

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỖ TRUNG PHONG

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC
NGÀNH KHOA HỌC MÁY TÍNH

TÌM HIỂU VỀ MÔ HÌNH HỌC SÂU GAN
(GENERATIVE ADVERSARIAL NETWORKS) VÀ
ỨNG DỤNG TRONG TẠO ẢNH NGHỆ THUẬT

CBHD: ThS. Nguyễn Thanh Hải

Sinh viên: Đỗ Trung Phong

Mã số sinh viên: 2020604284

KHOA HỌC MÁY TÍNH

Hà Nội – 2024

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC
NGÀNH KHOA HỌC MÁY TÍNH

TÌM HIỂU VỀ MÔ HÌNH HỌC SÂU GAN
(GENERATIVE ADVERSARIAL NETWORKS) VÀ
ỨNG DỤNG TRONG TẠO ẢNH NGHỆ THUẬT

CBHD:

ThS. Nguyễn Thanh Hải

Sinh viên:

Đỗ Trung Phong

Mã số sinh viên:

2020604284

Hà Nội – 2024

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT	iv
DANH MỤC HÌNH ẢNH	v
DANH MỤC BẢNG BIỂU	viii
LỜI CẢM ƠN	1
LỜI NÓI ĐẦU	2
CHƯƠNG 1. GIỚI THIỆU CHUNG VÀ PHÁT BIỂU BÀI TOÁN	4
1.1. Giới thiệu chung về đề tài	4
1.1.1. Lý do lựa chọn đề tài.....	4
1.1.2. Mục tiêu nghiên cứu.....	6
1.1.3. Giới hạn và phạm vi nghiên cứu	7
1.2. Bài toán.....	8
1.2.1. Phát biểu bài toán	8
1.2.2. Đầu vào và đầu ra.....	10
1.2.3. Ràng buộc liên quan.....	11
1.3. Một số cơ sở lý thuyết	11
1.3.1. Mạng neural tích chập (CNN).....	11
1.3.2. Lan truyền ngược (Backpropagation)	17
CHƯƠNG 2. PHƯƠNG PHÁP ỨNG DỤNG	19
2.1. Giới thiệu về GAN	19
2.2. Cấu trúc mạng GAN.....	20
2.2.1. Discriminator.....	20
2.2.2. Generator.....	22
2.2.3. Loss function	24
2.3. Một số mô hình biến thể của GAN	27

2.3.1.	GAN cơ bản	27
2.3.2.	DCGAN.....	28
2.3.3.	CGAN	29
2.4.	Ứng dụng vào tạo ảnh nghệ thuật.....	30
2.4.1.	Áp dụng DCGAN vào tạo ảnh nghệ thuật	30
2.4.2.	Áp dụng CGAN vào tạo ảnh nghệ thuật	33
CHƯƠNG 3.	TIẾN HÀNH THỰC NGHIỆM.....	37
3.1.	Dataset	37
3.1.1.	WikiArt	37
3.1.2.	Portrait paintings	38
3.2.	Chương trình ứng dụng	39
3.2.1.	Huấn luyện mô hình.....	39
3.2.2.	Đánh giá	49
CHƯƠNG 4.	TÍCH HỢP HỆ THỐNG	60
4.1.	Giới thiệu về framework sử dụng.....	60
4.2.	Phân tích thiết kế hệ thống	62
4.2.1.	Biểu đồ use case	62
4.2.2.	Mô tả chi tiết các use case.....	63
4.2.3.	Phân tích các use case	70
4.2.4.	Thiết kế cơ sở dữ liệu.....	78
4.2.5.	Thiết kế giao diện hệ thống.....	81
4.3.	Các chức năng của hệ thống.....	84
4.3.1.	Chức năng chính	84
4.3.2.	Một số chức năng khác	86
KẾT LUẬN		87

TÀI LIỆU THAM KHẢO.....	88
-------------------------	----

DANH MỤC TỪ VIẾT TẮT

GAN	Mạng đối nghịch tạo sinh (Generative Adversarial Network)
DCGAN	Mạng đối nghịch tạo sinh tích chập đa lớp (Deep Convolutional GAN)
CGAN	Mạng đối nghịch tạo sinh có điều kiện (Conditional GAN)
CNN	Mạng tích chập (convolutional neural network)
CGAN pix2pix	Image-to-Image Translation with Conditional Adversarial Networks
NN	Mạng nơ-ron nhân tạo (Neural Network)
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
GPU	Đơn vị xử lý đồ họa (Graphics Processing Unit)

DANH MỤC HÌNH ẢNH

Hình 1.1	Bức tranh do AI tạo ra được bán với giá cao	5
Hình 1.2	Mô hình CNN [6]	12
Hình 1.3	Mô tả lớp tích chập – ma trận 5×5 và bộ lọc 3×3 [6].....	13
Hình 1.4	Mô tả lớp tích chập – feature map [6].....	13
Hình 1.5	Ví dụ các bộ lọc [6].....	14
Hình 1.6	Mô tả lớp tích chập hoạt động với stride là 2 [6].....	14
Hình 1.7	Mô tả hàm phi tuyến [6].....	15
Hình 1.8	Ví dụ về lớp gộp [6]	16
Hình 1.9	Ví dụ về lớp kết nối đầy đủ [6]	16
Hình 2.1	Kiến trúc của mô hình GAN [7].....	20
Hình 2.2	Kiến trúc mô hình GAN – Discriminator [7].....	21
Hình 2.3	Kiến trúc mô hình GAN – Generator [7]	23
Hình 2.4	Mô tả kiến trúc của Generator trong DCGAN [9]	31
Hình 2.5	Mô tả kiến trúc của Discriminator trong DCGAN [9]	32
Hình 2.6	Kiến trúc encoder-decoder của Generator trong CGAN [7].....	35
Hình 2.7	Mô tả kiến trúc của Discriminator trong CGAN [7].....	35
Hình 3.1	Bộ dữ liệu ảnh WikiArt.....	38
Hình 3.2	Bộ dữ liệu ảnh Portrait Paintings	39
Hình 3.3	Khai báo thư viện - DCGAN	39
Hình 3.4	Thiết lập tham số và định nghĩa hàm loss – DCGAN	40
Hình 3.5	Đọc và tiền xử lý ảnh – DCGAN	40
Hình 3.6	Xây dựng mạng Discriminator – DCGAN	41
Hình 3.7	Xây dựng mạng generator – DCGAN.....	41
Hình 3.8	Hàm tính toán loss – DCGAN	42
Hình 3.9	Xây dựng hàm huấn luyện trên mỗi batch – DCGAN.....	42
Hình 3.10	Huấn luyện mô hình và lưu mô hình – DCGAN	43
Hình 3.11	Hàm sinh và hiển thị biến thể ảnh mới từ bộ ảnh từ model.....	43
Hình 3.12	Biến thể ảnh mới 1 được sinh ra từ dcgan_generator	44

Hình 3.13 Biến thể ảnh mới 2 được sinh ra từ dcgan_generator	44
Hình 3.14 Xây dựng hàm upsample và downsample – CGAN	45
Hình 3.15 Xây dựng mạng Discriminator – CGAN	45
Hình 3.16 Xây dựng mạng generator	46
Hình 3.17 Hàm tính toán loss – CGAN	46
Hình 3.18 Xây dựng hàm huấn luyện trên mỗi batch	47
Hình 3.19 Hàm huấn luyện mô hình – CGAN.....	47
Hình 3.20 Hàm tạo ảnh chi tiết từ ảnh phác họa tương ứng	48
Hình 3.21 Ảnh phác họa và chi tiết được tạo 1	48
Hình 3.22 Ảnh phác họa và ảnh chi tiết được tạo 2	49
Hình 3.23 Ảnh phác họa và ảnh chi tiết được tạo 3	49
Hình 3.24 Khai báo thư viện – Đánh giá mô hình DCGAN.....	52
Hình 3.25 Đọc và tiền xử lý ảnh – Đánh giá mô hình DCGAN	53
Hình 3.26 Tạo dữ liệu ảnh giả bằng generator – Đánh giá mô hình DCGAN	53
Hình 3.27 Hàm thay đổi kích thước ảnh – Đánh giá mô hình DCGAN.....	54
Hình 3.28 Hàm tính Inception Score – Đánh giá mô hình DCGAN	54
Hình 3.29 Hàm tính Fréchet Inception Distance) – Đánh giá DCGAN	55
Hình 3.30 Hàm đánh giá mô hình DCGAN.....	55
Hình 3.31 Kết quả đánh giá mô hình DCGAN	55
Hình 3.32 Hàm đọc ảnh và tiền xử lý ảnh – CGAN	56
Hình 3.33 Hàm sinh ảnh giả bằng Generator – CGAN	57
Hình 3.34 Hàm tính Fréchet Inception Distance – CGAN	57
Hình 3.35 Hàm tiền xử lý – CGAN	58
Hình 3.36 Hàm đánh giá mô hình CGAN.....	58
Hình 3.37 Kết quả đánh giá mô hình	58
Hình 4.1 Biểu đồ use case tổng quát	62
Hình 4.2 Biểu đồ mối quan hệ giữa các use case.....	63
Hình 4.3 Biểu đồ trình tự use case Đăng nhập.....	70
Hình 4.4 Biểu đồ lớp phân tích use case Đăng nhập	71

Hình 4.5 Biểu đồ trình tự use case Đăng ký	71
Hình 4.6 Biểu đồ lớp phân tích use case Đăng ký	72
Hình 4.7 Biểu đồ trình tự use case Tìm kiếm	72
Hình 4.8 Biểu đồ lớp phân tích use case Tìm kiếm	73
Hình 4.9 Biểu đồ trình tự use case Xem ảnh	73
Hình 4.10 Biểu đồ lớp phân tích use case Xem ảnh	74
Hình 4.11 Biểu đồ trình tự use case Thêm ảnh vào bộ sưu tập	75
Hình 4.12 Biểu đồ lớp phân tích use case Thêm ảnh vào bộ sưu tập	76
Hình 4.13 Biểu đồ trình tự use case Sinh biến thẻ ảnh mới.....	76
Hình 4.14 Biểu đồ lớp phân tích use case Sinh biến thẻ ảnh mới	77
Hình 4.15 Biểu đồ trình tự use case Tạo ảnh chi tiết từ sketch	77
Hình 4.16 Biểu đồ lớp phân tích use case Tạo ảnh chi tiết từ sketch	78
Hình 4.17 Mô hình cơ sở dữ liệu của hệ thống.....	78
Hình 4.18 Giao diện trang chủ	82
Hình 4.19 Giao diện trang sinh biến thẻ ảnh mới	82
Hình 4.20 Giao diện tạo ảnh chi tiết từ ảnh phác thảo.....	83
Hình 4.21 Giao diện trang đăng nhập	83
Hình 4.22 Giao diện trang đăng ký	84
Hình 4.23 Giao diện trang kết quả tìm kiếm.....	84
Hình 4.24 Test chức năng Sinh biến thẻ ảnh mới từ bộ dữ liệu	85
Hình 4.25 Kết quả test chức năng Sinh biến thẻ ảnh mới từ bộ dữ liệu	85
Hình 4.26 Test chức năng tạo ảnh chi tiết từ ảnh phác thảo.....	86
Hình 4.27 Kết quả test chức năng tạo ảnh chi tiết từ ảnh phác họa.....	86

DANH MỤC BẢNG BIỂU

Bảng 4.1 Chi tiết bảng User	79
Bảng 4.2 Chi tiết bảng Collections	79
Bảng 4.3 Chi tiết bảng Images	80
Bảng 4.4 Chi tiết bảng Tags	80
Bảng 4.5 Chi tiết bảng Image_tags	81

LỜI CẢM ƠN

Để có được một đồ án tốt nghiệp chỉnh chu và đạt được kết quả tốt đẹp, em đã nhận được sự giúp đỡ, hỗ trợ của các thầy cô bạn bè. Với tình cảm sâu sắc, chân thành của mình, cho phép em được bày tỏ lòng biết ơn sâu sắc đến tất cả các thầy cô và bạn bè đã nhiệt tình giúp đỡ, góp ý.

Đặc biệt em xin gửi lời cảm ơn chân thành nhất tới thầy giáo - Th.S Nguyễn Thanh Hải đã quan tâm giúp đỡ, trực tiếp hướng dẫn em hoàn thành tốt đồ án này trong thời gian qua.

Sinh viên thực hiện:

Đỗ Trung Phong

LỜI NÓI ĐẦU

Trước sự phát triển của khoa học và công nghệ đang diễn ra với tốc độ đáng kinh ngạc, trí tuệ nhân tạo (AI) đang trở thành trọng tâm của nhiều lĩnh vực, từ công nghệ thông tin, thương mại điện tử đến ngành nghệ thuật và thiết kế. Trong thập kỷ gần đây, sự tiến bộ trong lĩnh vực học sâu đã mở ra những cánh cửa mới đầy hứa hẹn cho AI, và mô hình học sâu Generative Adversarial Networks (Mạng đối nghịch tạo sinh) đã nổi lên như một trong những công cụ quan trọng nhất trong danh mục của nó.

Đồ án tốt nghiệp này tập chung vào việc tìm hiểu về mô hình học sâu GAN (Generative Adversarial Networks) và ứng dụng của nó trong việc tạo ảnh nghệ thuật. GAN là một mô hình quan trọng trong lĩnh vực học sâu, nổi bật với cách tiếp cận độc đáo là đào tạo 2 mạng nơron cạnh tranh với nhau được gọi là Generator dùng để sinh ra các ảnh giả và Discriminator dùng để phân biệt giữa ảnh thật và ảnh giả. Đồ án này sẽ đào sâu về cơ chế hoạt động của GAN, các phương pháp và kỹ thuật được sử dụng, cũng như áp dụng chúng vào việc tạo ra các hình ảnh nghệ thuật đa dạng và sáng tạo.

Đồ án này sẽ được chia thành các chương chính như sau:

Chương 1. Giới thiệu chung và phát biểu bài toán: Chương này sẽ đề cập đến bối cảnh và ngữ cảnh tổng quan của đề tài, cung cấp cái nhìn tổng quan về mô hình GAN và vấn đề cụ thể mà đồ án sẽ giải quyết.

Chương 2. Các phương pháp ứng dụng trong GAN: Chương 2 sẽ đi sâu vào các phương pháp và những kỹ thuật cơ bản cũng như tiên tiến được sử dụng trong mô hình GAN, giới thiệu và phân tích các thành phần chính của GAN, các biến thể phổ biến và cách chúng được áp dụng vào tạo ảnh nghệ thuật.

Chương 3. Tiến hành thực nghiệm: Phần này sẽ trình bày về các kết quả thực nghiệm và thử nghiệm của đồ án, từ việc đào tạo mô hình GAN để tạo ra biến thể ảnh mới từ bộ dữ liệu và tạo ảnh chi tiết từ ảnh phác họa, đồng thời cũng đánh giá và phân tích kết quả để hiểu rõ hơn về mô hình này.

Chương 4. Tích hợp hệ thống: Phần này sẽ trình bày cách tích hợp GAN vào 1 hệ thống tổng thể, bao gồm quy trình tích hợp, giao tiếp với các phần khác của hệ thống và các vấn đề liên quan đến tích hợp.

Kết luận và hướng phát triển: Cuối cùng, chương này sẽ tổng kết lại các kết quả đạt được từ đồ án và đề xuất các hướng phát triển tiếp theo.

Em xin chân thành cảm ơn Thầy Nguyễn Thanh Hải đã giúp em hoàn thiện bài báo cáo này. Do giới hạn về năng lực và kiến thức nên bài báo cáo có thể có một số sai sót, mong thầy và các bạn chỉ ra để giúp em có một bài báo cáo hoàn thiện hơn.

CHƯƠNG 1. GIỚI THIỆU CHUNG VÀ PHÁT BIỂU BÀI TOÁN

1.1. Giới thiệu chung về đề tài

1.1.1. Lý do lựa chọn đề tài

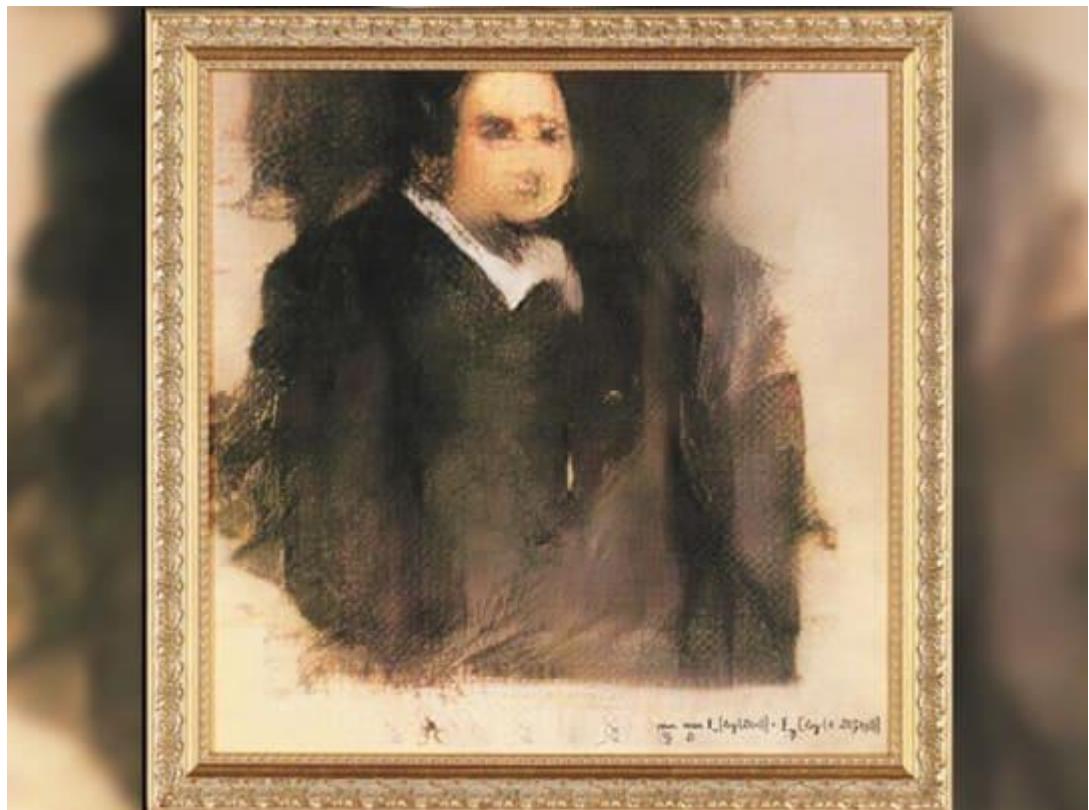
Trong thời đại số hóa ngày nay, nhu cầu về nội dung nghệ thuật độc đáo và sáng tạo ngày càng tăng lên. Mô hình học sâu GAN đã chứng minh được khả năng của mình trong việc tạo ra dữ liệu mới và độc đáo, từ ảnh đến âm nhạc và văn bản. Trong lĩnh vực nghệ thuật, GAN đã mở ra một thế giới mới của sự sáng tạo, cho phép các nghệ sĩ và nhà thiết kế khám phá và tạo ra những tác phẩm độc đáo mà trước đây có thể không thể.

Một trong những thành tựu lớn nhất của GAN là khả năng tạo ra hình ảnh người với độ chân thực gần như hoàn hảo. Từ việc tạo ra khuôn mặt người cho đến việc thay đổi độ tuổi, GAN đã phát triển những mô hình ngày càng chính xác và đa dạng hơn qua từng năm. Khả năng này đã mở ra cánh cửa cho các ứng dụng trong việc tạo ra hình ảnh phân khúc, thậm chí là tạo ra nhân vật hoạt hình.

Bên cạnh đó, GAN còn được sử dụng để tạo ra hình ảnh tự nhiên với độ phân giải cao hơn. Việc này giúp cải thiện chất lượng ảnh và khôi phục lại những chi tiết mà trước đây có thể bị mờ nhạt hoặc mất đi. Điều này có ý nghĩa lớn trong các ứng dụng y học, khoa học, và trong thực tế, khi chúng ta muốn nhìn rõ ràng hơn vào các hình ảnh từ các nguồn gốc khác nhau.

Thêm vào đó, GAN đã thành công trong việc chuyển đổi hình ảnh từ một dạng sang dạng khác một cách tự động và hiệu quả. Từ việc chuyển đổi các bức ảnh thành hình ảnh nghệ thuật, tạo ra các phiên bản khác nhau của cùng một bức ảnh, cho đến việc tạo ra hình ảnh dựa trên mô tả văn bản, GAN đã thúc đẩy sự đa dạng và sáng tạo trong việc tạo ra và tương tác với hình ảnh.

Cuối cùng, một trong những thành tựu đáng chú ý nhất của GAN là việc tạo ra các bức tranh hoàn toàn mới và độc đáo. Nhờ vào việc huấn luyện trên một lượng lớn dữ liệu chứa các bức tranh chân dung từ các thế kỷ trước, GAN có khả năng tạo ra những bức tranh mới mà không cần sự can thiệp của con người. Bức họa Edmond de Bellamy sau đây là một minh chứng rõ ràng cho điều này, khi mà bức tranh này đã thu về một khoản tiền khổng lồ trong một phiên đấu giá.



Hình 1.1 Bức tranh do AI tạo ra được bán với giá cao

Việc bức tranh này được đánh giá cao và bán với mức giá vượt xa dự kiến đã mở ra một trang mới trong lịch sử hội họa. Nó không chỉ là một thành tựu trong công nghệ và nghệ thuật số, mà còn là một tín hiệu cho thấy sự tiếp cận mới trong việc sáng tạo nghệ thuật.

Tuy nhiên, mặc dù có sự quan tâm ngày càng tăng về mô hình GAN, nhưng vẫn còn nhiều lỗ hổng kiến thức liên quan đến cách mô hình hoạt động và cách áp dụng nó trong lĩnh vực nghệ thuật. Cụ thể, việc hiểu rõ về

cách GAN có thể tạo ra ảnh nghệ thuật có chất lượng cao và độc đáo vẫn còn là một thách thức.

Sự hứng thú cá nhân của em đến đè tài này đến từ việc nhận thức về tiềm năng của GAN trong việc tạo ra các tác phẩm nghệ thuật độc đáo và ấn tượng. Sự kết hợp giữa công nghệ và nghệ thuật không chỉ tạo ra những cơ hội mới cho sự sáng tạo mà còn mở ra một thế giới mới của trải nghiệm nghệ thuật cho người xem.

Ngoài ra, đè tài này cũng có ứng dụng rất rộng rãi trong thực tế. Từ việc tạo ra các tác phẩm nghệ thuật số độc đáo đến việc áp dụng GAN trong việc tạo ra các thiết kế mới trong công nghiệp thời trang và truyền thông quảng cáo, khả năng sáng tạo của GAN không chỉ giới hạn trong lĩnh vực nghệ thuật mà còn lan rộng sang nhiều lĩnh vực khác nhau.

