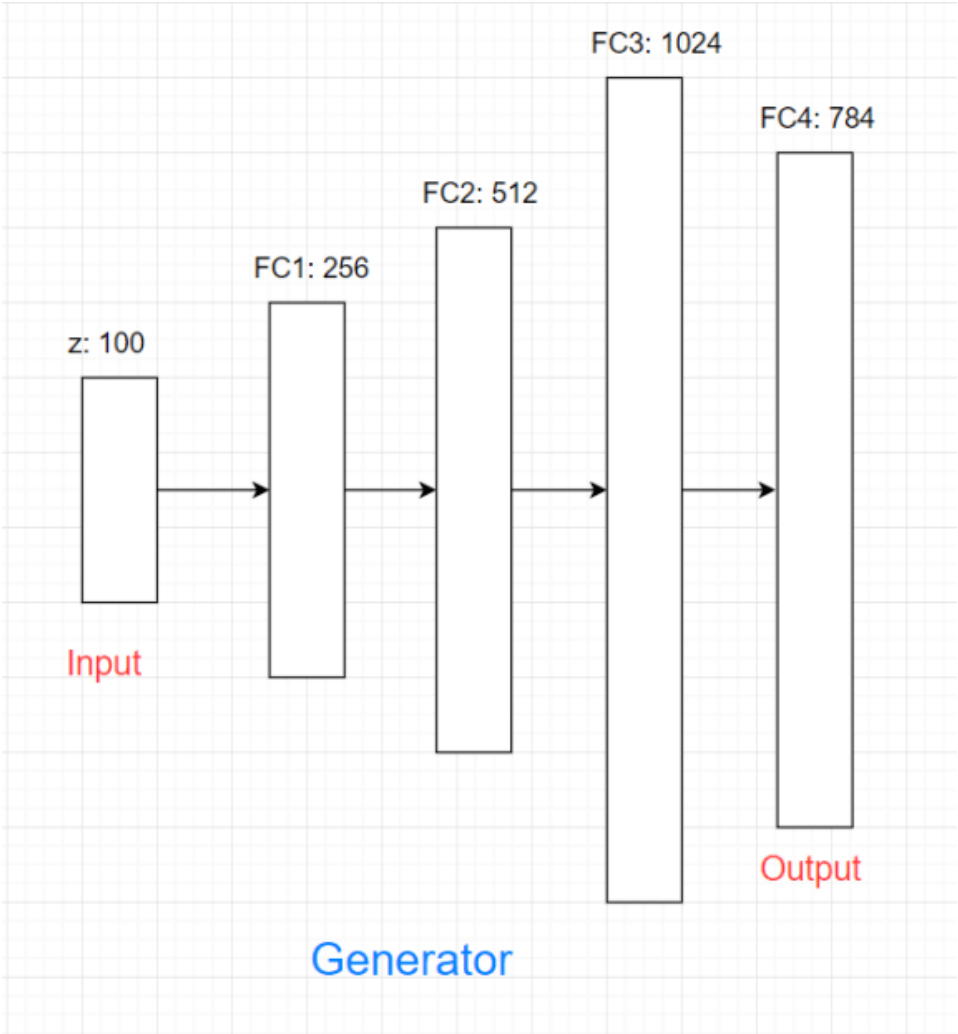
a. Cấu trúc mạng

- **Generator**

  Generator sinh ra các chữ số giống với dữ liệu trong MNIST dataset.
  Generator có input là noise (random vector) là output là chữ số.
  Input của Generator là noise để khi ta thay đổi noise ngẫu nhiên thì Generator có thể sinh ra một biến dạng khác của chữ viết tay. Noise cho Generator thường được sinh ra từ normal distribution hoặc uniform distribution.



  Input của Generator là noise vector 100 chiều.
  Sau đấy mô hình neural network được áp dụng với số node trong hidden layer lần lượt là 256, 512, 1024.`