1.1.2. Mục tiêu nghiên cứu

Mục tiêu của nghiên cứu này là tìm hiểu sâu hơn về mô hình học sâu GAN và khám phá cách ứng dụng của nó trong việc tạo ra các tác phẩm nghệ thuật đa dạng và sáng tạo. Cụ thể, các mục tiêu nghiên cứu được xác định như sau:

- Phân tích và hiểu rõ về cơ sở lý thuyết của mô hình học sâu GAN, bao gồm cách hoạt động của các thành phần cơ bản như mạng sinh và mạng phân biệt.
- Tìm hiểu về các phương pháp huấn luyện mô hình GAN, bao gồm các kỹ thuật tối ưu hóa và các chiến lược huấn luyện hiệu quả.
- Xây dựng được chương trình huấn luyện mô hình DCGAN để tạo ra các biến thể hình ảnh mới và đa dạng với màu sắc, ánh sáng, độ sắc nét và góc chụp khác nhau từ bộ dữ liệu ban đầu.
- Xây dựng được chương trình huấn luyện mô hình CGAN được áp dụng để chuyển đổi các bản phác thảo hoặc concept design thành

các hình ảnh chất lượng cao, cung cấp cho người sử dụng một cái nhìn cụ thể và hấp dẫn hơn về sản phẩm cuối cùng.

- Xây dựng một ứng dụng tích hợp để thực hiện demo các ứng dụng của mô hình được nhắc ở trên
- Đánh giá và so sánh chất lượng của các tác phẩm nghệ thuật được tạo ra bằng mô hình GAN với các phương pháp truyền thống khác trong lĩnh vực nghệ thuật.

Bằng cách đạt được những mục tiêu nghiên cứu này, em hy vọng rằng sẽ đóng góp vào việc tăng cường hiểu biết về mô hình GAN và tiềm năng của nó trong lĩnh vực nghệ thuật, từ đó thúc đẩy sự phát triển và ứng dụng sáng tạo của công nghệ trong ngành nghệ thuật.

1.1.3. Giới hạn và phạm vi nghiên cứu

Nghiên cứu này sẽ tập trung vào việc tìm hiểu về mô hình học sâu GAN và ứng dụng của nó trong việc tạo ảnh nghệ thuật. Tuy nhiên, để giới hạn và tập trung nghiên cứu, dưới đây là một số hạn chế và phạm vi được xác định:

- Giới hạn về Mô hình GAN: Nghiên cứu sẽ tập trung vào một số mô hình GAN phổ biến và đang được sử dụng rộng rãi trong lĩnh vực nghệ thuật, chẳng hạn như DCGAN (Deep Convolutional GAN) và CGAN (Conditional GAN). Với mô hình DCGAN, em sẽ tập chung vào 1 thể loại duy nhất là tranh chân dung để tạo ra các bức tranh chân dung ở những màu sắc, góc độ, độ sắc nét khác nhau. Với mô hình CGAN pix2pix, em sẽ tập chung vào biến những nét phác thảo đơn giản thành những hình ảnh chi tiết hơn.
- Giới hạn về Ứng dụng: Trong phạm vi của nghiên cứu này, em sẽ chỉ tập trung vào việc tạo ra các tác phẩm nghệ thuật 2D, bao gồm tranh vẽ, ảnh chân dung, và các thể loại nghệ thuật khác dựa trên ảnh.

- Giới hạn về Dữ liệu: em sẽ sử dụng các tập dữ liệu có sẵn để huấn luyện mô hình GAN, không đi sâu vào việc thu thập dữ liệu mới hoặc xử lý dữ liệu.
- Giới hạn về Đánh giá: Phần lớn đánh giá về chất lượng của các tác phẩm nghệ thuật sẽ dựa trên các phương pháp định lượng, chẳng hạn như sử dụng các chỉ số đánh giá chất lượng ảnh và đánh giá từ người dùng.

Bằng cách xác định rõ ràng giới hạn và phạm vi nghiên cứu, em hy vọng rằng sẽ có một cái nhìn tổng quan và sâu sắc hơn về mô hình GAN và ứng dụng của nó trong việc tạo ra các tác phẩm nghệ thuật.

1.2. Bài toán

1.2.1. Phát biểu bài toán

Mô hình GAN (Generative Adversarial Networks) là một lớp mô hình học sâu được thiết kế để sinh ra dữ liệu mới, không chỉ giống với dữ liệu thực mà còn có thể vượt qua được các ràng buộc và giới hạn của dữ liệu gốc. Bài toán đặt ra là làm thế nào để mô hình GAN có thể tạo ra ảnh nghệ thuật có chất lượng cao, không chỉ giống với dữ liệu huấn luyện mà còn phải có độ đa dạng và tính sáng tạo, tính mới mẻ và độc đáo so với dữ liệu huấn luyện ban đầu.

Mô hình GAN bao gồm 2 thành phần chính là Generator để tạo ra ảnh mới, hay nói cách khác là tạo ra dữ liệu có độ tương tự cao so với dữ liệu huấn luyện và Discriminator dùng để dự đoán về việc liệu ảnh đó có phải là thật (được lấy từ dữ liệu huấn luyện) hay giả (do Generator tạo ra). Trong quá trình huấn luyện, Generator và Discriminator cạnh tranh với nhau. Generator cố gắng tạo ra ảnh tốt nhất để có thể đánh lừa Discriminator, trong khi Discriminator cố gắng học cách phân biệt giữa ảnh thật và ảnh giả. Quá trình này sẽ tiếp tục cho đến khi Generator tạo ra các ảnh mà Discriminator không thể phân biệt được với ảnh thật.

Với mô hình DCGAN, bài toán tập trung vào việc xây dựng một mô hình có khả năng tạo ra các biến thể hình ảnh nghệ thuật mới và đa dạng từ một bộ dữ liệu hình ảnh nghệ thuật ban đầu. Điều này bao gồm việc sinh ra các hình ảnh mới với các biến thể về màu sắc, ánh sáng, độ sắc nét và góc chụp khác nhau. Ví dụ với một nhiếp ảnh gia hoặc một họa sĩ thực hiện vẽ hoặc chụp một số bức ảnh về phong cảnh hoặc con người để đăng lên một bài báo, tuy nhiên dù đã bỏ ra rất nhiều thời gian và công sức nhưng chưa nhận được một bức ảnh phù hợp và việc bỏ thêm thời gian và công sức là bất khả thi thì họ có thể sử dụng mô hình DCGAN với dữ liệu đầu vào là những bức ảnh không hợp lệ, DCGAN sẽ học từ những đặc trưng của ảnh và sinh ra số lượng lớn những bức ảnh tương tự với màu sắc, ánh sáng, độ sắc nét và góc độ khác nhau để họa sĩ hoặc nhiếp ảnh gia có thể chọn ra một bức ảnh phù hợp với yêu cầu.

Với mô hình CGAN Pix2Pix, bài toán được đặt ra là làm thế nào để chuyển đổi các bản phác thảo hoặc concept design đơn giản thành các hình ảnh nghệ thuật chi tiết và chất lượng cao. Mô hình này đặc biệt hữu ích trong việc biến các ý tưởng sơ khai thành các sản phẩm hoàn thiện, giúp các nghệ sĩ, nhà thiết kế và các doanh nghiệp dễ dàng hình dung và hiện thực hóa ý tưởng của mình. Mục tiêu là tạo ra các hình ảnh không chỉ đẹp mắt mà còn có độ chính xác cao về chi tiết và màu sắc, mang lại giá trị thực tế trong các lĩnh vực như thiết kế sản phẩm, quảng cáo, minh họa sách và truyện tranh, và nhiều lĩnh vực khác. Ví dụ, khi một tác giả muốn vẽ những hình ảnh minh họa cho tác phẩm của mình nhưng khả năng, hiệu suất vẽ của bản thân còn hạn chế và không thể chi ra quá nhiều tiền cho họa sĩ vẽ thuê. Trong khi đó, CGAN Pix2Pix cung cấp một công cụ mạnh mẽ để tiết kiệm thời gian và công sức trong quá trình phát triển sản phẩm và tạo ra các bản vẽ minh họa rõ ràng và hấp dẫn hơn. Với việc áp dụng CGAN Pix2Pix, tác giả trên có thể tiết kiệm thời gian và tiền bạc mà vẫn có những bức tranh minh họa phù hợp cho bản thân.

Tổng hợp lại, cả DCGAN và CGAN Pix2Pix đều mang lại những giải pháp mạnh mẽ và sáng tạo trong việc tạo ra các hình ảnh nghệ thuật chất lượng cao, đa dạng và độc đáo, mở ra nhiều cơ hội ứng dụng thực tế và kinh doanh trong lĩnh vực nghệ thuật số và thiết kế. Tuy nhiên, để đạt được mục đích đưa vào sử dụng, cần phải có một số lượng dữ liệu lớn, đa dạng và phong phú cũng như tài nguyên phần cứng lớn để đưa vào huấn luyện. Vì vậy, để giải quyết bài toán với lượng thời gian, nguồn tài nguyên phần cứng phù hợp với bản thân, em quyết định chỉ thực hiện bài toán trong phạm vi sau:

- Chỉ sử dụng 1 thể loại ảnh cụ thể là ảnh tranh chân dung với số lượng ảnh là khoảng 5000 ảnh để huấn luyện mô hình DCGAN. Ảnh sinh ra sẽ là những bức tranh chân dung với góc độ, ánh sáng, màu sắc khác nhau.
- Chỉ sử dụng 1 thể loại ảnh cụ thể là ảnh phong cảnh với số lượng ảnh là 1500 ảnh để huấn luyện mô hình CGAN Pix2Pix. Ảnh sinh ra sẽ tốt nhất với những hình ảnh đầu vào là ảnh phác thảo phong cảnh, từ đó sinh ra những bức tranh phong cảnh chi tiết hơn về nét vẽ hoặc màu sắc.

1.2.2. Đầu vào và đầu ra

Với mô hình DCGAN:

- Đầu vào: Mô hình DCGAN nhận vào một vector ngẫu nhiên (noise) để làm điểm khởi đầu cho quá trình sinh ảnh
- Đầu ra: Mô hình sẽ sinh ra ảnh chân dung mới mới, với mục tiêu là những bức tranh chân dung với góc độ, ánh sáng, màu sắc khác nhau mà không thể phân biệt được đâu là ảnh do con người tạo ra và đâu là ảnh do mô hình GAN sinh ra

Với mô hình CGAN Pix2Pix:

- Đầu vào: Mô hình CGAN nhận vào một ảnh phác họa đơn giản để làm điểm khởi đầu cho quá trình sinh ảnh
- Đầu ra: Mô hình sẽ sinh ra ảnh mới chi tiết hơn từ ảnh phác họa, với mục tiêu là ảnh phải chi tiết hơn về nét vẽ hoặc màu sắc hơn ảnh nguyên bản.

1.2.3. Ràng buộc liên quan

Ràng buộc dữ liệu: Dữ liệu huấn luyện cần phải đa dạng và phong phú để mô hình có thể học được các đặc trưng nghệ thuật cần thiết.

Ràng buộc thời gian: Quá trình huấn luyện mô hình GAN có thể mất nhiều thời gian, đặc biệt là khi làm việc với lượng dữ liệu lớn và mô hình phức tạp.

Ràng buộc tài nguyên: Việc triển khai một mô hình CNN đòi hỏi tài nguyên tính toán lớn, đặc biệt khi sử dụng trên cơ sở hạ tầng có hạn. Nếu không có tài nguyên tính toán đủ lớn, việc huấn luyện và triển khai mô hình có thể trở thành vấn đề lớn.

Ràng buộc hiệu suất: Mô hình GAN cần đạt được sự cân bằng giữa Generator và Discriminator để tạo ra kết quả tốt nhất.

Ràng buộc độ chính xác: Ảnh nghệ thuật sinh ra cần phải có độ chính xác cao, không chỉ về mặt hình ảnh mà còn cả về phong cách nghệ thuật.

Ràng buộc về quyền sở hữu trí tuệ: Cần phải đảm bảo rằng ảnh nghệ thuật sinh ra không vi phạm bản quyền hay quyền sở hữu trí tuệ của các tác phẩm nghệ thuật đã có.

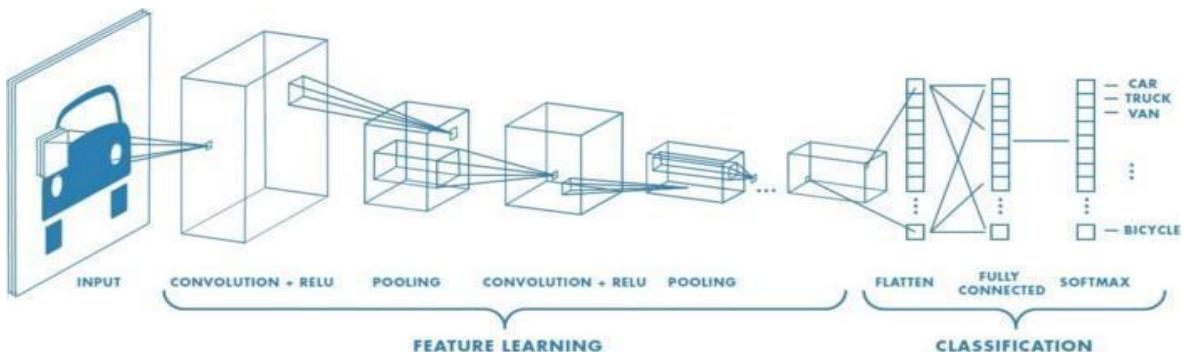
1.3. Một số cơ sở lý thuyết

1.3.1. Mạng neural tích chập (CNN)

Mô hình mạng neural tích chập (CNN) là một trong những mô hình được sử dụng rộng rãi để nhận dạng và phân loại hình ảnh. Trong các ứng

dụng cụ thể như xác định đối tượng và nhận dạng khuôn mặt, CNN đã chứng minh được hiệu quả vượt trội. CNN phân loại hình ảnh bằng cách nhận đầu vào là một hình ảnh, sau đó xử lý và phân loại nó theo các hạng mục nhất định. Máy tính coi hình ảnh đầu vào là một mảng pixel và tùy thuộc vào độ phân giải của hình ảnh, máy tính sẽ thấy $H \times W \times D$ (Height: Chiều cao, Width: Chiều rộng, Depth: Độ dày).

Kỹ thuật này trong mô hình CNN bao gồm quá trình huấn luyện và kiểm tra, trong đó mỗi hình ảnh đầu vào sẽ được chuyển qua một loạt các lớp tích chập với các bộ lọc (kernels), các lớp gộp và các lớp kết nối đầy đủ, cuối cùng áp dụng hàm Softmax để phân loại đối tượng dựa trên xác suất. Dưới đây là phần trình bày chi tiết về các lớp được sử dụng trong mô hình GAN:



Hình 1.2 Mô hình CNN [6]

Lớp tích chập - Convolution Layer

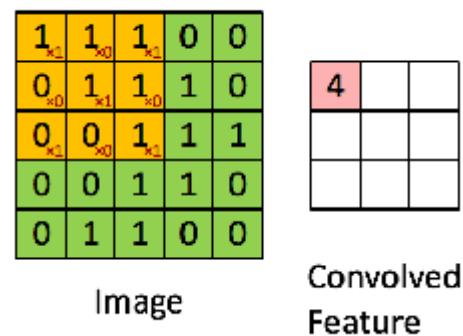
Lớp tích chập là lớp đầu tiên trong mạng CNN, chịu trách nhiệm trích xuất các đặc trưng từ hình ảnh đầu vào. Lớp này duy trì mối quan hệ không gian giữa các pixel bằng cách sử dụng các bộ lọc (kernels) để quét qua hình ảnh.

Xem xét 1 ma trận 5×5 có giá trị pixel là 0 và 1. Ma trận bộ lọc 3×3 như hình bên dưới.



Hình 1.3 Mô tả lớp tích chập – ma trận 5x5 và bộ lọc 3x3 [6]

Khi một ma trận đầu vào (ví dụ: 5x5) được nhân với một ma trận bộ lọc (ví dụ: 3x3), kết quả là một "feature map" như hình bên dưới.



Hình 1.4 Mô tả lớp tích chập – feature map [6]

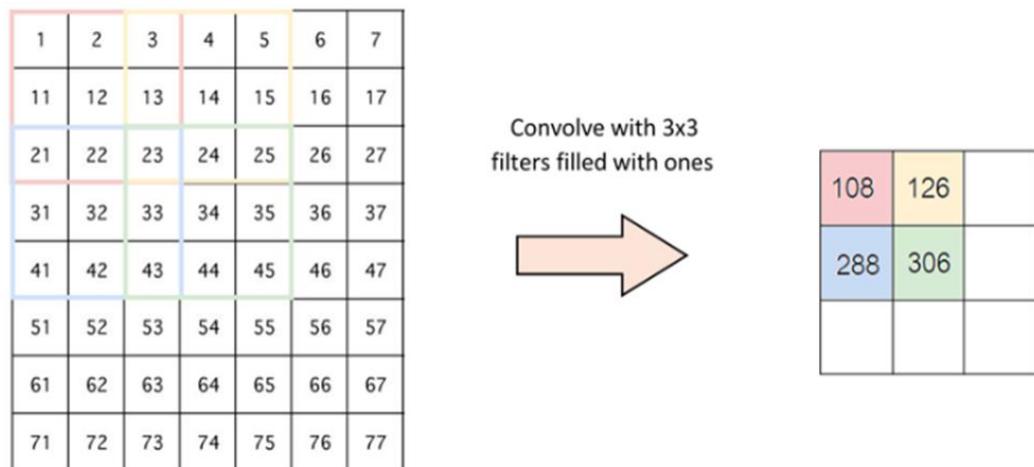
Quá trình này giúp phát hiện các đặc trưng cơ bản như cạnh và góc. Khi áp dụng các bộ lọc khác nhau, chúng ta có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét hình ảnh. Số lượng pixel mà kernel di chuyển trên ma trận đầu vào gọi là stride. Stride càng lớn thì kích thước feature map càng nhỏ. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Hình 1.5 Ví dụ các bộ lọc [6]

Bước nhảy – Stride

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.



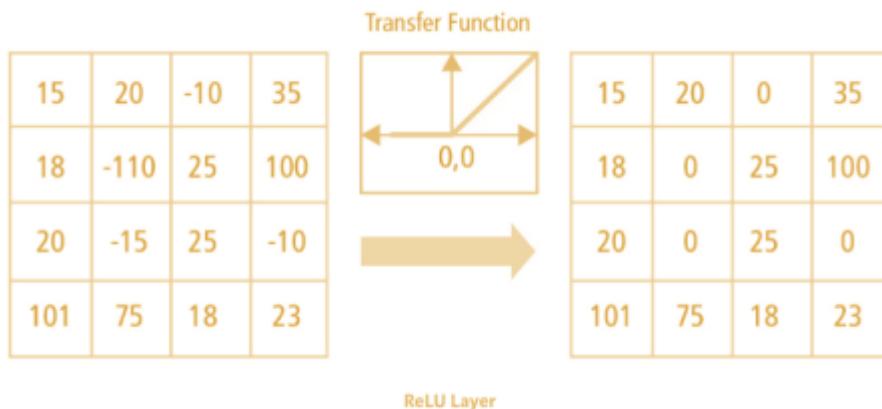
Hình 1.6 Mô tả lớp tích chập hoạt động với stride là 2 [6]

Đường viền - Padding

Padding được sử dụng để giải quyết vấn đề khi kernel không phù hợp với kích thước hình ảnh đầu vào. Có hai lựa chọn: chèn thêm các số 0 vào bốn đường biên của hình ảnh (zero-padding) hoặc cắt bớt hình ảnh tại những điểm không phù hợp với kernel. Padding giúp duy trì kích thước của feature map sau khi tích chập, đảm bảo không mất mát thông tin quan trọng ở các biên.

Hàm phi tuyến - ReLU

ReLU (Rectified Linear Unit) là một hàm phi tuyến được sử dụng phổ biến trong các lớp CNN. Hàm này có công thức $f(x) = \max(0, x)$, biến đổi tất cả các giá trị âm thành 0 để tránh ảnh hưởng cho việc tính toán ở các layer sau đó.



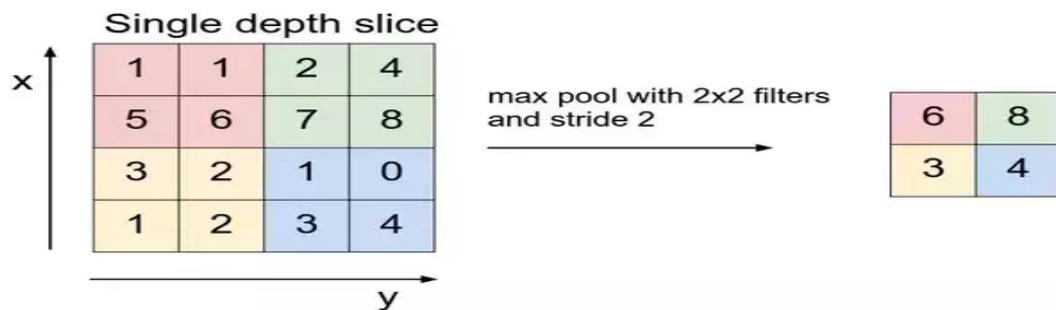
Hình 1.7 Mô tả hàm phi tuyến [6]

Ngoài ra, trong GAN thường sử dụng Leaky ReLU, một biến thể của ReLU, cho phép gradient nhỏ qua các giá trị âm để tránh hiện tượng "dying ReLU".

Lớp gộp - Pooling Layer

Lớp gộp (pooling layer) giúp giảm kích thước không gian của feature map, giảm số lượng tham số và tính toán trong mạng khi hình ảnh quá lớn.

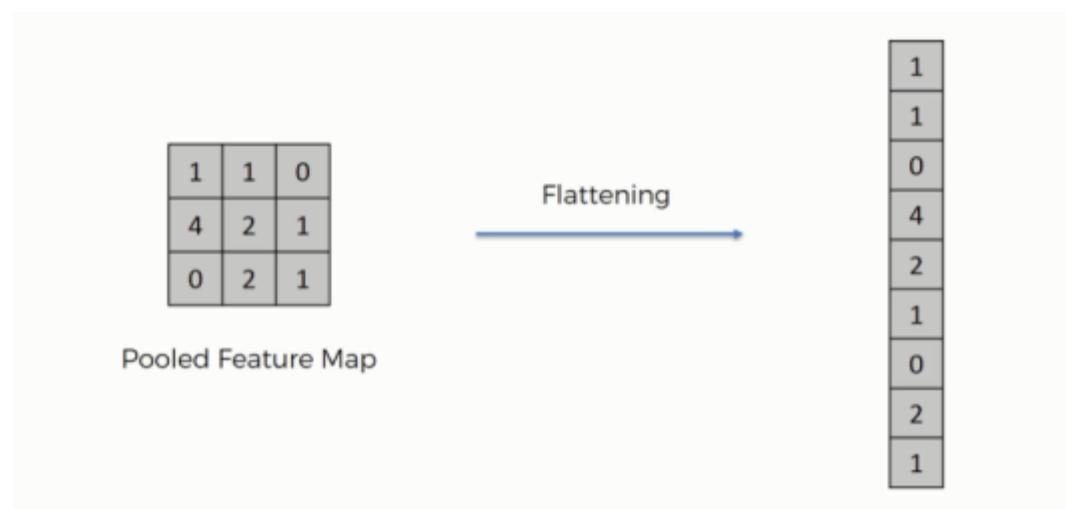
Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau nhưng hay được sử dụng nhất là Max pooling:



Hình 1.8 Ví dụ về lớp gộp [6]

Lớp kết nối đầy đủ - Fully Connected Layer

Lớp kết nối đầy đủ (Dense layer) kết nối tất cả các nơron của lớp trước đó với tất cả các nơron của lớp hiện tại. Trong mô hình GAN, lớp này được sử dụng ở cuối cùng của cả Generator và Discriminator. Trong Generator, lớp này giúp mở rộng và tăng kích thước vector đầu vào, còn trong Discriminator, nó giúp kết nối các đặc trưng không gian với lớp phân loại cuối cùng. Trước lớp fully connected, dữ liệu thường được chuyển đổi thành một vector một chiều bằng lớp Flatten.



Hình 1.9 Ví dụ về lớp kết nối đầy đủ [6]

Lớp Batch Normalization

Batch normalization là lớp được sử dụng để chuẩn hóa đầu ra của một lớp trước khi đưa vào lớp tiếp theo, giúp ổn định và tăng tốc quá trình huấn luyện. Lớp này chuẩn hóa đầu ra của lớp trước theo phân phối chuẩn, giảm hiện tượng gradient vanishing và exploding.

Lớp Transposed Convolution (Deconvolution)

Lớp transposed convolution, còn gọi là deconvolution, được sử dụng trong Generator để tăng kích thước không gian của đầu vào, chuyển đổi một vector nhiễu thành một hình ảnh lớn hơn. Lớp này thực hiện quá trình ngược lại của lớp convolution, tạo ra các hình ảnh chi tiết từ dữ liệu đầu vào có kích thước nhỏ.

Lớp Activation - Tanh và Sigmoid

Các lớp activation như Tanh và Sigmoid thường được sử dụng ở đầu ra của Generator và Discriminator. Lớp Tanh chuẩn hóa đầu ra thành các giá trị trong khoảng [-1, 1], trong khi lớp Sigmoid chuẩn hóa đầu ra thành các giá trị trong khoảng [0, 1]. Điều này giúp điều chỉnh giá trị đầu ra của mô hình phù hợp với yêu cầu cụ thể của bài toán.

Những lớp này kết hợp lại để tạo nên kiến trúc của GAN, trong đó Generator cố gắng tạo ra hình ảnh giả nhưng giống thật, và Discriminator cố gắng phân biệt giữa hình ảnh thật và giả. Quá trình huấn luyện liên tục này giúp GAN có khả năng sinh ra các hình ảnh mới có chất lượng cao, đa dạng và sáng tạo.

1.3.2. Lan truyền ngược (Backpropagation)

Lan truyền ngược (backpropagation) là giải thuật cốt lõi giúp cho các mô hình học sâu có thể dễ dàng thực thi tính toán được. Với các mạng NN hiện đại, nhờ giải thuật này mà thuật toán tối ưu với đạo hàm (gradient descent) có thể nhanh hơn hàng triệu lần so với cách thực hiện truyền thống. Ví dụ 1 mô hình với lan truyền ngược chạy mất 1 tuần thì có thể mất tới 200,000 năm để huấn luyện với phương pháp truyền thống!

Về cơ bản, nó là một kỹ thuật để nhanh chóng tính được đạo hàm của hàm mất mát theo các tham số trong mô hình machine learning khi áp dụng thuật toán gradient descent. Trong mô hình machine learning thì số lượng tham số có thể là rất lớn, đôi khi lên đến hàng triệu tham số. Tính toán đạo hàm của hàm mất mát theo từng tham số bằng cách sử dụng công thức tính đạo hàm bình thường có thể rất phức tạp và tốn kém về thời gian và chi phí tính toán. Thuật toán lan truyền ngược được thiết kế để tính toán đạo hàm của hàm mất mát theo từng tham số một cách hiệu quả bằng cách sử dụng kỹ thuật lan truyền ngược thông qua các lớp (layers) của mô hình. Quá trình tính toán đạo hàm này giúp cho thuật toán gradient descent điều chỉnh các tham số mô hình sao cho giá trị của hàm mất mát giảm dần theo thời gian, giúp em ưu hóa hiệu suất huấn luyện của mô hình.

Trong mạng GAN, lan truyền ngược (Backpropagation) chịu trách nhiệm trong quá trình huấn luyện cả generator và discriminator. Mục tiêu của generator là tạo ra các hình ảnh mới một cách chân thực, trong khi discriminator cố gắng phân biệt giữa các hình ảnh thật và giả. Quá trình lan truyền ngược giúp cập nhật các trọng số của cả hai mạng con dựa trên độ lỗi của chúng.

Trong generator, lan truyền ngược tính toán độ lỗi giữa hình ảnh được tạo ra và các nhãn "đúng", ví dụ nhãn "1" cho hình ảnh giả. Sau đó, độ lỗi này được truyền ngược từ đầu ra đến các lớp trước để cập nhật trọng số. Điều này giúp generator học được cách tạo ra các hình ảnh sao cho discriminator đánh lừa được.

Trong khi đó, discriminator cũng sử dụng lan truyền ngược để tính toán độ lỗi giữa dự đoán của mạng và các nhãn "đúng" cho hình ảnh thật và giả. Các độ lỗi này sau đó được truyền ngược để cập nhật trọng số của mạng phân biệt, giúp nó phân loại chính xác hơn.

Quá trình cập nhật trọng số trong GAN thường được thực hiện bằng cách sử dụng các phương pháp tối ưu hóa như Gradient Descent hoặc Adam Optimization. Điều này giúp cả hai mạng con học từ dữ liệu và cải thiện chất lượng của các hình ảnh được tạo ra.

Tuy nhiên, trong mạng GAN, cần phải cân nhắc kỹ lưỡng về việc cập nhật trọng số để đảm bảo rằng discriminator không trở nên quá mạnh để dễ dàng phân biệt giữa hình ảnh thật và giả, trong khi generator vẫn đủ mạnh để tạo ra các hình ảnh "đánh lừa" được discriminator.

CHƯƠNG 2. PHƯƠNG PHÁP ỨNG DỤNG

2.1. Giới thiệu về GAN

Mạng đối nghịch tạo sinh (hay còn gọi là Mạng đối nghịch phát sinh), tiếng Anh là Generative Adversarial Network (GAN) là một mô hình học sâu thuộc loại Generative Model. Cụ thể, các mô hình Machine Learning có thể được phân chia thành 2 loại là lớp Mô hình phân biệt (Discriminative model) và mô hình sinh (Generative model). Đây chỉ là một cách phân chia trong vô số các cách phân chia khác như: mô hình học có giám sát (*supervised learning*)/học không giám sát (*unsupervised learning*), mô hình tham số (*parametric*)/mô hình phi tham số (*non-parametric*), mô hình đồ thị (*graphic*)/mô hình phi đồ thị (*non-graphic*),.... [9]

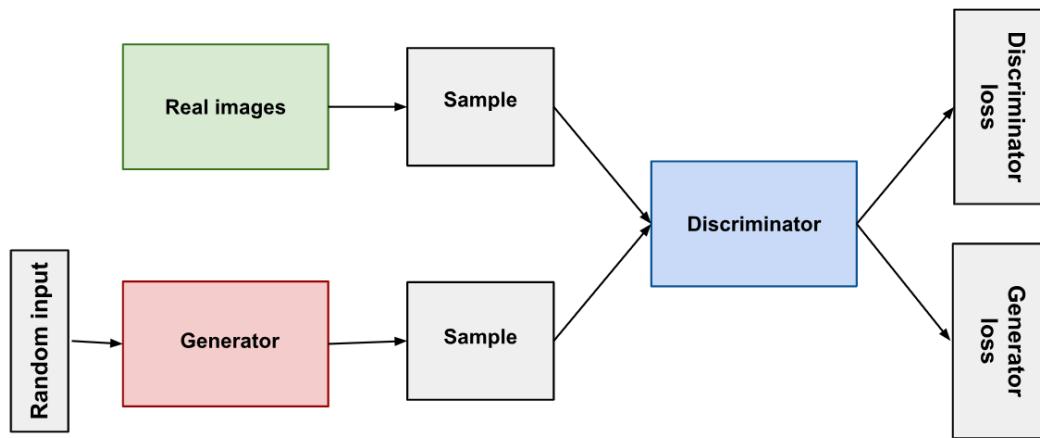
Mô hình phân biệt, được sử dụng cho phân loại hoặc dự đoán, sẽ dựa trên các biến đầu vào x để dự báo nhãn hoặc giá trị y . Nó có găng học mối quan hệ giữa các biến đầu vào và nhãn để phân loại chính xác dữ liệu. Mô hình dự báo đầu ra dựa trên các dấu hiệu là đầu vào đã biết và các giá trị dự báo là một xác suất có điều kiện: $P(y|x)$, trong đó y là mục tiêu cần dự báo và x được xem như điều kiện.

Một lớp mô hình khác làm nhiệm vụ trái ngược lại với mô hình phân biệt là mô hình sinh. Mô hình sinh học cách dữ liệu được tạo ra từ nhãn đã biết và sử dụng điều này để tạo ra mẫu dữ liệu mới. Xác suất $P(x|y)$ thể hiện xác suất của dữ liệu x được tạo ra dựa trên nhãn y .

Các mô hình sinh được chia thành 2 loại chính là mô hình hiện (Explicit model) và mô hình ẩn (Implicit model). Trong đó, mô hình hiện sử dụng các hàm phân phối xác suất lý thuyết để sinh mẫu dữ liệu còn mô hình ẩn thì mẫu sinh được sinh ra từ mô hình. Và GAN model là một loại mô hình thuộc loại mô hình ẩn.

GAN là một mô hình implicit generative được giới thiệu bởi Ian Goodfellow và các đồng nghiệp vào năm 2014. Như đã nói ở phần trên, GAN

bao gồm hai thành phần chính: Generator và Discriminator. Generator có gắng tạo ra dữ liệu mới từ một không gian ngẫu nhiên, trong khi Discriminator có gắng phân biệt giữa dữ liệu thật và dữ liệu được tạo ra bởi Generator. Sự cạnh tranh giữa Generator và Discriminator trong quá trình huấn luyện làm cho Generator học cách tạo ra dữ liệu mà có vẻ giống với dữ liệu thực sự, trong khi Discriminator học cách phân biệt giữa dữ liệu thật và dữ liệu giả. Quá trình này tiếp tục cho đến khi Generator tạo ra dữ liệu mà không thể được phân biệt với dữ liệu thực sự.



Hình 2.1 Kiến trúc của mô hình GAN [7]

2.2. Cấu trúc mạng GAN

2.2.1. Discriminator

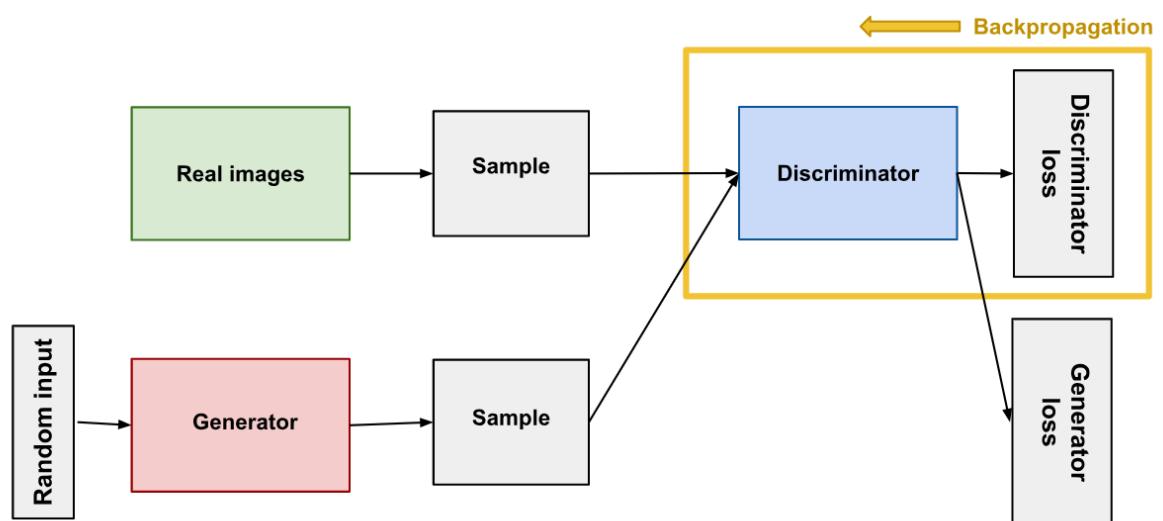
Discriminator là một thành phần của GAN có nhiệm vụ phân biệt giữa dữ liệu thật và dữ liệu được tạo ra bởi Generator. Tùy thuộc vào loại dữ liệu mà nó đang phân loại, Discriminator có thể sử dụng bất kỳ kiến trúc mạng phù hợp nào.

Dữ liệu huấn luyện của Discriminator bao gồm hai loại:

- Các trường hợp dữ liệu thật: Đây là các ví dụ thực tế từ tập dữ liệu, như các hình ảnh thực của con người. Discriminator sử dụng những ví dụ này như là các ví dụ tích cực trong quá trình huấn luyện.

- Các trường hợp dữ liệu giả: Đây là các trường hợp được tạo ra bởi Generator. Discriminator xem chúng như các ví dụ tiêu cực trong quá trình huấn luyện.

Trong quá trình huấn luyện, Discriminator nhận đầu vào từ cả hai nguồn, hai hộp "Sample" đại diện cho hai nguồn dữ liệu này đang cung cấp vào bộ phân loại. Trong quá trình huấn luyện của Discriminator, Generator không được huấn luyện. Các trọng số của nó vẫn cố định trong khi nó tạo ra các ví dụ cho Discriminator để huấn luyện.



Hình 2.2 Kiến trúc mô hình GAN – Discriminator [7]

Trong quá trình huấn luyện của Discriminator, Generator không cập nhật trọng số của mình. Trọng số của Generator duy trì không đổi trong khi nó tạo ra các ví dụ cho Discriminator huấn luyện. Discriminator, ngược lại, cập nhật trọng số của mình dựa trên măt măt phát sinh từ nhiệm vụ phân loại. Trong đó:

Trong đó:

- Classifies (Phân loại): Discriminator phân loại cả dữ liệu thật và dữ liệu giả.
 - Discriminator loss (mất mát): Discriminator bị phạt nếu phân loại sai một trường hợp thật thành giả hoặc một trường hợp giả thành

thật. Mất mát này được tính dựa trên sự khác biệt giữa dự đoán của Discriminator và nhãn đúng.

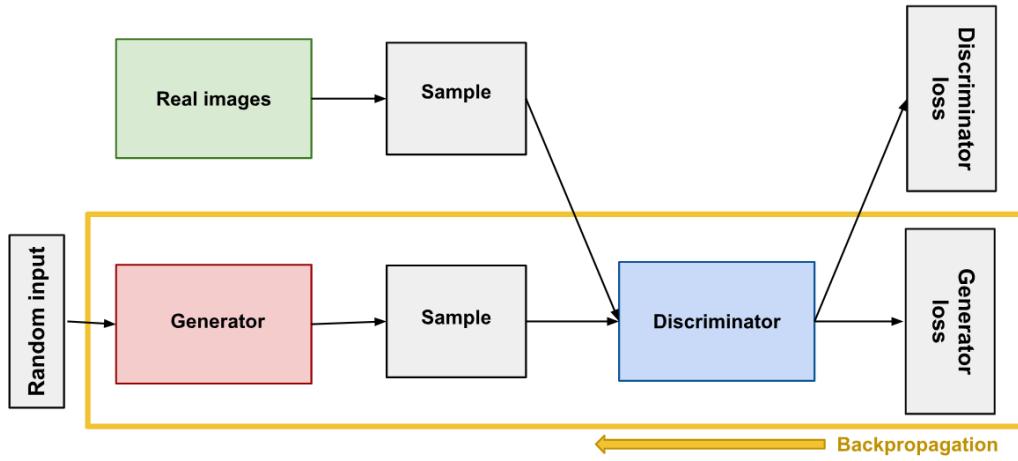
- Backpropagation (Lan truyền ngược): Discriminator cập nhật trọng số của mình thông qua lan truyền ngược từ mất mát của Discriminator qua mạng Discriminator.

Tổng kết lại, mất mát của Generator được bỏ qua, chỉ có mất mát của Discriminator được xem xét. Mất mát của Generator được sử dụng trong quá trình huấn luyện của Generator, như được mô tả trong phần tiếp theo.

2.2.2. Generator

Generator là bộ phận của GAN có nhiệm vụ học cách tạo ra dữ liệu giả bằng cách tích hợp phản hồi từ Discriminator. Mục tiêu chính của Generator là đánh lừa Discriminator rằng dữ liệu mà nó tạo ra là thật. Việc huấn luyện Generator cần sự tích hợp chặt chẽ hơn giữa Generator và Discriminator so với việc huấn luyện Discriminator. Phần huấn luyện Generator trong GAN sẽ bao gồm:

- Đầu vào ngẫu nhiên
- Mạng Generator, có nhiệm vụ biến đổi đầu vào ngẫu nhiên thành 1 trường dữ liệu
- Mạng Discriminator, phân loại dữ liệu được tạo ra
- Đầu ra của Discriminator
- Hàm mất mát của Generator, phạt Generator khi không làm cho Discriminator bị đánh lừa.



Hình 2.3 Kiến trúc mô hình GAN – Generator [7]

Để huấn luyện Generator trong mạng GAN, chúng ta sử dụng Discriminator như là một phần quan trọng của quá trình. Quá trình này bao gồm các bước sau:

- Lấy mẫu nhiễu (noise) ngẫu nhiên: Đầu tiên, chúng ta lấy mẫu nhiễu ngẫu nhiên từ không gian nhiễu, thường là một phân phối đồng đều. Nี่ sẽ được sử dụng làm đầu vào cho bộ sinh để tạo ra dữ liệu giả.
- Tạo ra đầu ra của Generator: Generator nhận nhiễu ngẫu nhiên này và biến nó thành một mẫu dữ liệu giả. Generator học cách tạo ra dữ liệu mới mà bộ phân biệt sẽ phân loại như là dữ liệu thực.
- Nhận phân loại từ Discriminator: Generator tạo ra dữ liệu giả và đưa nó vào Discriminator. Bộ phân biệt sau đó phân loại nó là "thật" hoặc "giả" dựa trên độ tin cậy của nó.
- Tính toán mất mát từ phân loại của Discriminator: Mất mát được tính toán dựa trên cách phân loại của Discriminator. Generator bị phạt nếu dữ liệu giả được phân loại sai.
- Lan truyền ngược (Backpropagation) qua cả Discriminator và Generator để thu được độ dốc: Độ dốc được thu được từ mất mát của Discriminator được lan truyền ngược qua cả Discriminator và Generator. Điều này đảm bảo rằng cả hai mạng đều học từ nhau.

- Sử dụng độ dốc để thay đổi chỉ trọng số của Generator: Cuối cùng, chúng ta sử dụng độ dốc tính được để điều chỉnh các trọng số của Generator. Điều này làm cho Generator học cách tạo ra dữ liệu mới một cách hiệu quả hơn, sao cho Discriminator khó phân biệt được dữ liệu giả và dữ liệu thật.

Quá trình này tiếp tục lặp lại trong mỗi vòng lặp của quá trình huấn luyện GAN, giúp cả hai mạng học từ nhau và cải thiện theo thời gian.

2.2.3. *Loss function*

Trong mạng đối nghịch tạo sinh (GAN), hàm mất mát (loss function) là một phần cực kỳ quan trọng, đóng vai trò trong việc định hình hiệu suất của cả bộ sinh (generator) và bộ phân biệt (discriminator). Hàm mất mát này được thiết kế để đảm bảo rằng generator tạo ra dữ liệu giả một cách hiệu quả và discriminator có khả năng phân biệt giữa dữ liệu thật và dữ liệu giả, được tính bằng công thức:

$$\min_G \max_D V(D, G) = \underbrace{\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)]}_{\text{log-probability that D predict } x \text{ is real}} + \underbrace{\mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]}_{\text{log-probability D predicts } G(z) \text{ is fake}}$$

Trong đó:

- Kí hiệu z là noise đầu vào của generator, x là dữ liệu thật từ bộ dataset.
- Kí hiệu mạng Generator là G, mạng Discriminator là D.
- G(z) là dữ liệu giả được tạo ra bởi bộ sinh từ nhiễu z (ảnh được sinh ra từ Generator).
- D(x) là đầu ra của bộ phân biệt cho dữ liệu thật, tức là giá trị dự đoán của Discriminator xem ảnh x là thật hay không.
- D(G(z)) là đầu ra của bộ phân biệt cho dữ liệu được tạo ra bởi bộ sinh từ nhiễu z, tức là giá trị dự đoán xem ảnh sinh ra từ Generator là ảnh thật hay không.

- E là kỳ vọng, hiểu đơn giản là lấy trung bình của tất cả dữ liệu, hay maximize D(x) với x là dữ liệu trong training set.

Thực chất, vì ta có 2 mạng Generator và Discriminator với mục tiêu khác nhau, nên cần thiết kế 2 loss function cho mỗi mạng và ảnh trên chính là một hàm kết hợp đồng thời giữa mục tiêu của Discriminator và mục tiêu của Generator.

Trong giai đoạn huấn luyện Discriminator, mục tiêu của giai đoạn này là huấn luyện một mô hình Discriminator sao cho khả năng phân loại là tốt nhất. Ở phase này chúng ta hãy tạm thời coi G không đổi và chỉ quan tâm đến $\max_D V(D, G)$. Đây là chính là đối của hàm cross entropy đối với trường hợp phân loại nhị phân. Để rõ ràng hơn thì mục tiêu của logistic regression đối với bài toán phân loại nhị phân là tối thiểu hóa một hàm cross entropy như sau:

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log p(y_i | \mathbf{x}_i) + (1 - y_i) \log(1 - p(y_i | \mathbf{x}_i))]$$

Trong đó $p(y_i | \mathbf{x}_i)$ là xác suất dự báo nhãn y_i từ mô hình logistic.

Gọi 2 phương trình trên lần lượt là phương trình (1) và phương trình (2). Trong phương trình (1) thì $D(x)$ cũng như $p(y_i | \mathbf{x}_i)$. Hay nói cách khác $D(x)$ đóng vai trò dự báo xác suất cho dữ liệu đầu vào. Và ta có hai khả năng xảy ra:

- Nếu đầu vào là ảnh thật thì $y_i = 1$ và $1 - y_i = 0$ và do đó loss function tương ứng với $y_i \log p(y_i | \mathbf{x}_i) = \log p(y_i | \mathbf{x}_i)$ ở phương trình (2). Giá trị này được coi như là $\log D(x)$ ở phương trình (1). Kí hiệu $x \sim p_{data(x)}$ ở phương trình (1) là phân phối xác suất của các điểm dữ liệu đầu vào, trong trường hợp ở phương trình (2) thì các quan sát có vai trò như nhau nên chúng có chung giá trị phân phối là $\frac{1}{N}$.

- Trường hợp ảnh đầu vào là giả thì $y_i = 0$ và $1 - y_i = 1$. Khi đó đóng góp vào hàm loss function chỉ còn thành phần $(1 - y_i) \log(1 - p(y_i|x_i)) = \log(1 - p(y_i|x_i))$ ở phương trình (2). Giá trị này được coi như là $\log(1 - D(G(z)))$ ở phương trình (1).