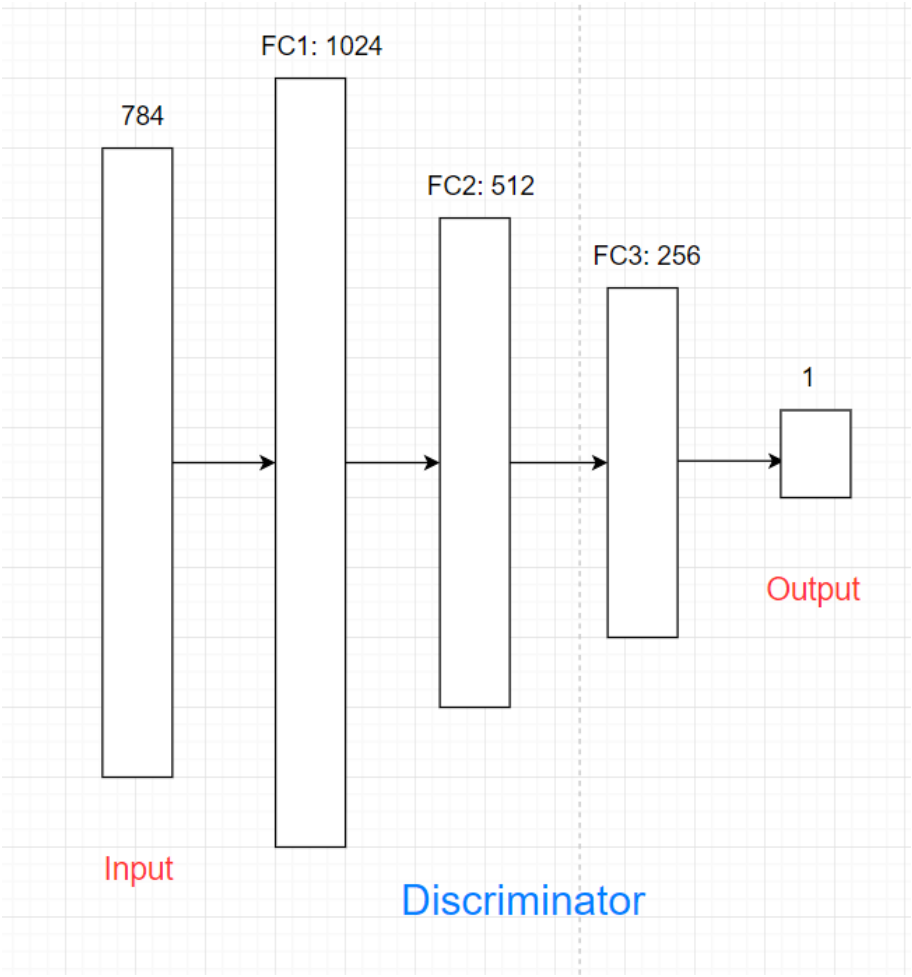


Output layer có số chiều là 784, vì output đầu ra là ảnh giống với dữ liệu MNIST, ảnh xám kích thước 28*28 (784 pixel).

Output là vector kích thước 784*1 sẽ được reshape về 28*28 đúng định dạng của dữ liệu MNIST.

- **Discriminator**

  Discriminator để phân biệt xem dữ liệu là thật (dữ liệu từ dataset) hay giả (dữ liệu sinh ra từ Generator). Trong bài toán này thì discriminator dùng để phân biệt chữ số từ bộ MNIST và dữ liệu sinh ra từ Generator. Discriminator có input là ảnh biểu diễn bằng 784 chiều, output là ảnh thật hay ảnh giả.



Input của Discriminator là ảnh kích thước 784 chiều.

Sau đấy mô hình neural network được áp dụng với số node trong hidden layer lần lượt là 1024, 512, 256. Mô hình đối xứng lại với Generator.

Output là 1 node thể hiện xác suất ảnh input là ảnh thật, hàm sigmoid được sử dụng.

b. Encoder

ResNet (Residual Network) giải quyết vấn đề mô hình học sâu càng nhiều lớp thì càng khó huấn luyện, bằng cách đưa vào "kết nối tắt" (skip connections) hoặc residual connections.

Thay vì học một ánh xạ trực tiếp H(x), ResNet học phần hiệu F(x) = H(x) - x, nên đầu ra sẽ là:

$$H(x) = F(x) + x$$

*** Cấu trúc ResNet-50**

- 50 lớp sâu, chia thành:
  - 1 lớp conv đầu (7×7)
  - 4 khối chính (residual blocks), mỗi khối gồm nhiều "bottleneck blocks":
    - Block1: 3 bottlenecks
    - Block2: 4 bottlenecks
    - Block3: 6 bottlenecks
    - Block4: 3 bottlenecks
- Tổng số lớp trọng số: 49 conv + 1 fully-connected = 50 lớp

Mỗi **bottleneck block** có 3 lớp:

- Conv 1×1 (giảm chiều)
- Conv 3×3 (xử lý chính)
- Conv 1×1 (tăng chiều)
  → Cộng thêm kết nối tắt: output = x + F(x)

c. Boundaries

Trong không gian này, các chiều không gian không chỉ là dữ liệu vô nghĩa – chúng có thể liên quan đến các thuộc tính cụ thể như:

- Tuổi (age)
- Giới tính (gender)
- Đeo kính / không (eyeglasses
- Biểu cảm khuôn mặt (smile, angry,...)
- Kiểu tóc, góc mặt, ánh sáng, v.v.

Boundary là một mặt phẳng hoặc hyperplane trong không gian latent, chia tách không gian thành 2 phần:

- Một bên là ảnh có thuộc tính A (ví dụ: có kính)
- Bên còn lại là ảnh không có A (ví dụ: không kính)

Khi bạn có latent vector w (đại diện cho 1 khuôn mặt), và 1 boundary vector b (ví dụ ranh giới giữa "già" và "trẻ"), bạn có thể điều chỉnh thuộc tính ảnh bằng cách dịch latent vector dọc theo boundary đó:

$$w' = w + \alpha \cdot b$$

- alpha > 0: tăng độ "già"
- alpha < 0: tăng độ "trẻ"

Sau đó đưa w' vào mô hình sinh ảnh → kết quả là khuôn mặt mới đã thay đổi thuộc tính.

2.3. Luồng hoạt động tổng quát

- Người dùng tương tác với giao diện PyQt5 để nhập dữ liệu và cấu hình các tham số.

- Dữ liệu được chuyển đến các thành phần trung gian để xử lý tuần tự hoặc song song:
  - Điều chỉnh kênh và độ phân giải (Channel/Resolution).
  - Sinh dữ liệu hoặc mô hình (Generator).
  - Mã hóa hoặc biểu diễn dữ liệu (Laten/Code, Encoder).
  - Trích xuất thuộc tính (Attributes).

- Các tác vụ tính toán nặng được thực hiện thông qua PyTorch và tối ưu hóa bằng CUDA để chạy trên GPU.

- Kết quả được trả về giao diện người dùng để hiển thị hoặc lưu trữ.

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

3.1.Ngôn ngữ lập trình Python

Hệ thống sử dụng ngôn ngữ lập trình Python

3.2. PyTorch

PyTorch là thư viện mã nguồn mở dùng cho Machine Learning và Deep Learning, được phát triển bởi Facebook AI Research (FAIR). Trong Anycost GAN, PyTorch được sử dụng để xây dựng và huấn luyện mô hình GAN, đồng thời cung cấp các API giúp quản lý tensor, huấn luyện mô hình trên GPU và tối ưu quá trình cập nhật tham số một cách hiệu quả. Nhờ PyTorch, hệ thống có thể tính toán linh hoạt và tăng tốc huấn luyện nhờ CUDA.

3.3. CUDA/cuDNN

CUDA (Compute Unified Device Architecture) là nền tảng tính toán song song của NVIDIA, cho phép xử lý nhanh chóng trên GPU. CUDA giúp tăng tốc huấn luyện bằng cách chạy nhiều lớp neuron song song trên GPU, giúp quá trình sinh ảnh trực quan nhanh hơn. CUDA cũng được sử dụng để giảm tải CPU và xử lý tensor một cách hiệu quả.

3.4. Horovod

Horovod là một thư viện giúp tối ưu hóa huấn luyện mô hình deep learning trên nhiều GPU, giúp giảm thời gian huấn luyện bằng cách phân tán dữ liệu. Horovod giúp hỗ trợ huấn luyện phân tán bằng cách chạy các script với nhiều GPU và giảm thời gian tính toán bằng cách tối ưu hóa giao tiếp giữa các GPU.

3.5. NumPy

NumPy là một thư viện hỗ trợ tính toán số học hiệu suất cao với mảng đa chiều. Numpy dùng để xử lý dữ liệu đầu vào trước khi đưa vào mô hình và hỗ trợ các phép toán ma trận và tensor trong huấn luyện mạng nơ-ron.

3.6. Torchvision

Torchvision là một thư viện đi kèm với PyTorch, cung cấp các công cụ và dataset hỗ trợ xử lý ảnh. Torchvision dùng để tải và tiền xử lý các tập dữ liệu như FFHQ, CelebA-HQ, LSUN Car và cung cấp các phương pháp biến đổi ảnh cần thiết cho huấn luyện mô hình.

CHƯƠNG 4. TRIỂN KHAI MÔ HÌNH

4.1. Triển Khai Mô Hình GAN

4.1.1. Bộ Sinh Ảnh (Generator)

- Bộ sinh ảnh (Generator) đóng vai trò trung tâm trong mô hình GAN, có nhiệm vụ tạo ra hình ảnh mới từ vector ngẫu nhiên (latent vector). Trong Anycost GAN, Generator có khả năng tùy biến cao, hỗ trợ nhiều cấu hình để đáp ứng nhu cầu sinh ảnh với độ phân giải và chi tiết khác nhau
- Hỗ trợ nhiều cấu hình khác nhau:
    - anycost-ffhq-config-f: Sinh ảnh khuôn mặt người dựa trên tập dữ liệu FFHQ.
    - anycost-ffhq-config-f-flexible: Phiên bản linh hoạt hơn, cho phép thay đổi kiến trúc trong lúc huấn luyện hoặc sinh ảnh
    - stylegan2-ffhq-config-f: Cấu hình gốc từ StyleGAN2, dùng làm tham chiếu
    - anycost-car-config-f: Dùng để sinh ảnh ô tô từ tập dữ liệu LSUN Car
- Các trọng số (pre-trained weights) được cung cấp dưới dạng URL, sau đó được tải về và load trực tiếp vào mô hình thông qua PyTorch.