Trở lại với loss function của Discriminator, Discriminator sẽ được train để input ảnh ở dataset thì output gần 1, còn input là ảnh sinh ra từ Generator thì output gần 0, hay $D(x) \rightarrow 1$ còn $D(G(z)) \rightarrow 0$. Hay nói cách khác là loss function muốn maximize $D(x)$ và minimize $D(G(z))$. Ta có minimize $D(G(z))$ tương đương với maximize $(1 - D(G(z)))$. Do đó loss function của Discriminator trong paper gốc được viết lại thành:

$$\max_D V(D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

recognize real images better
recognize generated images better

Trong giai đoạn huấn luyện Generator: Mục tiêu của giai đoạn này là củng cố khả năng tạo ảnh của Generator sao cho ảnh nó sinh ra là giống với thật nhất. Ở giai đoạn này ta coi như D là không đổi và chỉ quan tâm đến $G(z)$ sao cho giá trị dự báo xác suất từ D đối với nó gần bằng 1 nhất, tức là ảnh giả được sinh ra giống ảnh thật nhất (xác suất càng gần 1 thì khả năng giống ảnh thật càng lớn). Như vậy $D(G(z))$ sẽ càng lớn càng tốt. Đảo dấu của nó trong $\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(z)))]$ ta suy ra mục tiêu cần tối ưu là tối thiểu hóa $\min_G V(D, G)$.

$$\min_G V(G) = \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

Optimize G that can fool the discriminator the most.

Từ hàm loss của GAN có thể thấy là việc train Generator và Discriminator đối nghịch nhau, trong khi D cố gắng maximize loss thì G cố

gắng minimize loss. Quá trình train GAN kết thúc khi model GAN đạt đến trạng thái cân bằng của 2 models, gọi là Nash equilibrium. [10]

2.3. Một số mô hình biến thể của GAN

2.3.1. GAN cơ bản

Như đã giới thiệu ở các phần trên, cấu trúc của một mạng đối nghịch tạo sinh cơ bản sẽ là Generator và Discriminator. Trong đó, Generator sẽ có đầu vào là một vector ngẫu nhiên từ không gian ẩn có kích thước cố định. Generator thường bắt đầu với 1 lớp Dense (fully connected layer) để chuyển đổi vector đầu vào thành một tensor có kích thước phù hợp để tiếp tục xử lý. Trong trường hợp Generator cần tạo ảnh có kích thước cố định, một lớp Reshape có thể được sử dụng để chuyển tensor từ một vector thành một tensor 2D hoặc 3D. Sau đó, các lớp tích chập ngược (transposed convolutional layers) hoặc lớp upsampling được sử dụng để tạo ra ảnh từ tensor 2D hoặc 3D. Các lớp này có thể được xếp chồng lên nhau để tăng độ phức tạp và chiều sâu của ảnh. Một lớp kích hoạt, thường là Tanh, được sử dụng để đảm bảo rằng giá trị của các pixel trong ảnh được tạo ra nằm trong khoảng [-1, 1]. Cuối cùng, đầu ra của Generator là một ảnh. Tiếp theo là về Discriminator. Đầu vào sẽ là một ảnh thật hoặc ảnh được tạo ra bởi Generator. Discriminator sử dụng các lớp tích chập (convolutional layers) để trích xuất các đặc trưng của ảnh đầu vào. Các lớp này thường được thiết kế với số lượng và kích thước bộ lọc khác nhau để học các đặc trưng cục bộ và toàn cục của ảnh. Một số kiến trúc Discriminator có thể sử dụng lớp pooling để giảm kích thước của tensor và tăng tính tổng quát của mô hình. Một lớp kích hoạt, thường là LeakyReLU, được sử dụng sau mỗi lớp tích chập để đưa ra đầu ra không âm. Cuối cùng, một hoặc một số lớp fully connected được sử dụng để biến đổi đầu ra từ các lớp tích chập thành một đầu ra duy nhất, thường là một số thực trong khoảng [0, 1], đại diện cho xác suất ảnh đầu vào là thật hoặc giả. Thông thường, hàm kích hoạt sigmoid được sử dụng để đảm bảo rằng đầu ra nằm trong phạm vi [0, 1].

Ưu điểm của GAN truyền thống:

- GAN có khả năng tạo ra ảnh mới chất lượng cao, giúp giải quyết các vấn đề trong lĩnh vực tạo ảnh nghệ thuật và sinh học.
- Mô hình không cần giả định về phân phối của dữ liệu đầu vào, có thể học được các phân phối phức tạp.

Nhược điểm của GAN truyền thống:

- Quá trình huấn luyện GAN có thể khá không ổn định và dễ dàng rơi vào các vấn đề như mode collapse.
- Cần có một lượng lớn dữ liệu để huấn luyện mô hình một cách hiệu quả.
- Các mô hình GAN có thể tạo ra ảnh không thật một cách khó kiểm soát, đặc biệt là khi dữ liệu huấn luyện không đủ đa dạng.

2.3.2. DCGAN

DCGAN là một biến thể của GAN được thiết kế đặc biệt để tạo ra ảnh chất lượng cao thông qua việc sử dụng các lớp tích chập (convolutional layers) trong cả Generator và Discriminator.

Dựa trên kiến trúc của GAN truyền thống, mô hình DCGAN (Deep Convolutional GAN) điều chỉnh và cải tiến kiến trúc của cả Generator và Discriminator để tối ưu hóa hiệu suất và chất lượng ảnh được tạo ra. So với GAN truyền thống, DCGAN chỉ thay đổi cấu tạo của Generator và Discriminator từ neural network truyền thống sang sử dụng CNN. DCGAN thường sử dụng batch normalization để ổn định quá trình huấn luyện và giảm thiểu vấn đề về khả năng học của mô hình. Trong Generator của DCGAN, không sử dụng các lớp kích hoạt như ReLU trước các lớp tích chập ngược để tránh vấn đề vanishing gradient.

Ưu điểm của DCGAN:

- Tạo ra ảnh giả có chất lượng cao hơn so với các mô hình trước đó.
- Sử dụng mạng neural tích chập (CNN) cho cả Generator và Discriminator giúp cải thiện chất lượng ảnh giả.

Nhược điểm của DCGAN:

- DCGAN đòi hỏi nhiều dữ liệu huấn luyện và thời gian huấn luyện lâu hơn so với một số mô hình khác.
- DCGAN nhạy cảm với các tham số và siêu tham số, việc lựa chọn tham số không phù hợp có thể dẫn đến việc mô hình không huấn luyện hiệu quả hoặc sinh ra ảnh kém chất lượng.
- DCGAN sinh ảnh một cách ngẫu nhiên và không thể kiểm soát.

2.3.3. CGAN

Như đã nói ở phần trên, DCGAN sẽ tạo ảnh một cách ngẫu nhiên mà chúng ta không thể kiểm soát được bức ảnh được sinh ra. Trong kiến trúc của DCGAN, đầu vào mạng là một vector noise z ngẫu nhiên và mỗi lần gọi thì có thể tạo ra một số khác nhau. Đó là một hạn chế rất lớn của DCGAN. Và đó là lý do Conditional GAN được sinh ra. CGAN sẽ giúp chúng ta sinh ra được ảnh cụ thể theo một điều kiện đầu vào.

Conditional Generative Adversarial Networks (CGAN) là một biến thể của mô hình Generative Adversarial Networks (GAN) giúp cải thiện khả năng kiểm soát và hướng dẫn quá trình sinh dữ liệu bằng cách đưa vào điều kiện cụ thể. Trong CGAN, điều kiện này có thể là một vector đặc trưng, một ảnh, hoặc bất kỳ thông tin phụ trợ nào mà Generator có thể sử dụng để tạo ra dữ liệu mong muốn. [7]

Điểm khác biệt giữa CGAN so với DCGAN là vector ngẫu nhiên đầu vào của Generator được nối với vector điều kiện để tạo thành 1 đầu vào kết hợp. Đồng thời ảnh đầu vào của Discriminator cũng sẽ được nối với điều kiện trước khi đưa vào.

Ưu điểm của CGAN:

- CGAN cho phép kiểm soát chất lượng và đa dạng của dữ liệu được tạo ra thông qua điều kiện đầu vào, điều này là một điểm mạnh so với GAN truyền thống và DCGAN.
- CGAN có khả năng tương tác cao với điều kiện đầu vào, giúp tạo ra dữ liệu theo yêu cầu cụ thể, điều này là một điểm mạnh so với GAN truyền thống và DCGAN.

Nhược điểm của CGAN:

- CGAN cần một lượng lớn dữ liệu huấn luyện để tạo ra dữ liệu chất lượng cao và đa dạng.
- Quá trình huấn luyện CGAN có thể phức tạp hơn so với GAN truyền thống do phải xử lý thêm điều kiện đầu vào.

2.4. Ứng dụng vào tạo ảnh nghệ thuật

2.4.1. Áp dụng DCGAN vào tạo ảnh nghệ thuật

Để huấn luyện một DCGAN (Deep Convolutional Generative Adversarial Network) và áp dụng nó để tạo ra ảnh nghệ thuật, chương trình sẽ được thực hiện như sau:

Chuẩn bị dữ liệu:

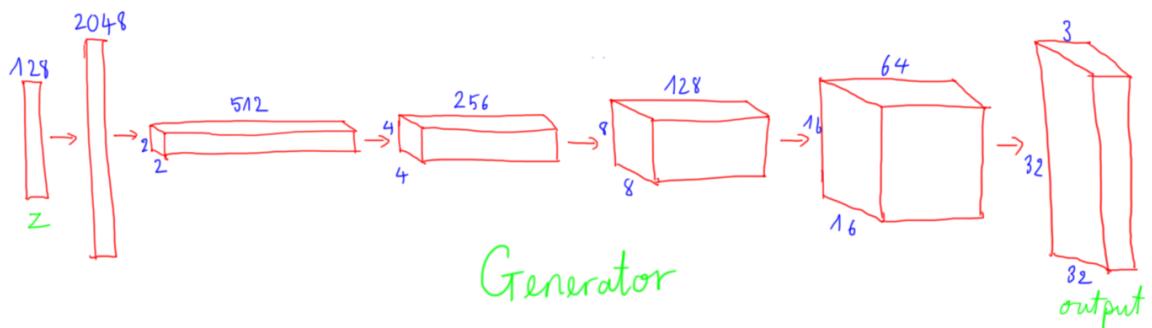
- Đầu tiên, cần chuẩn bị dữ liệu hình ảnh. Hình ảnh được resize về kích thước cố định (ở đây là 64x64 pixels) và chuẩn hóa về khoảng giá trị [-1, 1].
- Dữ liệu sau đó được lưu lại dưới dạng tệp nhị phân để tăng tốc quá trình tải và xử lý hình ảnh giữa các lần chạy.

Xây dựng mô hình DCGAN:

- Với Generator: Generator bắt đầu với một vector ngẫu nhiên có kích thước là SEED_SIZE (100). Vector này được truyền qua một lớp Dense để tạo ra một tensor 1D với kích thước là 4x4x512. Tiếp theo,

tensor này được chuyển đổi qua lớp Reshape để biến đổi thành một tensor 3D với hình dạng (4, 4, 512). Các lớp Conv2DTranspose sau đó tăng kích thước của tensor đầu ra để tạo ra hình ảnh cuối cùng. Mỗi lớp Conv2DTranspose sử dụng các tham số như sau:

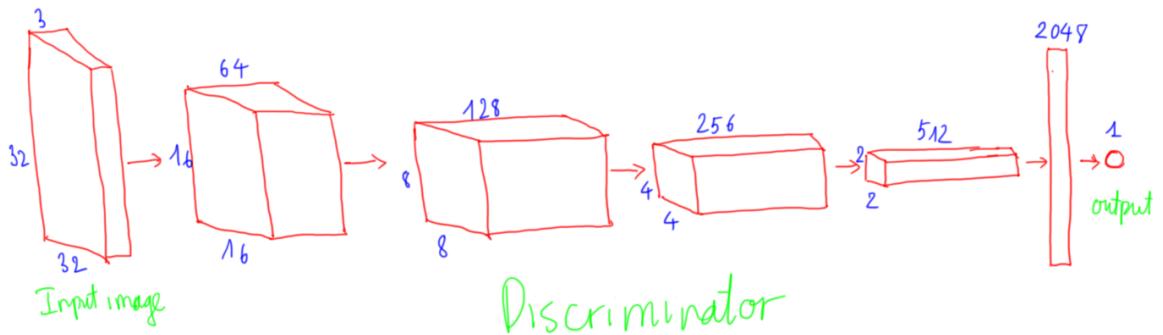
- + Lớp Conv2DTranspose với 512 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2DTranspose với 256 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2DTranspose với 64 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2DTranspose với 32 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2DTranspose cuối cùng với 3 filters (số kênh ảnh), kernel size 5x5, stride 1x1, padding same và hàm kích hoạt tanh.
- + Kết quả là một hình ảnh được tạo ra có kích thước (64, 64, 3), trong đó mỗi giá trị pixel nằm trong khoảng [-1, 1].



Hình 2.4 Mô tả kiến trúc của Generator trong DCGAN [9]

- Với Discriminator: Discriminator nhận vào một hình ảnh có kích thước (64, 64, 3) và xử lý nó qua các lớp Conv2D để giảm kích thước của tensor đầu vào. Mỗi lớp Conv2D sử dụng các tham số như sau:
 - + Lớp Conv2D với 64 filters, kernel size 5x5, stride 2x2, padding same và LeakyReLU.

- + Lớp Conv2D với 128 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2D với 256 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Lớp Conv2D với 512 filters, kernel size 5x5, stride 2x2, padding same, BatchNormalization và LeakyReLU.
- + Sau đó, tensor kết quả được chuyển qua lớp Flatten để biến đổi thành một vector 1D. Cuối cùng, vector này được đưa vào một lớp Dense cuối cùng với 1 unit và hàm kích hoạt sigmoid, tạo ra đầu ra là xác suất rằng hình ảnh đầu vào là thật.



Hình 2.5 Mô tả kiến trúc của Discriminator trong DCGAN [9]

Định nghĩa hàm loss và quá trình huấn luyện:

- Hàm loss cho Generator và Discriminator:
 - + Generator Loss: Generator loss được định nghĩa bằng binary crossentropy giữa đầu ra của Discriminator và nhãn "thật" (hoặc có thể đặt là "1"). Mục tiêu của Generator là tạo ra hình ảnh giả mà Discriminator phân loại như hình ảnh thật, do đó nó cố gắng làm cho đầu ra của Discriminator gần với 1.
 - + Discriminator Loss: Discriminator loss bao gồm hai phần là Phần loss từ việc phân loại hình ảnh thật (binary crossentropy giữa đầu ra của Discriminator cho hình ảnh thật và nhãn "thật") và phần loss từ việc phân loại hình ảnh giả (binary crossentropy giữa đầu

ra của Discriminator cho hình ảnh giả và nhãn "giả"). Mục tiêu của Discriminator là phân biệt chính xác giữa hình ảnh thật và giả, do đó nó cố gắng làm cho đầu ra của mình gần 1 cho hình ảnh thật và gần 0 cho hình ảnh giả.

- Quá trình huấn luyện:
 - + Chọn một batch ngẫu nhiên từ dữ liệu huấn luyện.
 - + Sử dụng GradientTape để tính gradient của generator và discriminator theo các hàm loss tương ứng.
 - + Sử dụng optimizer để cập nhật các trọng số của generator và discriminator dựa trên gradient tính được.
 - + Lặp lại các bước trên cho mỗi batch trong từng epoch.

Quá trình này giúp cải thiện generator sao cho nó tạo ra hình ảnh giả tốt hơn và cải thiện discriminator sao cho nó có khả năng phân biệt giữa hình ảnh thật và giả tốt hơn. Điều này dẫn đến việc cả hai mạng đều cải thiện qua các epoch.

Sau khi quá trình huấn luyện kết thúc, ta sẽ nhận được mô hình Generator để có thể sử dụng nhằm tạo ảnh.

2.4.2. Áp dụng CGAN vào tạo ảnh nghệ thuật

Trong bài báo cáo này, để áp dụng CGAN vào tạo ảnh nghệ thuật, chúng ta sẽ dùng một biến thể của CGAN là Pix2Pix hay còn được gọi là Image to Image Translation with Conditional Adversarial Network như trong paper được giới thiệu tại hội nghị thường niên về thị giác máy tính CVPR-2017.

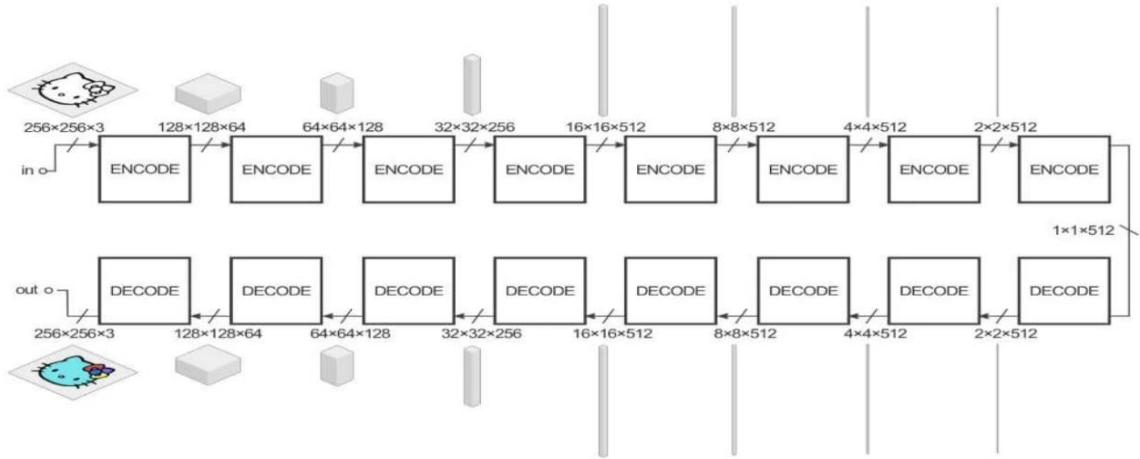
Kiến trúc của Pix2Pix cũng bao gồm 2 mô hình là generator và discriminator. Generator có tác dụng sinh ảnh giả để đánh lừa discriminator và discriminator có mục tiêu phân biệt giữa ảnh thật và ảnh giả.

Tương tự như khi huấn luyện DCGAN, bước đầu tiên khi huấn luyện CGAN pix2pix là chuẩn bị dữ liệu:

- Một bộ ảnh gốc sẽ được nhập vào, sau đó sẽ được chia thành 2 bộ ảnh là một bộ ảnh gốc (ảnh màu hoàn chỉnh) và một bộ ảnh phác họa tương ứng (ảnh phác họa được tạo từ ảnh gốc). Hai bộ ảnh này cũng sẽ được chuyển đổi về một kích thước cố định và chuẩn hóa giá trị về khoảng [-1, 1].

Sau đó là xây dựng mô hình CGAN pix2pix với 2 mạng là Generator và Discriminator:

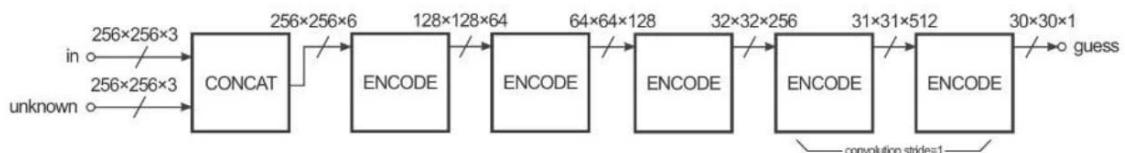
- Với Generator:
 - + Generator sử dụng một kiến trúc encoder-decoder với các lớp downsample và upsample. Các lớp downsample giúp trích xuất các đặc trưng quan trọng từ ảnh phác họa. Mỗi lớp downsample sử dụng Conv2D với kích thước kernel là 4 và hàm kích hoạt LeakyReLU, giúp tạo ra các feature map có độ sâu tăng dần. Các lớp upsample giúp tái tạo ảnh màu từ các đặc trưng đã trích xuất. Mỗi lớp upsample sử dụng Conv2DTranspose với kích thước kernel là 4 và hàm kích hoạt ReLU, kết hợp với BatchNormalization để duy trì mối quan hệ không gian giữa các pixel. Ngoài ra, các lớp Concatenate cũng sẽ được sử dụng để kết nối các đặc trưng từ encoder với decoder, giúp bảo toàn chi tiết ảnh.
 - + Quá trình ảnh đi qua mạng sẽ được biểu diễn như sau: mạng sẽ nhận đầu vào là ảnh phác họa có kích thước cố định. Ảnh phác họa đầu vào đi qua một chuỗi các lớp downsample, tạo ra các feature map. Ảnh phác họa đầu vào đi qua một chuỗi các lớp downsample, tạo ra các feature map. Cuối cùng, ảnh đi qua một lớp Conv2DTranspose với kernel là 4 và hàm kích hoạt tanh để tạo ra ảnh màu cuối cùng có kích thước 256x256.



Hình 2.6 Kiến trúc encoder-decoder của Generator trong CGAN [7]

- Với Discriminator:

- + Mạng này sẽ nhận vào một cặp ảnh (ảnh phác họa và ảnh màu tương ứng) và sử dụng các lớp Conv2D để phân biệt giữa ảnh thật và ảnh giả. Cuối cùng một dự đoán xác suất sẽ được đưa ra.
- + Quá trình sẽ được mô tả như sau: Đầu tiên, cặp ảnh được nối chồng lên nhau theo chiều sâu và đi qua một chuỗi các lớp downsample để giảm kích thước của ảnh và trích xuất đặc trưng. Mỗi lớp downsample sử dụng Conv2D với kích thước kernel là 4 và hàm kích hoạt LeakyReLU. Cuối cùng, feature map được đưa qua một lớp Conv2D để tính toán xem cặp ảnh là thật hay giả.



Hình 2.7 Mô tả kiến trúc của Discriminator trong CGAN [7]

Tiếp theo là định nghĩa hàm Loss và quá trình huấn luyện:

- Hàm Loss cho Generator và Discriminator:
 - + Generator Loss: Bao gồm binary crossentropy giữa đầu ra của Discriminator và nhãn "thật" và L1 loss giữa ảnh tạo ra và ảnh gốc.
 - + Discriminator Loss: Binary crossentropy giữa đầu ra của Discriminator cho ảnh thật và ảnh giả.
- Quá trình huấn luyện:
 - + Chọn một batch ngẫu nhiên từ dữ liệu huấn luyện.
 - + Sử dụng GradientTape để tính gradient của Generator và Discriminator theo các hàm loss tương ứng.
 - + Sử dụng optimizer để cập nhật các trọng số của Generator và Discriminator dựa trên gradient tính được.
 - + Lặp lại các bước trên cho mỗi batch trong từng epoch.

Tóm lại, quá trình áp dụng cGAN pix2pix để tạo ảnh nghệ thuật từ ảnh phác họa bao gồm các bước chuẩn bị dữ liệu, xây dựng mô hình Generator và Discriminator với kiến trúc Encoder-Decoder, định nghĩa hàm loss và quá trình huấn luyện để tối ưu hóa các trọng số của mô hình. So với DCGAN, cGAN pix2pix sử dụng thông tin điều kiện để tạo ra ảnh nghệ thuật có chất lượng cao và chi tiết hơn từ ảnh phác họa, nâng cao khả năng tạo ra các tác phẩm nghệ thuật phong phú và đa dạng.

CHƯƠNG 3. TIẾN HÀNH THỰC NGHIỆM

3.1. Dataset

3.1.1. *WikiArt*

Dataset WikiArt là một nguồn tài nguyên vô cùng quý giá và đa dạng về nghệ thuật hình ảnh, được thu thập từ nhiều thời kỳ và phong cách khác nhau trên toàn thế giới. Với hàng ngàn tác phẩm nghệ thuật và thông tin chi tiết về các nghệ sĩ, dataset này cung cấp một cơ sở dữ liệu lớn cho nghiên cứu, sáng tạo và phát triển ứng dụng trong lĩnh vực nghệ thuật và công nghệ.