❖ **Các bước hoạt động:**

1. Chuẩn bị style code: Sử dụng MLP nhằm chuyển đổi latent vector (z) sang phong cách (w).

```
# 1. get the style code (i.e., w+)
assert len(styles.shape) == 3  # n, n_lat, lat_dim
if not input_is_style:  # map from z to w
    styles = self.get_style(styles)

if truncation < 1:
    styles = (1 - truncation) * truncation_style.view(1, 1, -1) + truncation * styles

if styles.shape[1] == 1:  # only one style provided
    styles = styles.repeat(1, self.n_style, 1)
elif styles.shape[1] == 2:  # two styles to mix
    if inject_index is None:
        inject_index = random.randint(1, self.n_style - 1)
    style1 = styles[:, 0:1].repeat(1, inject_index, 1)
    style2 = styles[:, 1:2].repeat(1, self.n_style - inject_index, 1)
    styles = torch.cat([style1, style2], 1)
else:  # full style
    assert styles.shape[1] == self.n_style
```

2. Sinh noise ngẫu nhiên: Sinh noise để bổ sung các chi tiết tinh tế vào hình ảnh.

```
# 2. get noise
if noise is None:
    if randomize_noise:
        noise = [None] * self.num_layers
    else:
        noise = [getattr(self.noises, f'noise_{i}') for i in range(self.num_layers)]
```

3. Khởi tạo constant input: Ban đầu, Generator dùng tập constant input (thay vì random noise như các GAN truyền thống).

4. Xây dựng các tầng conv: Gồm các lớp StyledConv, ToRGB, và Upsample để tạo hình ảnh đạt độ phân giải mong muốn.

5. Sinh hình ảnh: Qua dàn layer conv, cuối cùng sinh ra hình ảnh RGB.

```
# 3. generate images
all_rgbs = []

out = self.input(styles.shape[0])  # get constant input
out = self.conv1(out, styles[:, 0], noise=noise[0])
skip = self.to_rgb1(out, styles[:, 1])
all_rgbs.append(skip)

if hasattr(self, 'target_res') and target_res is None:  # a quick fix for search
    target_res = self.target_res

i = 1
for conv1, conv2, noise1, noise2, to_rgb in zip(
        self.convs[::2], self.convs[1::2], noise[1::2], noise[2::2], self.to_rgbs
):
    out = conv1(out, styles[:, i], noise=noise1)
    out = conv2(out, styles[:, i + 1], noise=noise2)
    skip = to_rgb(out, styles[:, i + 2], skip)
    all_rgbs.append(skip)
```

4.1.2. Bộ phân biệt (Discriminator)

Discriminator nhận ảnh đầu vào (có thể là ảnh thật hoặc ảnh sinh ra bởi Generator), và đưa ra dự đoán xem ảnh đó là "thật" hay "giả".

❖ **Các tham số chính:**

- resolution: Kích thước hình ảnh nhận vào.
- channel_multiplier: Hằng số nhân kênh.
- blur_kernel: Nhân lọc.

- act_func: Hàm kích hoạt.
- n_res: Số tầng resblock.
- modulate: Tùy chọn bật/tắt g_arch modulation.

❖ **Các bước hoạt động:**

1. Nhận ảnh đầu vào: Discriminator nhận hình ảnh và tầng convolution đầu tiên xử lý tín hiệu.

2. Xây dựng các ResBlock: Gồm nhiều tầng ResBlock để giảm kích thước hình ảnh và tăng chi tiết quan trọng.

```
convs = [ConvLayer(3, channels[resolution], 1, activate=act_func)]

log_res = int(math.log(resolution, 2))

in_channel = channels[resolution]

for i in range(log_res, 2, -1):
    out_channel = channels[2 ** (i - 1)]  # the out channel corresponds to a lower resolut
    convs.append(ResBlock(in_channel, out_channel, blur_kernel, act_func=act_func))
    in_channel = out_channel
```

3. Thêm minibatch discrimination: Kỹ thuật giúp phân biệt được tạo ra bởi Generator hay hình thật.

```
    stddev = out.view(
        group, -1, self.stddev_feat, channel // self.stddev_feat, height, width
    )
    stddev = torch.sqrt(stddev.var(0, unbiased=False) + 1e-8)
    stddev = stddev.mean([2, 3, 4], keepdims=True).squeeze(2)
    stddev = stddev.repeat(group, 1, height, width)
    out = torch.cat([out, stddev], 1)

    out = self.final_conv(out)

    out = out.view(batch, -1)
    out = self.final_linear(out)

    return out
```

4. Fully Connected Layer: Cuối cùng, lớp fully connected sắp xếp dữ liệu và phân loại.

```
        self.final_conv = ConvLayer(in_channel + 1, channels[4], 3, activate=act_func)
        self.final_linear = nn.Sequential(
            EqualLinear(channels[4] * 4 * 4, channels[4], activation=act_func),
            EqualLinear(channels[4], 1),
        )
```

4.1.3. Bộ Mã Hóa (Encoder)

- Dựa trên kiến trúc ResNet50.
- Được thiết kế để trích xuất mã đặc trưng từ ảnh đầu vào.
- Huấn luyện với VGG LPIPS loss để đảm bảo chất lượng mã hóa phù hợp với StyleGAN2.

❖ **Cách Thức Hoạt Động**

- Mã Hóa Hình Ảnh:
  - Hình ảnh đầu vào được mã hóa thành các đặc trưng trung gian thông qua các lớp convolution và residual blocks của ResNet50.
  - Các đặc trưng này sau đó được ánh xạ thành latent code thông qua lớp fully connected.

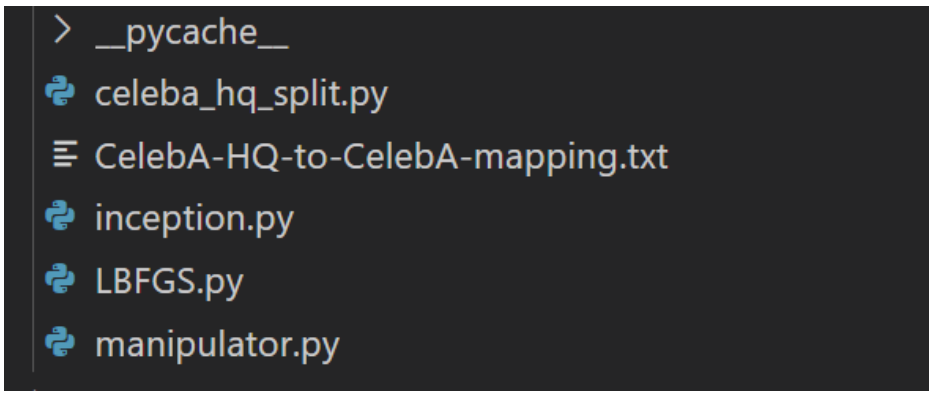
```
class ResNet50Encoder(nn.Module):
    def __init__(self, n_style, style_dim=512, mean_latent=None, pretrained=False):
        super().__init__()
        self.n_style = n_style
        self.style_dim = style_dim
        # copy all the modules from an existing resnet
        model_tmp = resnet50(pretrained=pretrained)
        self.conv1 = model_tmp.conv1
        self.bn1 = model_tmp.bn1
        self.relu = model_tmp.relu
        self.maxpool = model_tmp.maxpool

        self.layer1 = model_tmp.layer1
        self.layer2 = model_tmp.layer2
        self.layer3 = model_tmp.layer3
        self.layer4 = model_tmp.layer4

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(model_tmp.fc.in_features, n_style * style_dim)

        # register a mean buffer
        self.register_buffer('mean_latent', torch.rand(style_dim))
```

- Tạo Style Code: Latent code được điều chỉnh bằng cách cộng thêm giá trị trung bình mean latent, tạo ra các style code có thể được sử dụng trong các mô hình sinh.

```python
def forward(self, x):
    assert x.shape[-1] == x.shape[-2] == 256  # only accept input with resolution 256x256
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    x = x.view(x.shape[0], self.n_style, self.style_dim) + self.mean_latent.view(1, 1, -1

    return x  # shape: n, n_style, style_dim
```

4.1.4. Xử lý dữ liệu CelebA-HQ và trích xuất đặc trưng từ ảnh

Xử lý dữ liệu CelebA-HQ, trích xuất đặc trưng từ ảnh, huấn luyện mô hình phân loại latent space, và tối ưu hóa.



a. celeba_hq_split.py

- Dùng để chia tập dữ liệu CelebA-HQ thành 3 tập: train / validation / test dựa trên thông tin mapping từ file CelebA-HQ-to-CelebA-mapping.txt.
- Cụ thể, nó đọc file mapping, rồi dựa trên ID để xác định ảnh nào thuộc tập nào.

```python
def get_celeba_hq_split():
    train_idx = []
    test_idx = []
    val_idx = []

    with open('thirdparty/CelebA-HQ-to-CelebA-mapping.txt') as f:
        lines = f.readlines()
    celeba_ids = [int(l.strip().split()[1]) for l in lines]

    for idx, x in enumerate(celeba_ids):  # celeba-hq idx, celeba idx
        if 162771 <= x < 182638:
            val_idx.append(idx)
        elif x >= 182638:
            test_idx.append(idx)
        else:
            train_idx.append(idx)

    # NOTICE: the range of the index is 0-29999
    # but the range of the celebahq images fname is 1-30000
    return train_idx, val_idx, test_idx
```