Trong bối cảnh sự phát triển mạnh mẽ của trí tuệ nhân tạo và học máy, việc có một nguồn dữ liệu phong phú và đa dạng như Dataset WikiArt trở nên vô cùng quan trọng. Đây không chỉ là một nguồn cảm hứng lớn cho các nghệ sĩ và nhà thiết kế, mà còn là một công cụ hữu ích cho các nhà nghiên cứu trong việc phân tích xu hướng nghệ thuật, nghiên cứu về sự ảnh hưởng của các trường phái và nghệ sĩ, cũng như trong việc phát triển các ứng dụng mới trong lĩnh vực học máy và trí tuệ nhân tạo.

Dataset WikiArt được tổ chức một cách cẩn thận và có cấu trúc, với thông tin chi tiết về mỗi tác phẩm và nghệ sĩ tương ứng. Điều này giúp cho việc tìm kiếm, lọc và sử dụng dữ liệu trở nên thuận tiện hơn cho các nhà nghiên cứu và nhà phát triển. Bên cạnh đó, việc dataset này luôn được cập nhật với các tác phẩm mới và thông tin về các nghệ sĩ giúp cho việc nghiên cứu và sáng tạo luôn được tiếp tục và phát triển.

Dataset WikiArt bao gồm 80020 hình ảnh từ 1119 họa sĩ với 27 thể loại khác nhau trong 25GB dữ liệu. Đây là một kích thước khá lớn và do giới hạn về tài nguyên nên em chỉ lấy 1-3 thể loại để thực hiện training và mỗi lần training không quá 6000 ảnh với model DCGAN và không quá 1500 ảnh với model CGAN.



About Dataset

Dataset used for training <https://archive.org/details/wikiart-stylegan2-conditional-model> by Peter Baylies

Upscaled and resized, originally from <https://github.com/cs-chan/ArtGAN/tree/master/WikiArt%20Dataset>

Note:

1. The WikiArt dataset can be used only for non-commercial research purpose.
2. The images in the WikiArt dataset were obtained from WikiArt.org. The authors are neither responsible for the content nor the meaning of these images.
3. By using the WikiArt dataset, you agree to obey the terms and conditions of WikiArt.org.

Dataset contains 80020 unique images from 1119 different artists in 27 styles

Usability 8.75

License
CC0: Public Domain

Expected update frequency
Never

Tags

Image Computer Vision
Art

Hình 3.1 Bộ dữ liệu ảnh WikiArt

3.1.2. *Portrait paintings*

Bộ dữ liệu Portrait Paintings trên Kaggle là một kho dữ liệu đa dạng và phong phú về các bức tranh chân dung từ nhiều thời kỳ và nghệ sĩ trên khắp thế giới. Được biên soạn từ các nguồn đáng tin cậy, bộ dữ liệu này cung cấp một tài nguyên quý báu cho việc nghiên cứu, phân tích, và khám phá về nghệ thuật chân dung.

Bộ dữ liệu này bao gồm 5734 hình ảnh chất lượng cao với 250MB dữ liệu của các bức tranh chân dung, từ các nghệ sĩ nổi tiếng đến những tác phẩm không rõ nguồn gốc. Các bức tranh này được phân loại và gắn nhãn cẩn thận để dễ dàng tìm kiếm và sử dụng. Bộ dữ liệu bao gồm các trường thông tin như tên tác phẩm, tên nghệ sĩ, thời gian sáng tác, và nơi lưu trữ.

Bộ dữ liệu này được sử dụng để huấn luyện cho mô hình DCGAN – mô hình không thể kiểm soát được dữ liệu đầu ra. Việc sử dụng một bộ dữ liệu với các hình ảnh thuộc cùng một thể loại (tranh chân dung) sẽ giúp định hướng dữ liệu đầu ra tốt hơn.

The screenshot shows a Kaggle notebook interface. At the top, there's a navigation bar with a user icon, 'DEEKWAKAR CHAKRABORTY - UPDATED 3 YEARS AGO', a page number '16', a 'New Notebook' button, a 'Download (234 MB)' button, and a more options menu. Below the header, the notebook title is 'Portrait Paintings' with a subtitle 'Portraits scrapped from WikiArt website to train a GAN'. A preview image shows a grid of various portrait paintings. Below the title, there are links for 'Data Card', 'Code (4)', 'Discussion (0)', and 'Suggestions (0)'. On the right side, there are sections for 'Usability' (rating 7.50), 'License' (CC0: Public Domain), 'Expected update frequency' (Not specified), and 'Tags' (Business, Image, Intermediate, Art, TensorFlow, GAN).

Hình 3.2 Bộ dữ liệu ảnh *Portrait Paintings*

3.2. Chương trình ứng dụng

3.2.1. Huấn luyện mô hình

Quá trình huấn luyện bao gồm cả đánh giá và test này sẽ được thực hiện trên notebook của nền tảng Kaggle để tận dụng nguồn tài nguyên GPU giúp thực hiện quá trình huấn luyện diễn ra nhanh hơn. Như đã mô tả ở chương 2, sẽ có 2 mô hình được huấn luyện là mô hình DCGAN và mô hình CGAN pix2pix.

Trước hết, chúng ta sẽ nói chi tiết về quá trình huấn luyện mô hình đơn giản hơn là DCGAN.

Bước đầu tiên khi thực hiện huấn luyện là khai báo các thư viện:

```

import os
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm

from tensorflow.keras import layers
from tensorflow.keras import models
from keras.models import Sequential, Model, load_model
from keras.layers import Input, Dense, Reshape, Flatten, Activation, Concatenate
from keras.layers import ReLU, LeakyReLU, BatchNormalization, Dropout
from keras.layers import Conv2D, Conv2DTranspose, UpSampling2D, ZeroPadding2D
from keras.optimizers import Adam
from keras.initializers import RandomNormal

from IPython import display
from PIL import Image
import time

```

Hình 3.3 Khai báo thư viện - DCGAN

Tiếp theo là thiết lập các tham số và định nghĩa hàm loss:

```
# Preview image Frame
PREVIEW_ROWS = 4
PREVIEW_COLS = 7
PREVIEW_MARGIN = 4

# Image parameters
SEED_SIZE = 100
IMG_SIZE = 256
BATCH_SIZE = 64
EPOCHS = 150
LAMBDA = 100
SAVE_FREQ = 15

# define loss function
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

Hình 3.4 Thiết lập tham số và định nghĩa hàm loss – DCGAN

Tiếp theo là đọc ảnh, tiền xử lý ảnh và lưu ảnh vào tệp dưới dạng nhị phân:

```
training_data = []

# Look for saved file to save time loading and processing images between runs
print("Looking for saved binary file...")

if not os.path.isfile(os.path.join(input_dir, 'training_data.npy')):
    print("\n File not found, creating new file...\n")
    print('resizing...')

    for filename in os.listdir(data_dir):
        image_path = os.path.join(data_dir, filename)
        if os.path.isfile(image_path):
            image = Image.open(image_path).resize((IMG_SIZE, IMG_SIZE), Image.Resampling.LANCZOS)
            image_array = np.array(image)
            training_data.append(image_array)

    training_data = np.array(training_data, dtype=np.float32)
    training_data = (training_data - 127.5) / 127.5 # Normalize to [-1 , 1]

    print("Saving dataset binary file...")
    os.makedirs(input_dir, exist_ok=True)
    np.save(os.path.join(input_dir, 'training_data.npy'), training_data) # Save processed images as npy file
else:
    print("Data found, loading..")
    training_data = np.load(os.path.join(input_dir, 'training_data.npy'))

print("Dataset length: ", len(training_data))

Looking for saved binary file...
Data found, loading..
Dataset length: 10733
```

Hình 3.5 Đọc và tiền xử lý ảnh – DCGAN

Sau đó là viết các hàm xây dựng Generator, Discriminator, các hàm tính toán loss và gọi chúng trong các hàm dùng để huấn luyện:

- Hàm build_discriminator:

```
# Discriminator model
def build_discriminator():
    model = models.Sequential()
    model.add(layers.Conv2D(64, kernel_size=5, strides=2, input_shape=(64, 64, 3), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))

    model.add(layers.Conv2D(128, kernel_size=5, strides=2, padding='same'))
    model.add(layers.BatchNormalization(momentum=0.9))
    model.add(layers.LeakyReLU(alpha=0.2))

    model.add(layers.Conv2D(256, kernel_size=5, strides=2, padding='same'))
    model.add(layers.BatchNormalization(momentum=0.9))
    model.add(layers.LeakyReLU(alpha=0.2))

    model.add(layers.Conv2D(512, kernel_size=5, strides=2, padding='same'))
    model.add(layers.BatchNormalization(momentum=0.9))
    model.add(layers.LeakyReLU(alpha=0.2))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    model.add(layers.Activation('sigmoid'))
    return model
```

Hình 3.6 Xây dựng mạng Discriminator – DCGAN

- Hàm build_generator:

```
# Generator model
def build_generator():
    model = models.Sequential()
    model.add(layers.Dense(4*4*512, activation="relu", input_dim=SEED_SIZE)) #64x64 units
    model.add(layers.Reshape((4, 4, 512)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha = 0.2))

    model.add(layers.Conv2DTranspose(512, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha = 0.3))
    model.add(layers.Dropout(0.4))

    model.add(layers.Conv2DTranspose(256, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha = 0.2))
    model.add(layers.Dropout(0.4))

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha = 0.3))
    model.add(layers.Dropout(0.4))

    model.add(layers.Conv2DTranspose(32, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU(alpha = 0.2))

    model.add(layers.Conv2DTranspose(IMAGE_CHANNELS, (5, 5), strides=(1, 1), padding='same', use_bias=True, activation='tanh'))

    return model
```

Hình 3.7 Xây dựng mạng generator – DCGAN

- Hàm Generator loss và discriminator loss:

```
# Generator loss
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
# Discriminator loss
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

Hình 3.8 Hàm tính toán loss – DCGAN

- Hàm train_step: Hàm này thực hiện một bước huấn luyện cho mỗi batch dữ liệu. Nó sẽ tính toán gradient của hàm mất mát của generator và discriminator, sau đó áp dụng gradient này để cập nhật trọng số của hai mô hình.

```
# Train step
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, SEED_SIZE])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

        gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
        gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

        generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
        discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))

    return gen_loss, disc_loss
```

Hình 3.9 Xây dựng hàm huấn luyện trên mỗi batch – DCGAN

- Hàm train: thực hiện vòng lặp huấn luyện qua mỗi epoch, trong mỗi epoch sẽ thực hiện đối với mỗi batch trong tập dữ liệu, gọi hàm train_step để thực hiện một bước huấn luyện và thu thập các giá trị mất mát của generator và discriminator. Cuối cùng là lưu mô hình.

```

# Training loop
def train(dataset, epochs, checkpoint, checkpoint_prefix):
    for epoch in range(epochs):
        start = time.time()
        for image_batch in dataset:
            t = train_step(image_batch)
            gen_loss_list.append(t[0])
            disc_loss_list.append(t[1])

        g_loss = sum(gen_loss_list) / len(gen_loss_list)
        d_loss = sum(disc_loss_list) / len(disc_loss_list)

        if (epoch + 1) % SAVE_FREQ == 0:
            save_images(epoch + 1, generator)
            checkpoint.save(file_prefix=checkpoint_prefix)

        print('Epoch {}, Generator Loss: {:.4f}, Discriminator Loss: {:.4f}, Time: {:.2f} sec'.format(epoch + 1, g_loss, d_loss))

    # Save the model
if epoch == EPOCHS - 1:
    generator_file = '/kaggle/working/model/generator.keras'
    discriminator_file = '/kaggle/working/model/discriminator.keras'
    # Remove existing files if they exist
    try:
        os.remove(generator_file)
        os.remove(discriminator_file)
    except FileNotFoundError:
        pass
os.makedirs('/kaggle/working/model', exist_ok=True)
# Lưu mô hình generator và discriminator
generator.save(generator_file)
discriminator.save(discriminator_file)

```

Hình 3.10 Huấn luyện mô hình và lưu mô hình – DCGAN

Sau khi huấn luyện xong, chúng ta sẽ có model dcgan_generator.h5, và với model này chúng ta có thể sinh biến thể ảnh mới từ bộ ảnh và hiển thị lên màn hình với hàm display_generated_image:

```

def display_generated_image(generator, seed_size):
    # Create a random seed for generating new images
    random_seed = tf.random.normal([1, seed_size])

    # Generate an image using the generator
    new_image = generator(random_seed, training=False)

    # Convert the generated image to numpy array
    generated_image = new_image.numpy()

    # Rescale the pixel values to be between 0 and 255
    generated_image = 0.5 * generated_image + 0.5
    generated_image = (generated_image * 255).astype(np.uint8)

    # Plot the generated image
    plt.imshow(generated_image[0])
    plt.axis('off')
    plt.title('Generated Image')
    plt.show()

    return generated_image

```

Hình 3.11 Hàm sinh và hiển thị biến thể ảnh mới từ bộ ảnh từ model

Sau đây là một số hình ảnh được sinh ngẫu nhiên từ model:



Hình 3.12 Biến thể ảnh mới 1 được sinh ra từ dcgan_generator



Hình 3.13 Biến thể ảnh mới 2 được sinh ra từ dcgan_generator

Trên đây là quá trình huấn luyện của DCGAN. Tiếp đến em sẽ nói về quá trình huấn luyện CGAN pix2pix. Quá trình này khá tương tự với quá trình huấn luyện DCGAN, tuy nhiên sẽ có thay đổi ở bước đọc và tiền xử lý ảnh, hàm loss và kiến trúc model generator và discriminator.

Quá trình khai báo thư viện và thiết lập tham số sẽ được thực hiện tương tự như trong DCGAN, tiếp theo là các hàm xây dựng kiến trúc của Generator và Discriminator:

- Hàm upsample và hàm downsample:

```
# Downsample and Upsample blocks
def downsample(filters, size, apply_batchnorm=True):
    initializer = RandomNormal(0., 0.02)
    result = Sequential()
    result.add(Conv2D(filters, size, strides=2, padding='same', kernel_initializer=initializer, use_bias=False))
    if apply_batchnorm:
        result.add(BatchNormalization())
    result.add(LeakyReLU())
    return result
```

+ Code + Markdown

```
def upsample(filters, size, apply_dropout=False):
    initializer = RandomNormal(0., 0.02)
    result = Sequential()
    result.add(Conv2DTranspose(filters, size, strides=2, padding='same', kernel_initializer=initializer, use_bias=False))
    result.add(BatchNormalization())
    if apply_dropout:
        result.add(Dropout(0.5))
    result.add(ReLU())
    return result
```

Hình 3.14 Xây dựng hàm upsample và downsample – CGAN

- Hàm build_discriminator:

```
# Discriminator
def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    inp = Input(shape=[IMG_SIZE, IMG_SIZE, 3], name='input_image')
    tar = Input(shape=[IMG_SIZE, IMG_SIZE, 3], name='target_image')
    x = Concatenate()([inp, tar])

    down1 = downsample(64, 4, False)(x)
    down2 = downsample(128, 4)(down1)
    down3 = downsample(256, 4)(down2)
    zero_pad1 = ZeroPadding2D()(down3)
    conv = Conv2D(512, 4, strides=1, kernel_initializer=initializer, use_bias=False)(zero_pad1)
    batchnorm1 = BatchNormalization()(conv)
    leaky_relu = LeakyReLU()(batchnorm1)
    zero_pad2 = ZeroPadding2D()(leaky_relu)
    last = Conv2D(1, 4, strides=1, kernel_initializer=initializer)(zero_pad2)

    return Model(inputs=[inp, tar], outputs=last)
```

Hình 3.15 Xây dựng mạng Discriminator – CGAN

- Hàm build_generator:

```
# Generator
def Generator():
    inputs = Input(shape=[IMG_SIZE, IMG_SIZE, 3])
    down_stack = [
        downsample(64, 4, apply_batchnorm=False), # (bs, 128, 128, 64)
        downsample(128, 4), # (bs, 64, 64, 128)
        downsample(256, 4), # (bs, 32, 32, 256)
        downsample(512, 4), # (bs, 16, 16, 512)
        downsample(512, 4), # (bs, 8, 8, 512)
        downsample(512, 4), # (bs, 4, 4, 512)
        downsample(512, 4), # (bs, 2, 2, 512)
        downsample(512, 4), # (bs, 1, 1, 512)
    ]
    up_stack = [
        upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
        upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
        upsample(512, 4), # (bs, 16, 16, 1024)
        upsample(256, 4), # (bs, 32, 32, 512)
        upsample(128, 4), # (bs, 64, 64, 256)
        upsample(64, 4), # (bs, 128, 128, 128)
    ]
    initializer = RandomNormal(0., 0.02)
    last = Conv2DTranspose(3, 4, strides=2, padding='same', kernel_initializer=initializer, activation='tanh') # (bs, 256, 256, 3)
    x = inputs
    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)
    skips = reversed(skips[:-1])
    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = Concatenate()([x, skip])
    x = last(x)
    return Model(inputs=inputs, outputs=x)
```

Hình 3.16 Xây dựng mạng generator

- Hàm tính toán loss:

```
def generator_loss(disc_generated_output, gen_output, target):
    gan_loss = cross_entropy(tf.ones_like(disc_generated_output), disc_generated_output)
    l1_loss = tf.reduce_mean(tf.abs(target - gen_output))
    total_gen_loss = gan_loss + (LAMBDA * l1_loss)
    return total_gen_loss, gan_loss, l1_loss
```

```
def discriminator_loss(disc_real_output, disc_generated_output):
    real_loss = cross_entropy(tf.ones_like(disc_real_output), disc_real_output)
    generated_loss = cross_entropy(tf.zeros_like(disc_generated_output), disc_generated_output)
    total_disc_loss = real_loss + generated_loss
    return total_disc_loss
```

Hình 3.17 Hàm tính toán loss – CGAN

- Hàm train_step: Hàm này thực hiện một bước huấn luyện cho mỗi batch dữ liệu. Nó tính toán loss, gradient và cập nhật trọng số cho Generator và Discriminator. Nó tính toán đầu ra của hai mạng, sau đó tính toán loss cho mỗi mạng và gradient của nó. Cuối cùng, gradient được áp dụng để cập nhật các tham số của mỗi mạng. Điều này giúp cải thiện chất lượng ảnh được tạo ra và khả năng phân biệt giữa ảnh thật và ảnh giả qua các epoch của quá trình huấn luyện.

```

def train_step(input_image, target, epoch):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        # Generator output
        gen_output = generator(input_image, training=True)

        # Discriminator output
        disc_real_output = discriminator([input_image, target], training=True)
        disc_generated_output = discriminator([input_image, gen_output], training=True)

        # Generator loss
        gen_total_loss, gen_gan_loss, gen_l1_loss = generator_loss(disc_generated_output, gen_output, target)

        # Discriminator loss
        disc_loss = discriminator_loss(disc_real_output, disc_generated_output)

    # Calculate gradients
    generator_gradients = gen_tape.gradient(gen_total_loss, generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    # Apply gradients
    generator_optimizer.apply_gradients(zip(generator_gradients, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(discriminator_gradients, discriminator.trainable_variables))

    return gen_total_loss, gen_gan_loss, gen_l1_loss, disc_loss

```

Hình 3.18 Xây dựng hàm huấn luyện trên mỗi batch

- Hàm fit: thực hiện vòng lặp huấn luyện qua mỗi epoch, trong mỗi epoch sẽ thực hiện đối với mỗi batch trong tập dữ liệu, gọi hàm train_step để thực hiện một bước huấn luyện và thu thập các giá trị măt măt của generator và discriminator. Cuối cùng là lưu mô hình.

```

# Training loop
def fit(train_ds, epochs):
    for epoch in range(epochs):
        start = time.time()
        for input_image, target in train_ds:
            t = train_step(input_image, target, epoch)

            gen_total_loss_list.append(t[0])
            gen_gan_loss_list.append(t[1])
            gen_l1_loss_list.append(t[2])
            disc_loss_list.append(t[3])

        gen_total_loss_epoch = sum(gen_total_loss_list) / len(gen_total_loss_list)
        gen_gan_loss_epoch = sum(gen_gan_loss_list) / len(gen_gan_loss_list)
        gen_l1_loss_epoch = sum(gen_l1_loss_list) / len(gen_l1_loss_list)
        disc_loss_epoch = sum(disc_loss_list) / len(disc_loss_list)

        # Print losses for each epoch
        print(f'Epoch {epoch + 1}, '
              f'Gen Total Loss: {gen_total_loss_epoch}, '
              f'Gen GAN Loss: {gen_gan_loss_epoch}, '
              f'Gen L1 Loss: {gen_l1_loss_epoch}, '
              f'Disc Loss: {disc_loss_epoch}')

        # Save generated images
        # for example_input, example_target in train_ds.take(1):
        #     generate_images(generator, example_input, example_target)

        print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                          time.time()-start))

    # Save checkpoint and images every save_every_n_epochs epochs
    if (epoch + 1) % SAVE_FREQ == 0:
        checkpoint.save(file_prefix=checkpoint_prefix)
        save_images(generator, example_sketch, example_color, epoch + 1, output_dir)

```

Hình 3.19 Hàm huấn luyện mô hình – CGAN

Sau quá trình huấn luyện mô hình, cuối cùng sẽ thu được mô hình cgan_generator.h5. Với mô hình này, chúng ta có thể tạo ảnh chi tiết từ ảnh phác họa với hàm load_show_and_save_image:

```
def load_show_and_save_images(generator, test_folder, save_folder):
    if not os.path.exists(save_folder):
        os.makedirs(save_folder)

    for image_name in os.listdir(test_folder):
        image_path = os.path.join(test_folder, image_name)

        # Đọc và tiền xử lý ảnh đầu vào
        original_image = preprocess_test_image(image_path)

        # Sinh ảnh từ model generator
        generated_image = generate_image(generator, original_image)

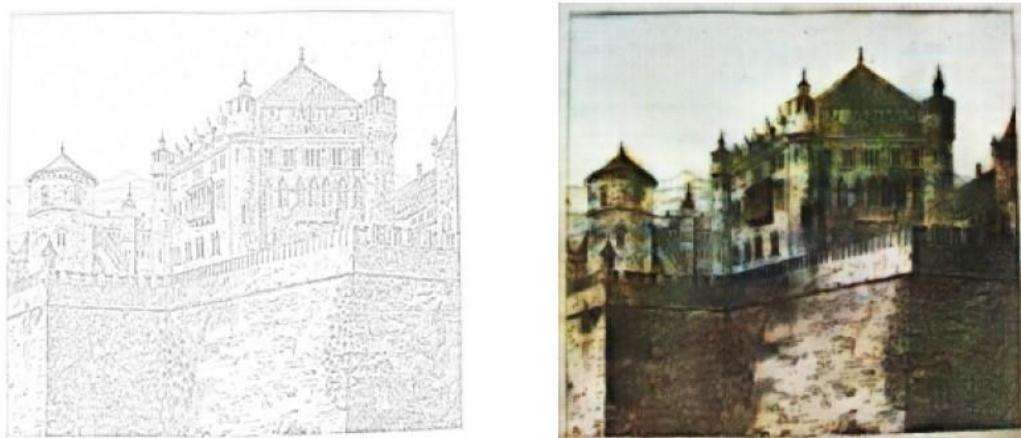
        # Hiển thị và lưu ảnh
        save_path = os.path.join(save_folder, 'generated_' + image_name)
        show_and_save_images(original_image, generated_image, save_path)
```

Hình 3.20 Hàm tạo ảnh chi tiết từ ảnh phác họa tương ứng

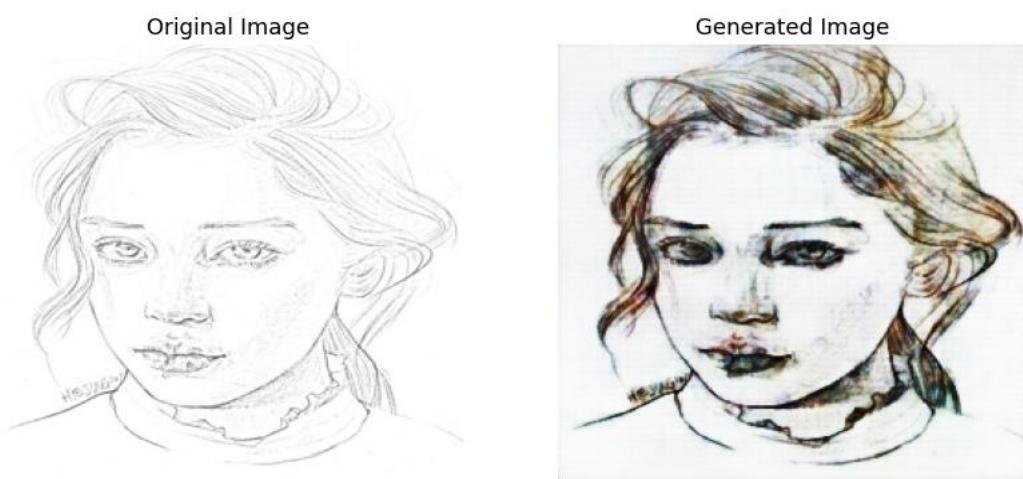
Sau đây là một số ảnh chi tiết được tạo từ ảnh phác họa tương ứng:



Hình 3.21 Ảnh phác họa và chi tiết được tạo 1



Hình 3.22 Ảnh phác họa và ảnh chi tiết được tạo 2



Hình 3.23 Ảnh phác họa và ảnh chi tiết được tạo 3

3.2.2. Đánh giá

Ở phần huấn luyện mô hình, em đã thực hiện huấn luyện 2 model là Deep Convolutional GAN (DCGAN) và Conditional GAN (CGAN) – cụ thể là pix2pix. Sau khi có được model, em đã viết hàm để sinh một số hình ảnh từ generator, tuy nhiên chỉ nhìn vào bằng mắt thường rất khó để đánh giá trực quan xem chất lượng hình ảnh đã tốt chưa (model đã tốt chưa). Việc thực hiện đánh giá này thường được thực hiện dựa trên 2 yếu tố chính là:

- Chất lượng hình ảnh: Các hình ảnh cần rõ ràng, có chất lượng cao giống với dataset.
- Độ đa dạng của hình ảnh: với model DCGAN, Generator cần phải sinh ra nhiều ảnh khác nhau khi một vector noise z ngẫu nhiên được

đưa vào. Với model CGAN pix2pix, Generator cần phải sinh ra nhiều ảnh khác nhau với cùng một điều kiện đầu vào.