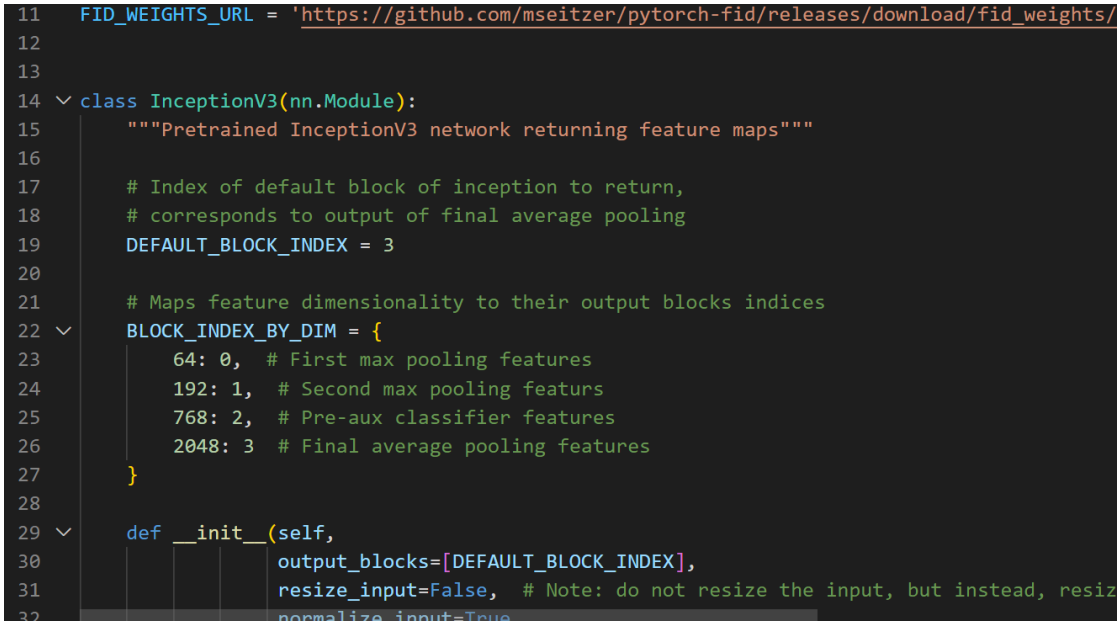
b. CelebA-HQ-to-CelebA-mapping.txt

- File mapping giúp liên kết chỉ số ảnh trong CelebA-HQ với ảnh tương ứng trong CelebA gốc.
- Mỗi dòng chứa: chỉ số CelebA-HQ ID CelebA gốc tên file ảnh .jpg

```
 1   0          119613      119614.jpg
 2   1          99094       099095.jpg
 3   2          200121      200122.jpg
 4   3          81059       081060.jpg
 5   4          202040      202041.jpg
 6   5          614         000615.jpg
 7   6          50915       050916.jpg
 8   7          166545      166546.jpg
 9   8          143861      143862.jpg
10   9          101741      101742.jpg
11   10         157745      157746.jpg
12   11         78315       078316.jpg
13   12         167419      167420.jpg
14   13         167706      167707.jpg
15   14         55525       055526.jpg
16   15         89615       089616.jpg
17   16         34958       034959.jpg
18   17         174481      174482.jpg
19   18         154921      154922.jpg
20   19         163131      163132.jpg
21   20         57379       057380.jpg
22   21         97643       097644.jpg
23   22         82111       082112.jpg
```

c. inception.py

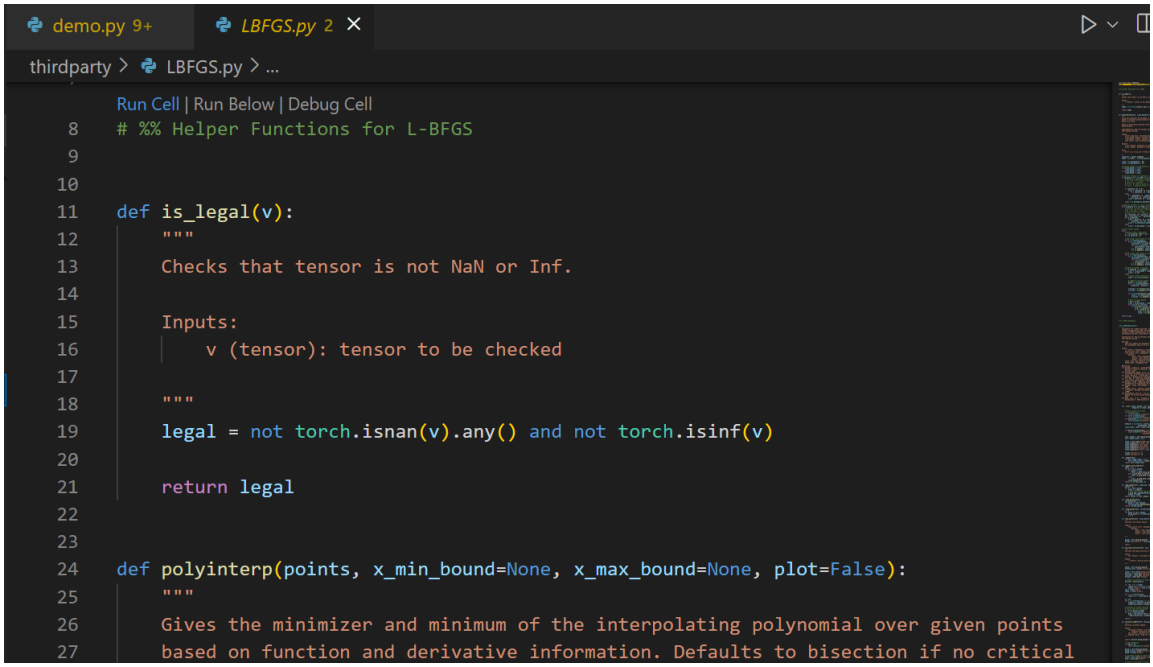
- Định nghĩa mô hình InceptionV3 (đặc biệt là phiên bản dùng để tính FID (Fréchet Inception Distance)).
- Dùng để trích xuất đặc trưng (feature) của ảnh từ mô hình Inception cho việc đánh giá chất lượng ảnh sinh/gán.



```
11  FID_WEIGHTS_URL = 'https://github.com/mseitzer/pytorch-fid/releases/download/fid_weights/
12
13
14  class InceptionV3(nn.Module):
15      """Pretrained InceptionV3 network returning feature maps"""
16
17      # Index of default block of inception to return,
18      # corresponds to output of final average pooling
19      DEFAULT_BLOCK_INDEX = 3
20
21      # Maps feature dimensionality to their output blocks indices
22      BLOCK_INDEX_BY_DIM = {
23          64: 0,  # First max pooling features
24          192: 1,  # Second max pooling featurs
25          768: 2,  # Pre-aux classifier features
26          2048: 3  # Final average pooling features
27      }
28
29      def __init__(self,
30                   output_blocks=[DEFAULT_BLOCK_INDEX],
31                   resize_input=False,  # Note: do not resize the input, but instead, resiz
32                   normalize_input=True
```

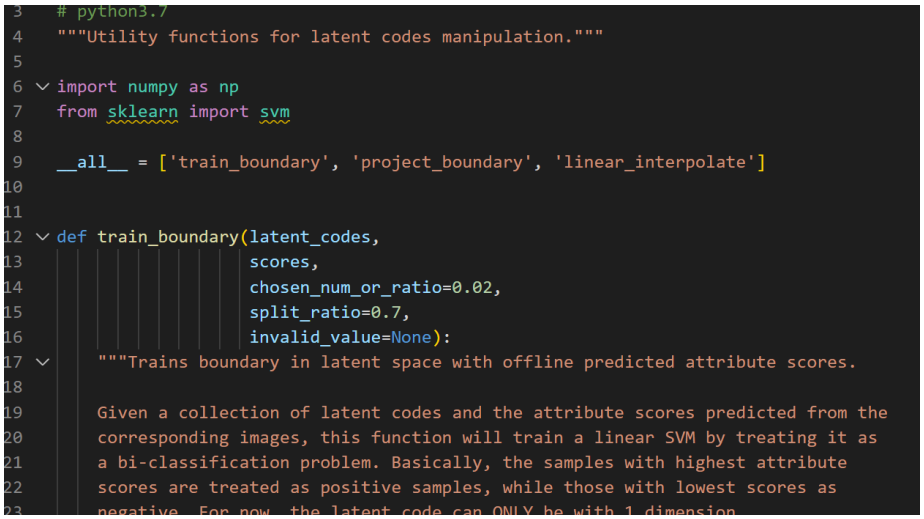
d. LBFGS.py

- Cài đặt thuật toán tối ưu L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) dạng custom cho PyTorch.
- Hỗ trợ bài toán tối ưu hóa trong huấn luyện mạng, đặc biệt là với yêu cầu tối ưu mượt mà hơn SGD.