Có hai phương pháp phổ biến được sử dụng để đánh giá chất lượng hình ảnh do GAN tạo ra: Inception Score (IS) và Fréchet Inception Distance (FID).

Inception Score (IS) là một trong những thước đo đầu tiên được sử dụng để đánh giá chất lượng hình ảnh được tạo ra bởi GAN. Inception Score sử dụng mạng Inception v3 được huấn luyện trên tập dữ liệu ImageNet để tính toán.

Inception Score đo lường hai yếu tố chính:

- Độ đa dạng của hình ảnh: Các hình ảnh được tạo ra bởi GAN nên đa dạng và không trùng lặp.
- Chất lượng hình ảnh: Các hình ảnh nên có ý nghĩa rõ ràng và thuộc về các lớp đối tượng xác định.

Công thức tính IS:

$$IS(G) = \exp(E_{\mathbf{x}} \sim pg D_{KL}(p(y | \mathbf{x}) \| p(y)))$$

Trong đó,

- G là mô hình GAN.
- pg là phân phối của dữ liệu được tạo ra bởi mô hình G .
- \mathbf{x} là một hình ảnh được tạo ra bởi G .
- $p(y | \mathbf{x})$ là phân phối của các nhãn dự đoán bởi mô hình Inception trên hình ảnh \mathbf{x} .
- $p(y)$ là phân phối tổng quát của các nhãn.
- $D_{KL}(p(y | \mathbf{x}) \| p(y))$ là KL divergence giữa phân phối của các nhãn dự đoán cho mỗi hình ảnh \mathbf{x} và phân phối tổng quát của các nhãn.

KL divergence là một khái niệm trong lý thuyết thông tin để đo lường sự khác biệt giữa hai phân phối xác suất. Nó được sử dụng để đo lường mức độ mà một phân phối xác suất p khác biệt so với một phân phối xác suất q.

Công thức tính KL divergence giữa hai phân phối xác suất $p(x)$ và $q(x)$ là:

$$D_{KL}(p \parallel q) = \sum_x p(x) \log\left(\frac{q(x)}{p(x)}\right)$$

Nói một cách đơn giản, KL divergence tính toán tỷ lệ trung bình của sự khác biệt giữa các xác suất của hai phân phối cho mỗi sự kiện. Nếu KL divergence gần bằng 0, nghĩa là hai phân phối xác suất gần nhau, và nếu KL divergence lớn, nghĩa là hai phân phối khác biệt nhiều.

Để tính IS, chúng ta cần tính toán trung bình của KL divergence giữa phân phối của các nhãn dự đoán cho mỗi hình ảnh và phân phối tổng quát của các nhãn trên tập dữ liệu. Sau đó, áp dụng hàm mũ để đánh giá sự khác biệt giữa phân phối của mỗi hình ảnh và phân phối tổng quát.

IS đo lường sự khác biệt giữa phân phối của các nhãn dự đoán cho mỗi hình ảnh và phân phối tổng quát của các nhãn. Một IS cao cho thấy các hình ảnh có đa dạng và chất lượng cao.

Ưu điểm:

- Dễ tính toán và không yêu cầu hình ảnh thực để so sánh.

Nhược điểm:

- Không phản ánh đầy đủ chất lượng hình ảnh, có thể bị ảnh hưởng bởi các mẫu không phân phối đồng đều.

Fréchet Inception Distance (FID) là một thước đo đánh giá khác, khắc phục một số hạn chế của Inception Score. FID đo khoảng cách giữa phân phối của các đặc trưng hình ảnh thực và hình ảnh do GAN tạo ra. Để tính

FID, chúng ta sử dụng mạng Inception v3 để trích xuất các đặc trưng từ một lớp trung gian của mạng.

FID được tính toán dựa trên Fréchet Distance giữa hai phân phối Gaussian:

$$\text{FID} = \| \mu_r - \mu_g \|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

Trong đó,

- (μ_r, Σ_r) là trung bình và ma trận hiệp phương sai của các đặc trưng của hình ảnh thực.
- (μ_g, Σ_g) là trung bình và ma trận hiệp phương sai của các đặc trưng của hình ảnh do GAN tạo ra.

FID tính toán hai thành phần:

- Bình phương của khoảng cách giữa trung bình của hai phân phối.
- Sự tương tự giữa hai ma trận hiệp phương sai.

Cả hai thành phần này đều phản ánh sự tương đồng giữa hai phân phối Gaussian, thể hiện bằng khoảng cách Fréchet.

FID được coi là tốt khi nó thấp, tức là khi các hình ảnh được tạo ra gần giống với hình ảnh thực và có sự đa dạng và chất lượng cao.

Áp dụng Inception Score (IS) và Fréchet Inception Distance (FID) vào đánh giá mô hình DCGAN:

- Đầu tiên là khai báo các thư viện cần dùng:

```
import os
import numpy as np
import tensorflow as tf
import cv2
from scipy.linalg import sqrtm
# from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from PIL import Image
import keras
from keras.applications.inception_v3 import InceptionV3, preprocess_input
```

Hình 3.24 Khai báo thư viện – Đánh giá mô hình DCGAN

- Xây dựng hàm load_images_from_directory: Hàm này có tác dụng đọc ảnh từ địa chỉ đưa vào. Nó duyệt qua tất cả các file ảnh trong thư mục và chuyển đổi chúng thành dạng RGB, sau đó chuẩn hóa giá trị pixel về khoảng từ 0 đến 1. Trong quá trình đánh giá GAN, ảnh này được sử dụng làm ảnh thật để so sánh với ảnh giả được tạo ra bởi mô hình.

```
def load_images_from_directory(image_dir):
    images = []
    count = 0
    for filename in os.listdir(image_dir):
        if count >= 150: # Số lượng ảnh tối đa
            break
        img_path = os.path.join(image_dir, filename)
        if os.path.isfile(img_path):
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Chuyển đổi từ BGR sang RGB
            img = img / 255.0 # Chuẩn hóa giá trị ảnh từ 0-255 về 0-1
            images.append(img)
            count += 1
    return images
```

Hình 3.25 Đọc và tiền xử lý ảnh – Đánh giá mô hình DCGAN

- Xây dựng hàm generate_fake_samples: Hàm này sinh ra các ảnh giả bằng cách đưa vào generator một nguồn ngẫu nhiên (random_seed) và tạo ra ảnh từ nguồn ngẫu nhiên đó. Nó được sử dụng để tạo ra một lượng ảnh giả có cùng phân phối với dữ liệu thực.

```
def generate_fake_samples(generator, seed_size, n_samples):
    generated_images = []
    for _ in range(n_samples):
        random_seed = tf.random.normal([1, seed_size])
        new_image = generator(random_seed, training=False)
        generated_images.append(new_image[0].numpy())
    return generated_images
```

Hình 3.26 Tạo dữ liệu ảnh giả bằng generator – Đánh giá mô hình DCGAN

- Xây dựng hàm preprocess_images: Hàm này thực hiện các bước tiền xử lý cho ảnh trước khi chúng được đưa vào mô hình InceptionV3 để tính toán Inception Score và FID. Cụ thể, nó thay đổi kích thước

ảnh thành 299x299 pixel và chuẩn hóa giá trị pixel về khoảng [-1, 1].

```
def preprocess_images(images):
    images_resized = np.array([np.array(Image
        .fromarray((img * 255)
        .astype(np.uint8))
        .resize((299, 299))) for img in images])
    images_preprocessed = preprocess_input(images_resized)
    return images_preprocessed
```

Hình 3.27 Hàm thay đổi kích thước ảnh – Đánh giá mô hình DCGAN

- Xây dựng hàm inception_score: Hàm này tính toán Inception Score của các ảnh dựa trên mô hình InceptionV3 được huấn luyện trước trên ImageNet. Inception Score đo lường độ chất lượng và đa dạng của các ảnh bằng cách sử dụng xác suất phân loại từ mạng Inception.

```
def inception_score(images, batch_size=32):
    inception_model = InceptionV3(include_top=True, weights='imagenet')
    processed_images = preprocess_images(images)
    preds = inception_model.predict(processed_images, batch_size=batch_size)
    preds = np.exp(preds) / np.sum(np.exp(preds), axis=1, keepdims=True)
    scores = np.sum(preds * np.log(preds), axis=1)
    scores = np.exp(np.mean(scores))
    return scores
```

Hình 3.28 Hàm tính Inception Score – Đánh giá mô hình DCGAN

- Xây dựng hàm calculate_fid: Hàm này tính toán Fréchet Inception Distance (FID) giữa phân phối của các đặc trưng từ ảnh thật và ảnh giả, dựa trên mạng InceptionV3. FID đo lường sự khác biệt giữa hai phân phối bằng cách tính khoảng cách giữa trung bình và ma trận hiệp phương sai của chúng.

```

def calculate_fid(real_images, generated_images):
    inception_model = InceptionV3(include_top=False, pooling='avg', input_shape=(299, 299, 3))
    real_images = preprocess_images(real_images)
    generated_images = preprocess_images(generated_images)
    real_activations = inception_model.predict(real_images)
    generated_activations = inception_model.predict(generated_images)
    mu_real, sigma_real = np.mean(real_activations, axis=0), np.cov(real_activations, rowvar=False)
    mu_gen, sigma_gen = np.mean(generated_activations, axis=0), np.cov(generated_activations, rowvar=False)
    diff_mean = np.sum((mu_real - mu_gen)**2)
    cov_mean = sqrtm(sigma_real.dot(sigma_gen))
    if np.iscomplexobj(cov_mean):
        cov_mean = cov_mean.real
    fid = diff_mean + np.trace(sigma_real + sigma_gen - 2*cov_mean)
    return fid

```

Hình 3.29 Hàm tính Fréchet Inception Distance) – Đánh giá DCGAN

- Xây dựng hàm evaluate_GAN: Hàm này đánh giá mô hình GAN bằng cách tính toán Inception Score và Fréchet Inception Distance (FID). Nó sinh ra các ảnh giả, sau đó tính toán Inception Score và FID giữa các ảnh thật và ảnh giả. Inception Score đo sự đa dạng và chất lượng của ảnh giả, trong khi FID đo sự tương đồng giữa phân phối của các đặc trưng từ ảnh thật và ảnh giả.

```

def evaluate_GAN(generator, seed_size, n_samples, real_images):
    fake_samples = generate_fake_samples(generator, seed_size, n_samples)
    score = inception_score(fake_samples)
    fid = calculate_fid(real_images, fake_samples)
    return score, fid

```

Hình 3.30 Hàm đánh giá mô hình DCGAN

Kết quả:

```

5/5 [=====] - 25s 4s/step
5/5 [=====] - 35s 5s/step
5/5 [=====] - 40s 7s/step
Inception Score: 0.0010000323
Fréchet Inception Distance: 349.9564547103416

```

Hình 3.31 Kết quả đánh giá mô hình DCGAN

Dựa trên kết quả đánh giá của mô hình DCGAN, có thể thấy điểm Inception Score rất thấp (0.0010001063) và Fréchet Inception Distance có

giá trị là âm (-9.635637705439616e+86), cho thấy mô hình không hoạt động hiệu quả:

- Inception Score thấp cho thấy sự đa dạng và chất lượng của ảnh được tạo ra không cao.
- Giá trị cao của Fréchet Inception Distance cũng chỉ ra rằng khoảng cách giữa phân phối của ảnh thật và ảnh được tạo ra bởi mô hình là khá lớn.

Vì vậy, kết quả này cho thấy mô hình DCGAN không tạo ra các hình ảnh phù hợp và có thể cần phải được cải thiện để tăng cường chất lượng và đa dạng của các hình ảnh được tạo ra.

Áp dụng Fréchet Inception Distance (FID) vào đánh giá mô hình CGAN pix2pix:

- Đầu tiên là khai báo thư viện cần dùng tương tự như trong phần đánh giá DCGAN.
- Xây dựng hàm load_images_from_folder: Hàm này tải ảnh từ thư mục **folder_path**. Nó duyệt qua tất cả các file ảnh trong thư mục, tiền xử lý chúng bằng cách sử dụng **preprocess_test_image**, và trả về một danh sách các ảnh đã được tiền xử lý. Trong quá trình đánh giá, hàm này được sử dụng để tải các ảnh thật từ thư mục để tính FID.

```
def load_images_from_folder(folder_path):
    images = []
    count = 0
    for filename in os.listdir(folder_path):
        if count >= 100:
            break
        img_path = os.path.join(folder_path, filename)
        if os.path.isfile(img_path):
            image = preprocess_test_image(img_path)
            images.append(image)
            count += 1
    return images
```

Hình 3.32 Hàm đọc ảnh và tiền xử lý ảnh – CGAN

- Xây dựng hàm **generate_image**: Hàm này tạo ra ảnh giả từ một ảnh đầu vào bằng cách sử dụng generator của CGAN. Ảnh đầu vào được thay đổi kích thước về (256, 256), sau đó được mở rộng thêm một chiều để tạo thành batch và đưa vào generator để sinh ra ảnh giả. Trong quá trình đánh giá, hàm này được sử dụng để tạo ra các ảnh giả từ các ảnh thật để tính FID.

```
def generate_image(generator, input_image):
    # Thay đổi kích thước của ảnh đầu vào
    input_image_resized = tf.image.resize(input_image, (256, 256))
    input_image_resized = tf.expand_dims(input_image_resized, 0) # Thêm chiều batch
    generated_image = generator(input_image_resized, training=False)[0]
    return generated_image
```

Hình 3.33 Hàm sinh ảnh giả bằng Generator – CGAN

- Xây dựng hàm **calculate_fid_pix2pix**: Hàm này tính toán Fréchet Inception Distance (FID) giữa hai phân phối đặc trưng từ ảnh thật và ảnh giả. Nó tính trung bình và hiệp phương sai của các đặc trưng, sau đó tính toán khoảng cách FID dựa trên trung bình và hiệp phương sai đó. FID đo lường sự khác biệt giữa các phân phối đặc trưng của ảnh thật và ảnh giả.

```
def calculate_fid_pix2pix(real_activations, fake_activations):
    # Calculate mean and covariance statistics
    mu_real, sigma_real = np.mean(real_activations, axis=0), np.cov(real_activations, rowvar=False)
    mu_fake, sigma_fake = np.mean(fake_activations, axis=0), np.cov(fake_activations, rowvar=False)

    # Calculate squared difference between means
    diff_mean = np.sum((mu_real - mu_fake) ** 2)

    # Calculate squared root of matrix product of covariances
    cov_mean = sqrtm(sigma_real.dot(sigma_fake))
    if np.iscomplexobj(cov_mean):
        cov_mean = cov_mean.real

    # Calculate FID
    fid = diff_mean + np.trace(sigma_real + sigma_fake - 2 * cov_mean)

    return fid
```

Hình 3.34 Hàm tính Fréchet Inception Distance – CGAN

- Xây dựng hàm **preprocess_evaluate_image**: Hàm này được sử dụng để tiền xử lý ảnh từ đường dẫn image_path. Nó đảm bảo rằng ảnh có 3 kênh (RGB), thay đổi kích thước ảnh về IMG_SIZE x IMG_SIZE, và chuẩn hóa giá trị pixel về khoảng [-1, 1]. Trong quá

trình đánh giá CGAN, hàm này được sử dụng để tiền xử lý các ảnh thật và ảnh giả trước khi tính toán FID.

```
def preprocess_test_image(image_path):
    image = Image.open(image_path).convert('RGB') # Đảm bảo ảnh có 3 kênh
    image = image.resize((IMG_SIZE, IMG_SIZE))
    image = np.array(image).astype(np.float32)
    image = (image / 127.5) - 1 # Chuẩn hóa ảnh về khoảng [-1, 1]
    return image
```

Hình 3.35 Hàm tiền xử lý – CGAN

- Xây dựng hàm **evaluate_generator_FID**: Hàm này đánh giá mô hình CGAN bằng cách tính toán FID giữa ảnh thật và ảnh giả. Nó tính toán kích hoạt cho ảnh thật và ảnh giả từ mô hình InceptionV3, sau đó sử dụng hàm calculate_fid_pix2pix để tính FID giữa chúng.

```
def evaluate_generator_FID(generator, test_images):
    # Load InceptionV3 model pretrained on ImageNet
    inception_model = InceptionV3(include_top=False, pooling='avg', input_shape=[299, 299, 3])

    real_activations = []
    fake_activations = []

    # Tính toán kích hoạt cho ảnh thật
    for image in test_images:
        original_image_resized = tf.image.resize(image, (299, 299))
        expanded_image = np.expand_dims(original_image_resized, axis=0).copy() # Tạo bản sao trước khi thay đổi
        real_activations.append(inception_model.predict(preprocess_input(expanded_image)))

    real_activations = np.concatenate(real_activations, axis=0)

    # Tính toán kích hoạt cho ảnh giả
    for image in test_images:
        generated_image = generate_image(generator, image)
        expanded_image = np.expand_dims(generated_image, axis=0).copy() # Tạo bản sao trước khi thay đổi
        fake_activations.append(inception_model.predict(preprocess_input(expanded_image)))

    fake_activations = np.concatenate(fake_activations, axis=0)

    # Calculate FID
    fid = calculate_fid_pix2pix(real_activations, fake_activations)

    return fid
```

Hình 3.36 Hàm đánh giá mô hình CGAN

Kết quả:

```
print("Fréchet Inception Distance:", fid)
```

Fréchet Inception Distance: 117.85714285714286

Hình 3.37 Kết quả đánh giá mô hình

Theo lý thuyết, FID không âm và FID càng thấp thì 2 distribution càng gần nhau => ảnh sinh ra càng giống ảnh gốc trong dataset. Và với kết quả này thì FID không quá nhỏ cũng không quá lớn. Ta thu được một kết quả ở mức trung bình.

CHƯƠNG 4. TÍCH HỢP HỆ THỐNG

4.1. Giới thiệu về framework sử dụng

Python Flask là một micro-framework được viết bằng ngôn ngữ lập trình Python dùng cho các nhà phát triển web. Nó được phát triển bởi Armin Ronacher, người dẫn đầu một nhóm những người đam mê Python quốc tế có tên là Poco. Flask dựa trên bộ công cụ Werkzeug WSGI và template engine Jinja2. Cả hai đều là các dự án của Poco. Micro ở đây không có nghĩa là framework này thiếu các chức năng mà thể hiện ở việc nó sẽ cung cấp những chức năng “cốt lõi” nhất cho các ứng dụng web và có khả năng mở rộng, người dùng cũng có thể mở rộng bất cứ lúc nào vì Flask hỗ trợ rất nhiều các tiện ích mở rộng như tích hợp CSDL, hệ thống upload, xác thực, template, email... Việc là một micro-framework cũng giúp cho flask có một môi trường xử lý độc lập và ít phải sử dụng các thư viện bên ngoài, điều này giúp nó nhẹ và ít gặp phải các lỗi hơn, việc phát hiện và xử lý các lỗi cũng dễ dàng và đơn giản hơn.

[11]

Flask cung cấp các tính năng cơ bản như định tuyến URL, tạo giao diện người dùng, xử lý yêu cầu và phản hồi HTTP. Ngoài ra, nó cũng hỗ trợ các tính năng mở rộng thông qua các thư viện mở rộng Flask (Flask extensions) để thêm các chức năng như xác thực người dùng, kết nối cơ sở dữ liệu, quản lý phiên và nhiều tính năng khác.

Ưu điểm của Flask framework:

- Dễ học và sử dụng: Flask có cú pháp đơn giản, dễ hiểu và tài liệu rất dễ tiếp cận. Điều này làm cho việc học và sử dụng Flask trở nên thuận lợi đối với cả người mới học lập trình và lập trình viên có kinh nghiệm.
- Linh hoạt: Flask không áp đặt cấu trúc dự án cụ thể nào, cho phép người phát triển tự do tổ chức dự án theo cách mà họ muốn. Tiện ích

này làm cho Flask linh hoạt và có thể được áp dụng vào nhiều loại ứng dụng web khác nhau.

- Cộng đồng phát triển mạnh mẽ: Flask có một cộng đồng phát triển rộng lớn với nhiều người dùng đóng góp và chia sẻ kiến thức trên các diễn đàn, blog và trang tài liệu.
- Extension ecosystem: Flask cung cấp một hệ sinh thái mở rộng (extension ecosystem) phong phú cho phép người dùng mở rộng chức năng của Flask thông qua các extension như Flask - SQLAlchemy cho việc kết nối cơ sở dữ liệu, Flask - WTF cho việc xử lý mẫu, nhiều extension khác.
- Tích hợp dễ dàng: Flask có khả năng tích hợp với các công nghệ và thư viện khác trong cộng đồng Python như SQLAlchemy cho kết nối cơ sở dữ liệu, Jinja2 cho tạo mẫu và WTForms cho xử lý form.
- Lightweight: Flask là một framework siêu nhẹ, không yêu cầu nhiều tài nguyên và có thể được triển khai một cách dễ dàng.