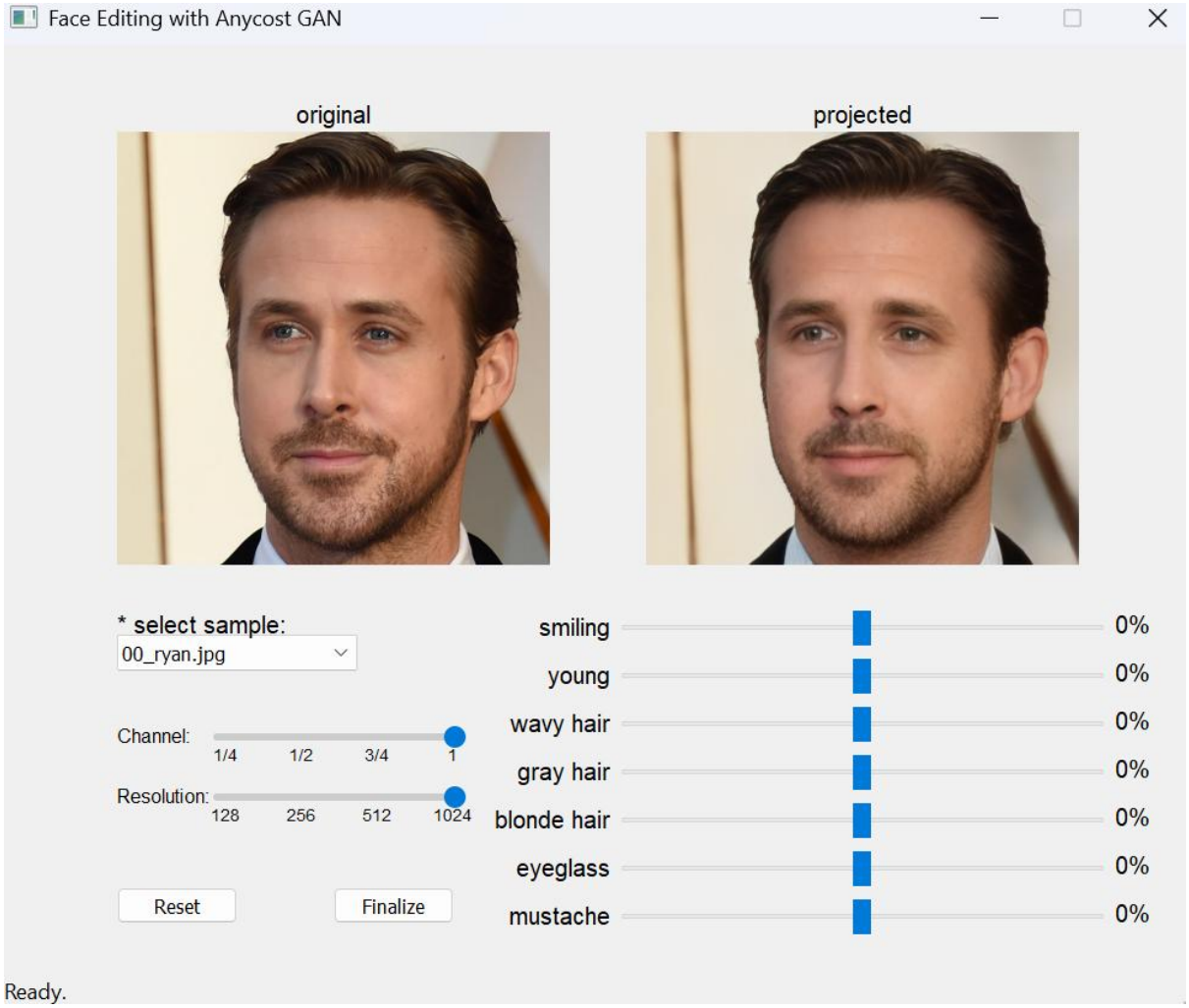
e. manipulator.py

- Cung cấp các hàm manipulation trong latent space của mô hình sinh ảnh.
- Bao gồm
  - train_boundary: Huấn luyện SVM để tìm đường phân cách (boundary) trong latent space (ví dụ: gương mặt có đeo kính hay không).
  - project_boundary: Điều chỉnh boundary với các điều kiện bổ sung.
  - linear_interpolate: Nội suy tuyến tính giữa các latent vector.



4.2. Triển Khai Ứng Dụng Chỉnh Sửa Khuôn Mặt Sử Dụng Anycost GAN

DEMO



1. Ảnh khuôn mặt

- Bên trái (original): Ảnh gốc được tải lên
- Bên phải (projected): Ảnh đã được "chiếu" (projected) vào không gian latent của StyleGAN. Đây là kết quả của việc encoder chuyển đổi ảnh gốc thành latent code và generator tạo lại ảnh từ latent code đó.

2. Thanh chọn mẫu

- "select sample": Dropdown menu để chọn ảnh mẫu có sẵn

- "Tải ảnh": Nút đã được thêm vào cho phép người dùng tải ảnh riêng của họ lên thay vì sử dụng ảnh mẫu.

3. Các thanh điều chỉnh

Các thuộc tính khuôn mặt

- smiling: Điều chỉnh độ cười
- young: Điều chỉnh độ trẻ/già
- wavy hair: Điều chỉnh độ gợn sóng của tóc
- gray hair: Điều chỉnh độ bạc của tóc
- blonde hair: Điều chỉnh độ vàng của tóc
- eyeglass: Điều chỉnh kính mắt
- mustache: Điều chỉnh râu mép

Các thanh này đều đang ở vị trí gần về phía bên phải, cho thấy các thuộc tính đang được điều chỉnh ở mức độ dương (tăng) nhẹ.

Tham số hiệu suất

- channel: Điều chỉnh tỷ lệ kênh từ 1/4 đến 1
- resolution: Độ phân giải từ 128px đến 1024px

4. Các nút chức năng

- Reset: Đặt lại tất cả các thanh trượt về vị trí ban đầu
- Finalize: Tạo hình ảnh cuối cùng với chất lượng tốt nhất

  Điều chỉnh kênh sẽ giúp tốc độ tính toán nhanh hơn và đỡ tốn tài nguyên hơn, có thể phù hợp với một số máy cấu hình yếu khi điều chỉnh kênh thấp

  Thay đổi độ phân giải sẽ quyết định về độ nét của ảnh

  Độ phân giải cao:

  - Yêu cầu nhiều tài nguyên GPU/CPU
  - Xử lý chậm hơn
  - Cần nhiều bộ nhớ hơn

  Độ phân giải thấp:

  - Yêu cầu ít tài nguyên
  - Xử lý nhanh hơn
  - Sử dụng ít bộ nhớ hơn

TÀI LIỆU THAM KHẢO

(Ji Lin, Anycost GANs for Interactive Image Synthesis and Editing, 2021)

(Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, Alexei A. Efros, Generative Visual Manipulation on the Natural Image Manifold, 2016)

(Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila, Analyzing and Improving the Image Quality of StyleGAN, *2019*)