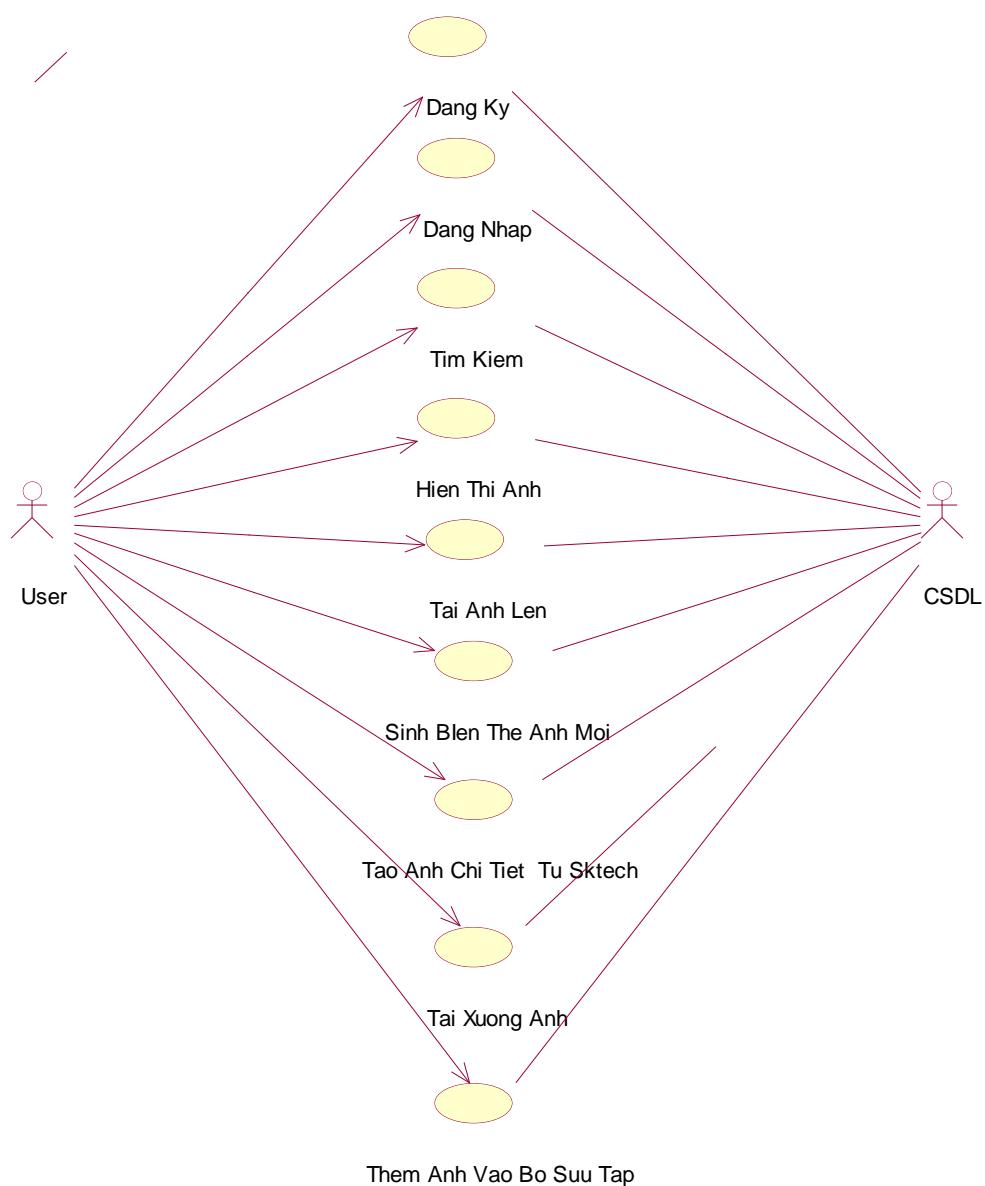
Nhược điểm của Flask framework:

- Thiếu cấu trúc: Do Flask không ép buộc một cấu trúc dự án cụ thể nên đã khiến nhiều dự án rơi vào tình trạng mất cấu trúc, đặc biệt khi dự án phát triển lớn và phức tạp hơn.
- Tính năng giới hạn: So với các framework web khác, Flask có ít tính năng và đặc điểm tích hợp sẵn. Điều này có nghĩa là người phát triển cần phụ thuộc vào các extension từ bên ngoài để thực hiện các chức năng cụ thể.
- Đứng sau so với các framework toàn diện: Đối với các ứng dụng lớn và phức tạp, Flask không thể cung cấp đầy đủ các tính năng và công cụ cần thiết khi so sánh với các framework toàn diện như Django.
- Phải tự quản lý những vấn đề như bảo mật: Do tính đơn giản mà Flask không thể cung cấp sẵn các tính năng bảo mật nâng cao. Nên

tảng còn đòi hỏi người phát triển phải tự quản lý và triển khai các biện pháp bảo mật cho ứng dụng.

4.2. Phân tích thiết kế hệ thống

4.2.1. Biểu đồ use case



Hình 4.1 Biểu đồ use case tổng quát



Hình 4.2 Biểu đồ mối quan hệ giữa các use case

4.2.2. Mô tả chi tiết các use case

4.2.2.1 Mô tả use case Đăng nhập

1: Tên Usecase: Đăng nhập

2: Mô tả văn tắt: Người dùng sử dụng tên người dùng và mật khẩu để đăng nhập vào hệ thống.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng nhập tên người dùng và mật khẩu.

2) Hệ thống kiểm tra thông tin đăng nhập.

3) Nếu thông tin đúng, hệ thống đăng nhập người dùng và chuyển hướng đến trang chính.

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

1) Nếu thông tin đăng nhập không đúng: Hệ thống hiển thị thông báo lỗi "Invalid username or password".

Usecase kết thúc.

4: Yêu cầu đặc biệt:

Không có.

5: Tiền điều kiện:

Không có

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.2 Mô tả use case Đăng ký

1: Tên Usecase: Đăng ký

2: Mô tả văn tắt: Người dùng tạo một tài khoản mới bằng cách cung cấp thông tin tên người dùng, email và mật khẩu.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng nhập thông tin đăng ký: tên người dùng, email và mật khẩu.

2) Hệ thống kiểm tra tính hợp lệ của thông tin và tạo tài khoản mới nếu đủ điều kiện.

3) Hệ thống thông báo kết quả đăng ký thành công.

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

1) Nếu tên người dùng đã tồn tại: Hệ thống hiển thị thông báo lỗi "Username already exists".

Usecase kết thúc.

4: Yêu cầu đặc biệt:

Không có.

5: Tiền điều kiện:

Không có

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.3 Mô tả use case Tìm kiếm

1: Tên Usecase: Tìm kiếm

2: Mô tả vắn tắt: Người dùng tìm kiếm ảnh hoặc bộ sưu tập bằng cách nhập từ khóa vào ô tìm kiếm.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng nhập từ khóa tìm kiếm vào ô tìm kiếm.

2) Hệ thống tìm kiếm các ảnh hoặc bộ sưu tập liên quan.

3) Hệ thống hiển thị kết quả tìm kiếm.

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

Không có

4: Yêu cầu đặc biệt:

Không có.

5: Tiền điều kiện:

Không có

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.4 Mô tả use case Xem ảnh

1: Tên Usecase: Xem ảnh

2: Mô tả văn tắt: Người dùng xem chi tiết về một ảnh cụ thể.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng nhập vào một ảnh từ kết quả tìm kiếm hoặc từ bộ sưu tập.

2) Hệ thống hiển thị chi tiết về ảnh, bao gồm ảnh gốc, ảnh được sinh ra, tiêu đề và bộ sưu tập nếu có.

3) Người dùng có thể tải ảnh lên hoặc thêm ảnh vào bộ sưu tập (nếu đăng nhập).

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

Không có

4: Yêu cầu đặc biệt:

Không có.

5: Tiền điều kiện:

Người dùng đã xem trang kết quả tìm kiếm hoặc bộ sưu tập và chọn một ảnh cụ thể.

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.5 Mô tả use case Thêm ảnh vào bộ sưu tập

1: Tên Usecase: Thêm ảnh vào bộ sưu tập

2: Mô tả văn tắt: Người dùng thêm một ảnh vào một bộ sưu tập của họ.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng xem chi tiết về một ảnh.

2) Người dùng nhấn vào nút "Thêm vào bộ sưu tập".

3) Hệ thống hiển thị danh sách bộ sưu tập của người dùng.

4) Người dùng chọn một bộ sưu tập hoặc tạo mới nếu cần.

5) Hệ thống thêm ảnh vào bộ sưu tập được chọn.

Use case kết thúc.

3.2: Luồng rẽ nhánh:

1) Nếu người dùng chưa đăng nhập: Hệ thống yêu cầu người dùng đăng nhập hoặc đăng ký.

Usecase kết thúc.

4: Yêu cầu đặc biệt:

Người dùng cần đăng nhập để thêm ảnh vào bộ sưu tập.

5: Tiền điều kiện:

Người dùng đã xem chi tiết về một ảnh.

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.6 Mô tả use case Sinh biến thể ảnh mới

1: Tên Usecase: Sinh biến thể ảnh mới

2: Mô tả vắn tắt: Người dùng tạo ra một ảnh mới bằng cách sử dụng mô hình tạo ảnh là biến thể của dữ liệu đầu vào (tranh chân dung).

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng nhấn vào nút “Sinh biến thể ảnh mới” ở trang chủ.

2) Hệ thống tạo ra một ảnh biến thể ngẫu nhiên bằng cách sử dụng mô hình sinh biến thể ảnh mới từ bộ ảnh cũ dcgan_generator.

3) Hệ thống hiển thị ảnh vừa được tạo cho người dùng.

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

Không có

4: Yêu cầu đặc biệt:

Không có

5: Tiền điều kiện:

Không có

6: Hậu điều kiện:

Không có

7: Điểm mở rộng:

Không có

4.2.2.7 Mô tả use case Tạo ảnh chi tiết từ sketch

1: Tên Usecase: Tạo ảnh từ sketch

2: Mô tả văn tắt: Người dùng tạo ra một ảnh từ một bức tranh phác thảo.

3: Các luồng sự kiện:

3.1: Luồng cơ bản:

1) Người dùng truy cập vào trang tạo ảnh từ sketch.

2) Người dùng tải lên một bức tranh phác thảo (sketch).

3) Hệ thống tạo ảnh từ sketch bằng cách sử dụng mô hình tạo ảnh pix2pix_generator.

4) Hệ thống hiển thị ảnh được tạo ra cho người dùng.

Usecase kết thúc.

3.2: Luồng rẽ nhánh:

1) Nếu người dùng không tải lên bức tranh phác thảo: Hệ thống hiển thị thông báo lỗi và yêu cầu người dùng tải lên một bức tranh phác thảo.

Usecase kết thúc.

4: Yêu cầu đặc biệt:

Không có.

5: Tiền điều kiện:

Không có

6: Hậu điều kiện:

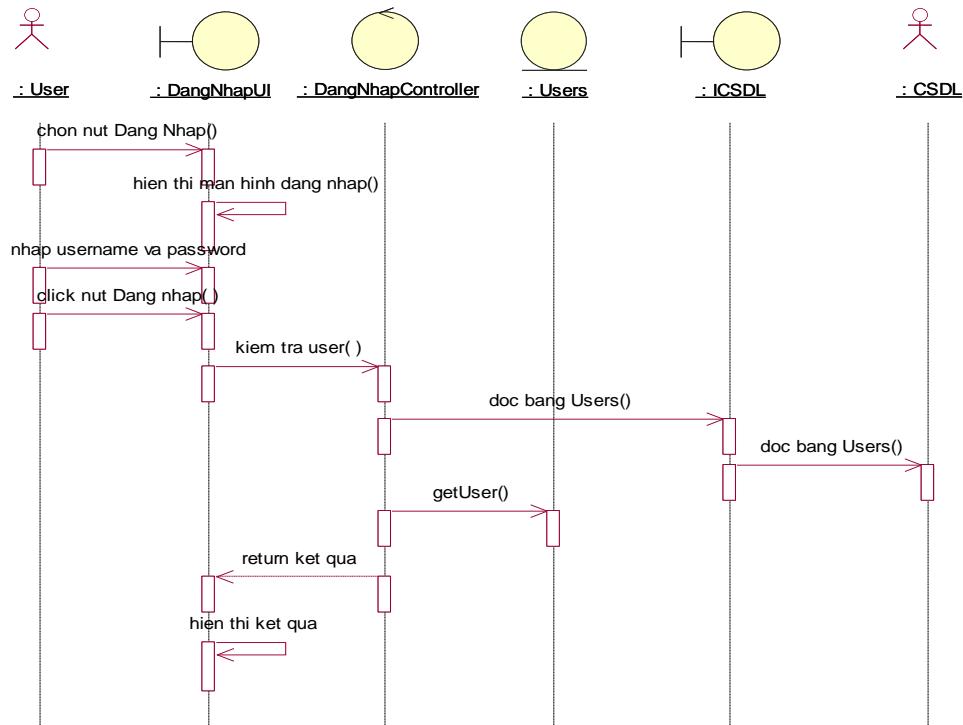
Không có

7: Điểm mở rộng:

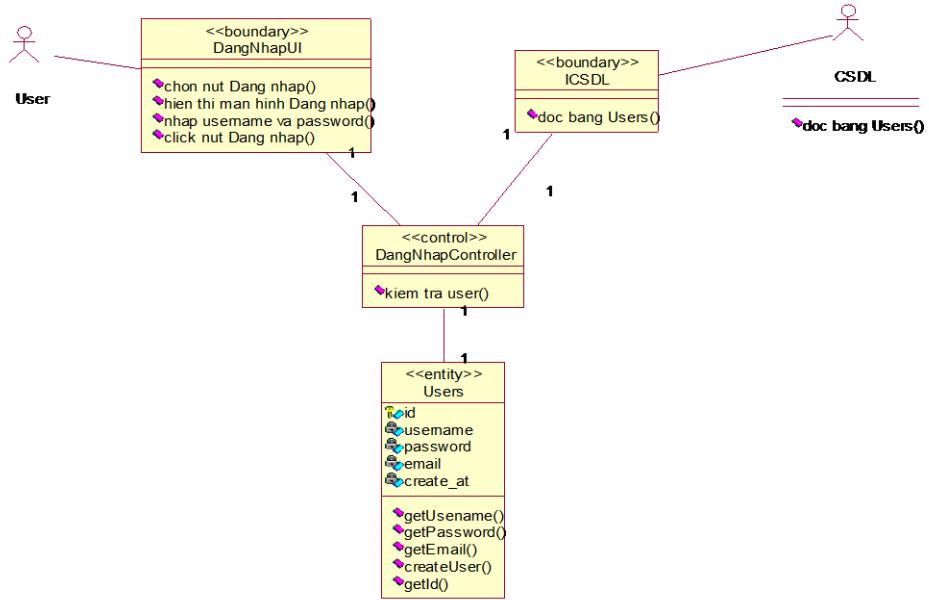
Không có

4.2.3. Phân tích các use case

4.2.3.1 Use case Đăng nhập

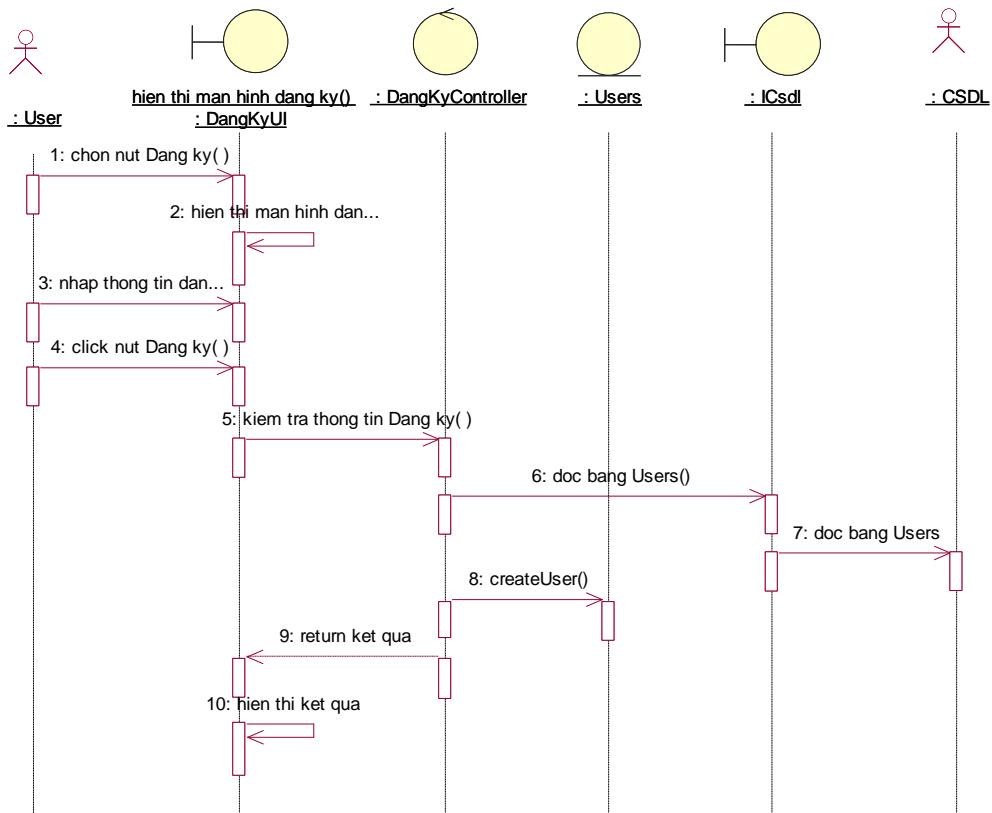


Hình 4.3 Biểu đồ trình tự use case Đăng nhập

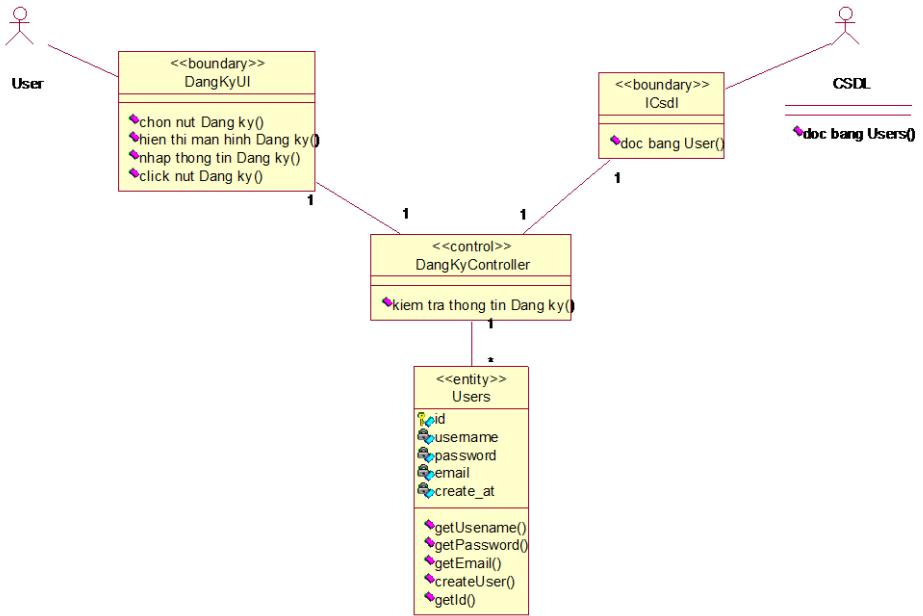


Hình 4.4 Biểu đồ lớp phân tích use case Đăng nhập

4.2.3.2 Use case Đăng ký

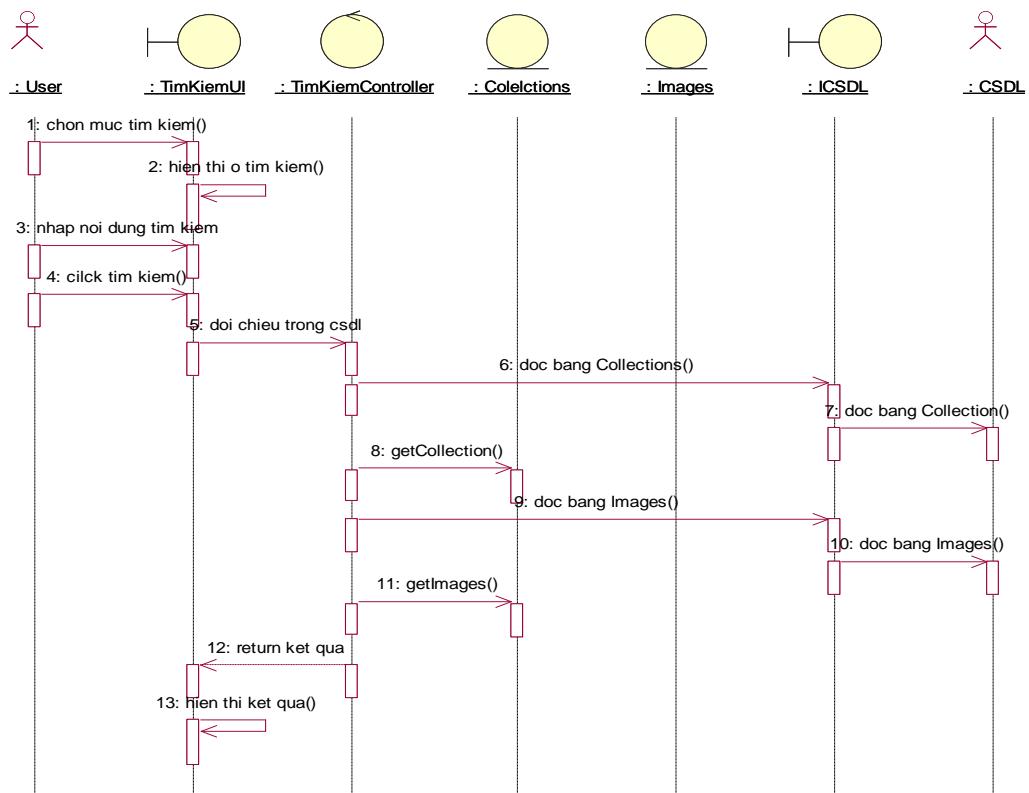


Hình 4.5 Biểu đồ trình tự use case Đăng ký

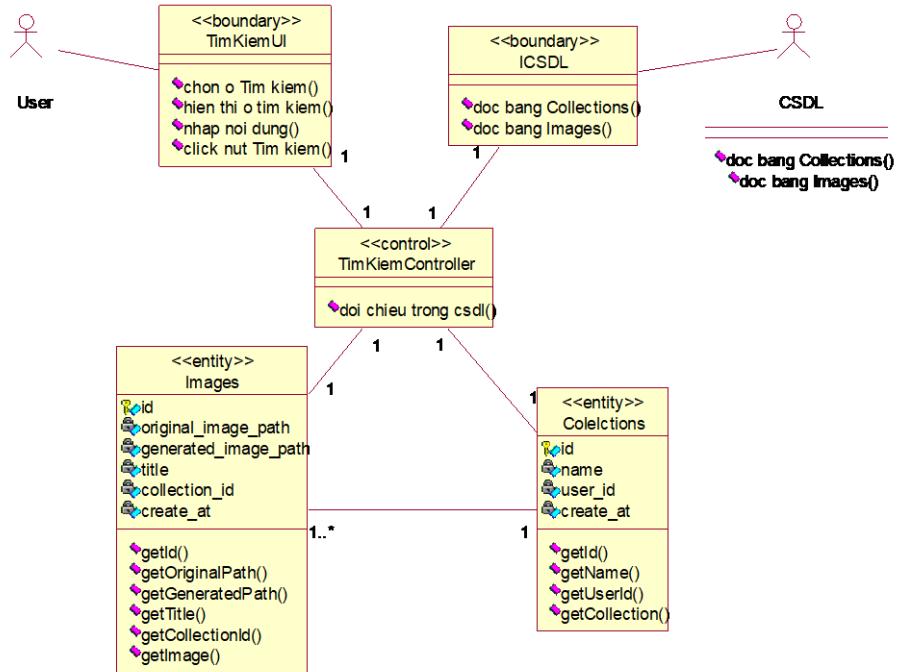


Hình 4.6 Biểu đồ lớp phân tích use case Đăng ký

4.2.3.3 Use case Tìm kiếm

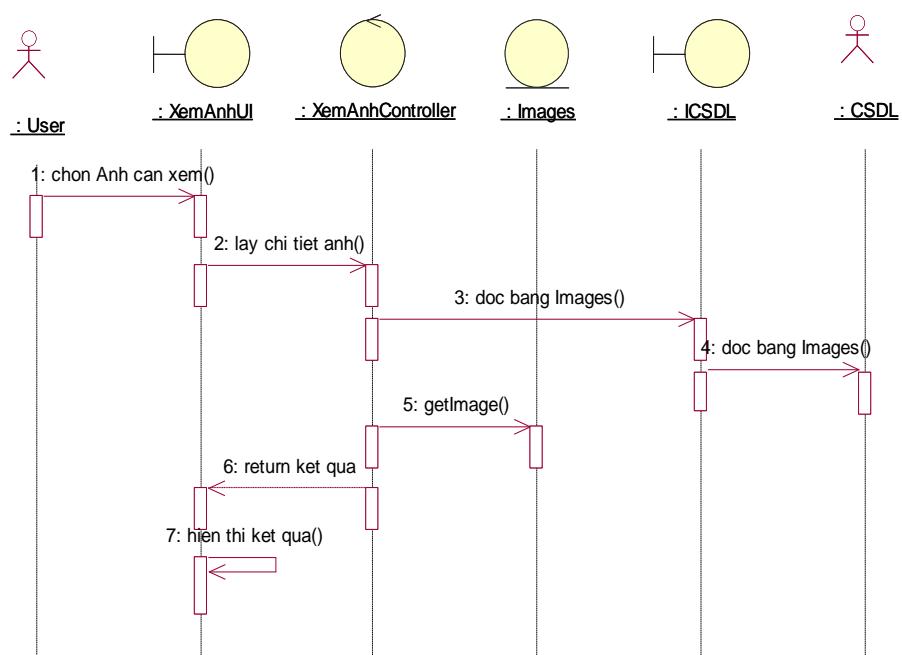


Hình 4.7 Biểu đồ trình tự use case Tìm kiếm

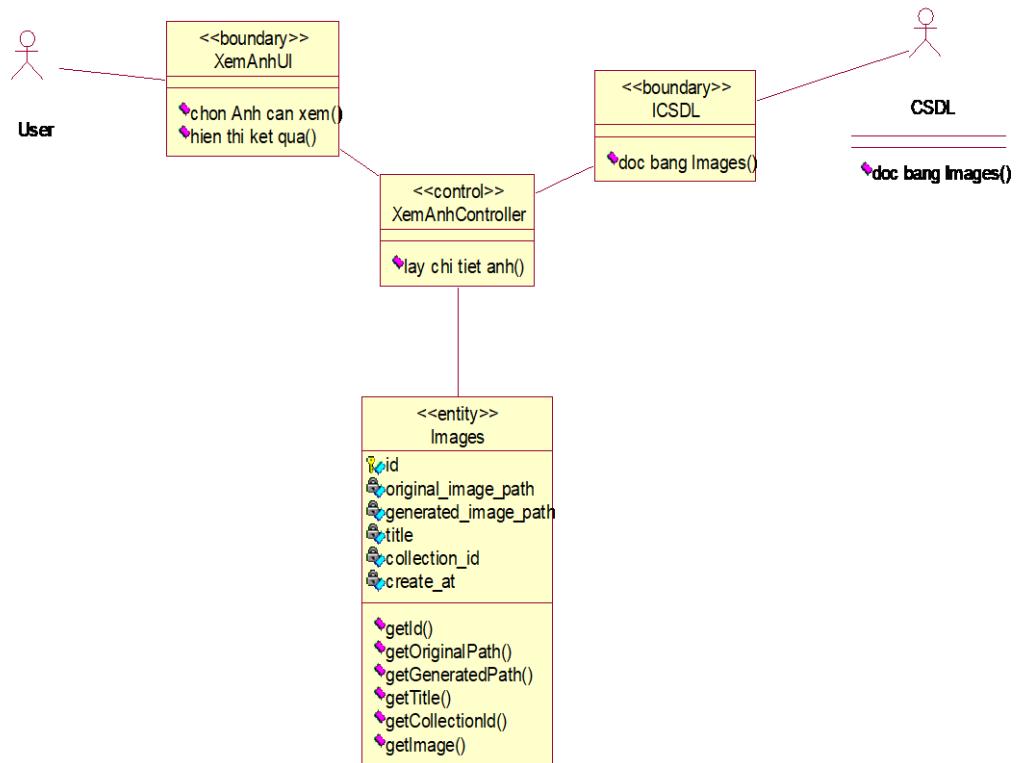


Hình 4.8 Biểu đồ lớp phân tích use case Tìm kiếm

4.2.3.4 Use case Xem ảnh

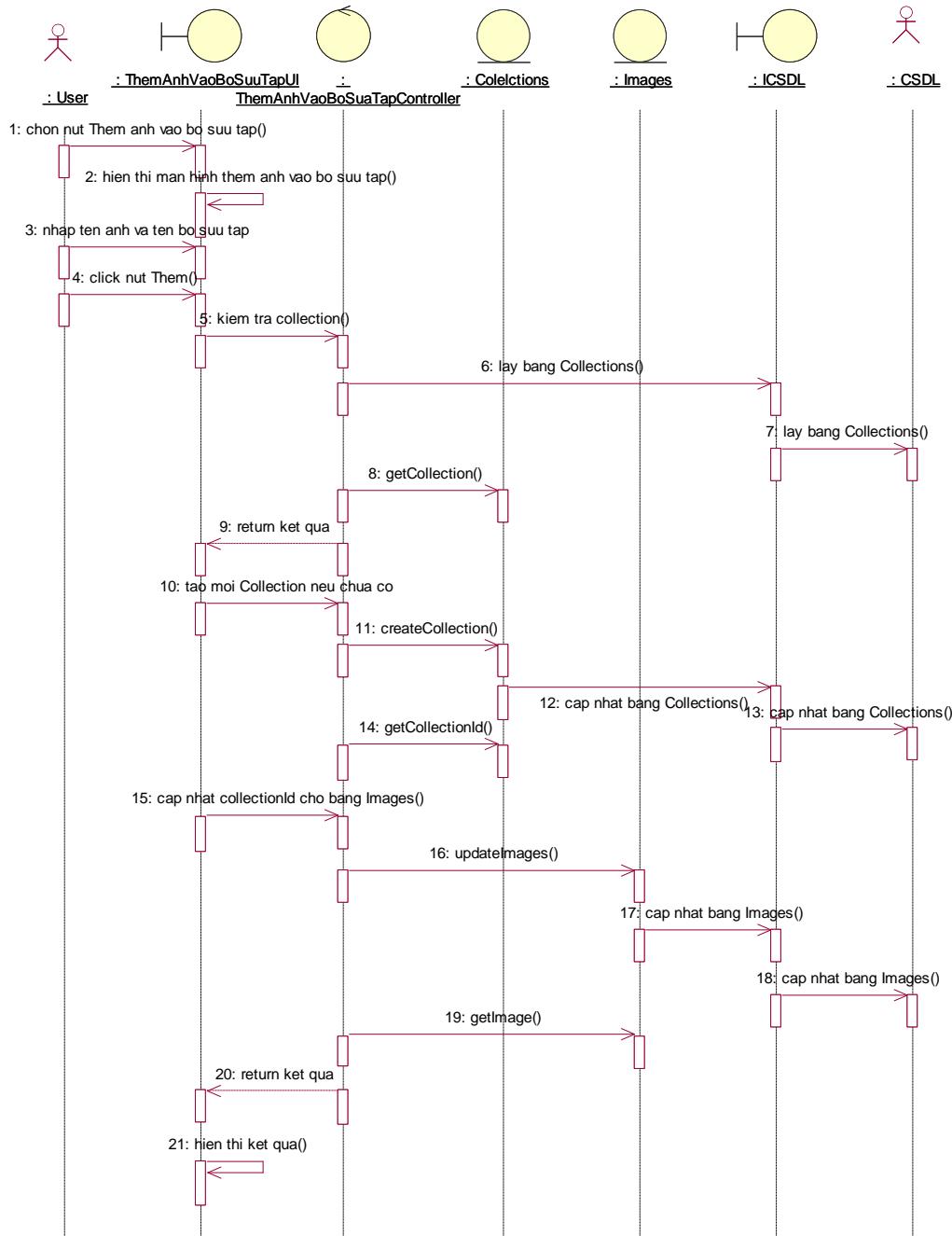


Hình 4.9 Biểu đồ trình tự use case Xem ảnh

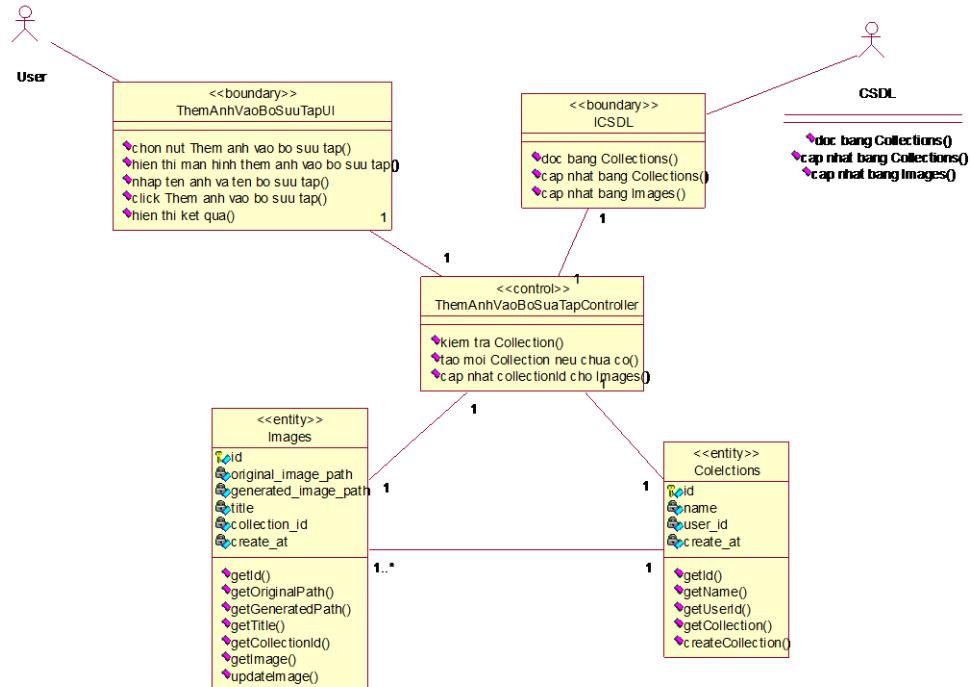


Hình 4.10 Biểu đồ lớp phân tích use case Xem ảnh

4.2.3.5 Use case Thêm ảnh vào bộ sưu tập

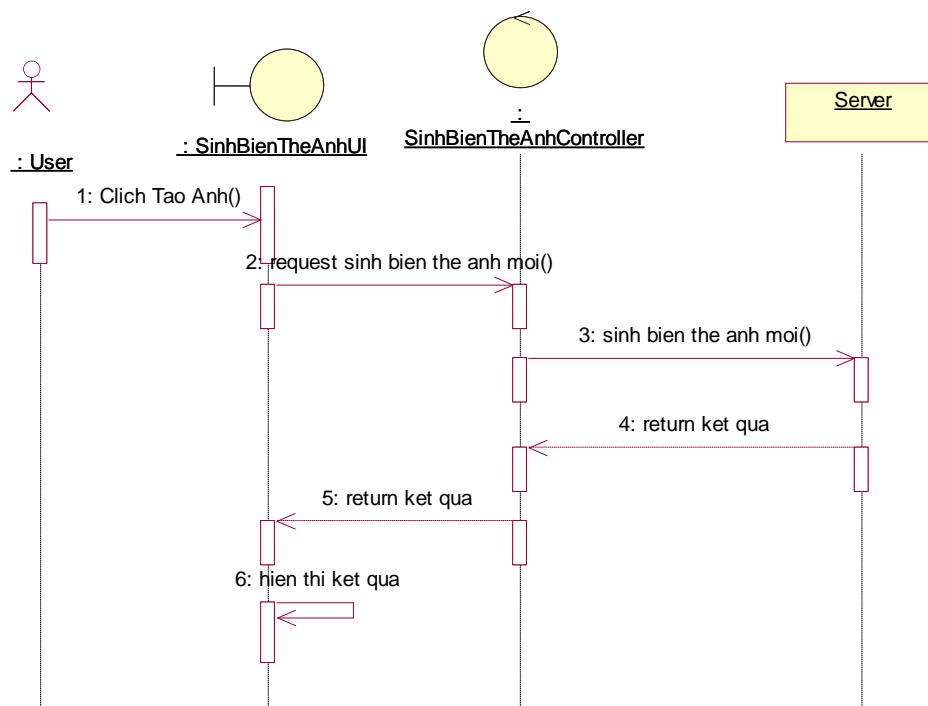


Hình 4.11 Biểu đồ trình tự use case Thêm ảnh vào bộ sưu tập

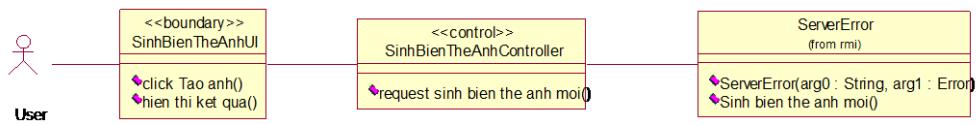


Hình 4.12 Biểu đồ lớp phân tích use case Thêm ảnh vào bộ sưu tập

4.2.3.6 Sinh biến thể ảnh mới

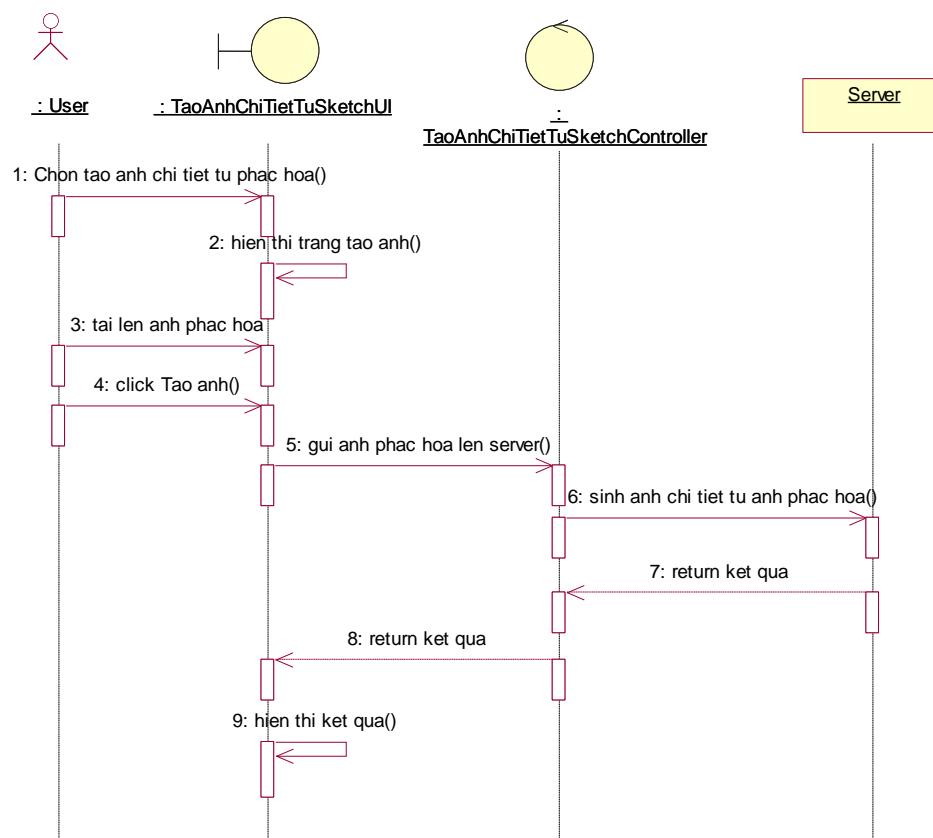


Hình 4.13 Biểu đồ trình tự use case Sinh biến thể ảnh mới

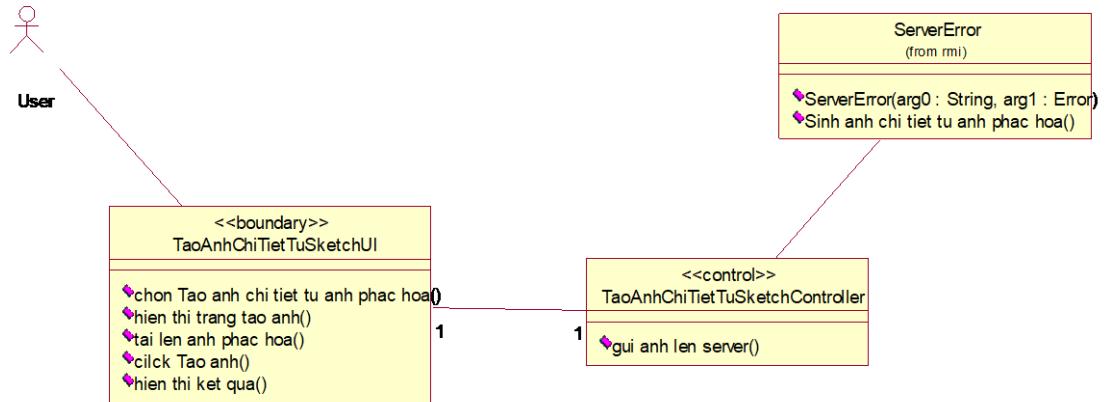


Hình 4.14 Biểu đồ lớp phân tích use case Sinh biển thẻ ảnh mới

4.2.3.7 Use case Tao anh chi tiết từ Sketch



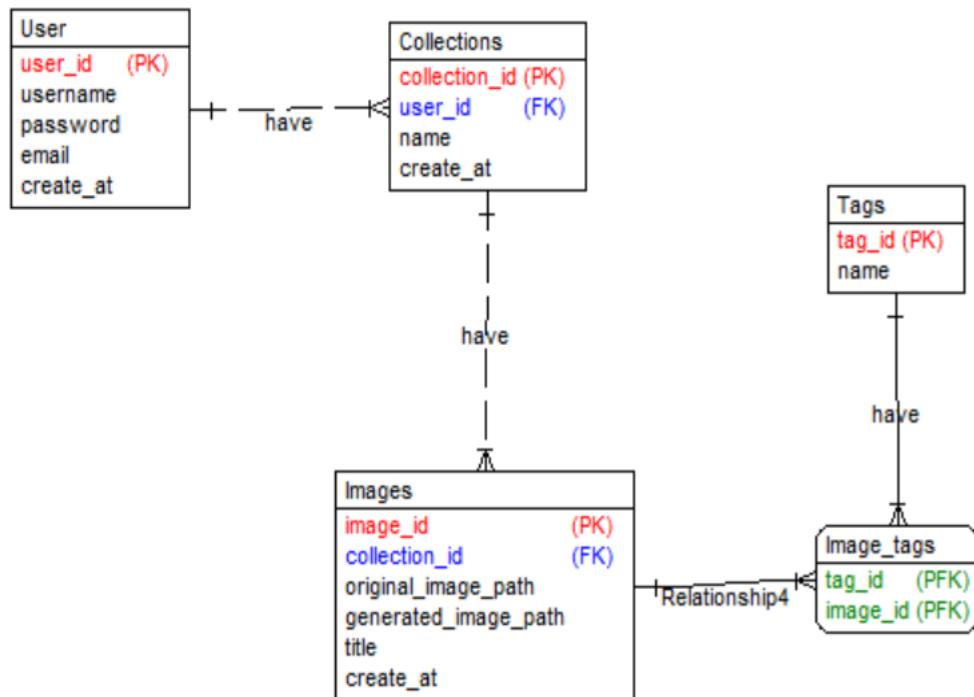
Hình 4.15 Biểu đồ trình tự use case Tao anh chi tiet từ sketch



Hình 4.16 Biểu đồ lớp phân tích use case Tạo ảnh chi tiết từ sketch

4.2.4. Thiết kế cơ sở dữ liệu

Mô hình cơ sở dữ liệu quan hệ:



Hình 4.17 Mô hình cơ sở dữ liệu của hệ thống

Chi tiết các bảng trong cơ sở dữ liệu:

Bảng 4.1 Chi tiết bảng User

Tên cột	Kiểu dữ liệu	Null	Ràng buộc	Mô tả
Id	Int	Not null	Primary key, auto increment	Mã của người dùng
Username	Text	Not null	Không	Tên đăng nhập
Password	Text	Not null	Không	Mật khẩu
Email	Text	Not null	Không	Email
Create_at	Time_Stamp	Not null	Không	Thời gian tạo

Bảng 4.2 Chi tiết bảng Collections

Tên cột	Kiểu dữ liệu	Null	Ràng buộc	Mô tả
Id	Int	Not null	Primary key, auto increment	Mã bộ sưu tập
Name	Text	Not null	Không	Tên bộ sưu tập
User_id	Int	Not null	Foreign key bảng User	Tên người tạo bộ sưu tập
Create_at	Time_Stamp	Not null	Không	Thời gian tạo

Bảng 4.3 Chi tiết bảng Images

Tên cột	Kiểu dữ liệu	Null	Ràng buộc	Mô tả
Id	Int	Not null	Primary key, auto increment	Mã ảnh
Original_image_path	Text	Not null	Không	Địa chỉ ảnh gốc
Generated_image_path	Text	Not null	Không	Địa chỉ ảnh tạo
Title	Text		Không	Tên ảnh
Collection_id	Int		Foreign key bảng Collections	Mã bộ sưu tập
Create_at	Time_Stamp	Not null	Không	Thời gian tạo

Bảng 4.4 Chi tiết bảng Tags

Tên cột	Kiểu dữ liệu	Null	Ràng buộc	Mô tả
Id	Int	Not null	Primary key, auto increment	Mã thẻ
Name	Text	Not null	Không	Tên thẻ

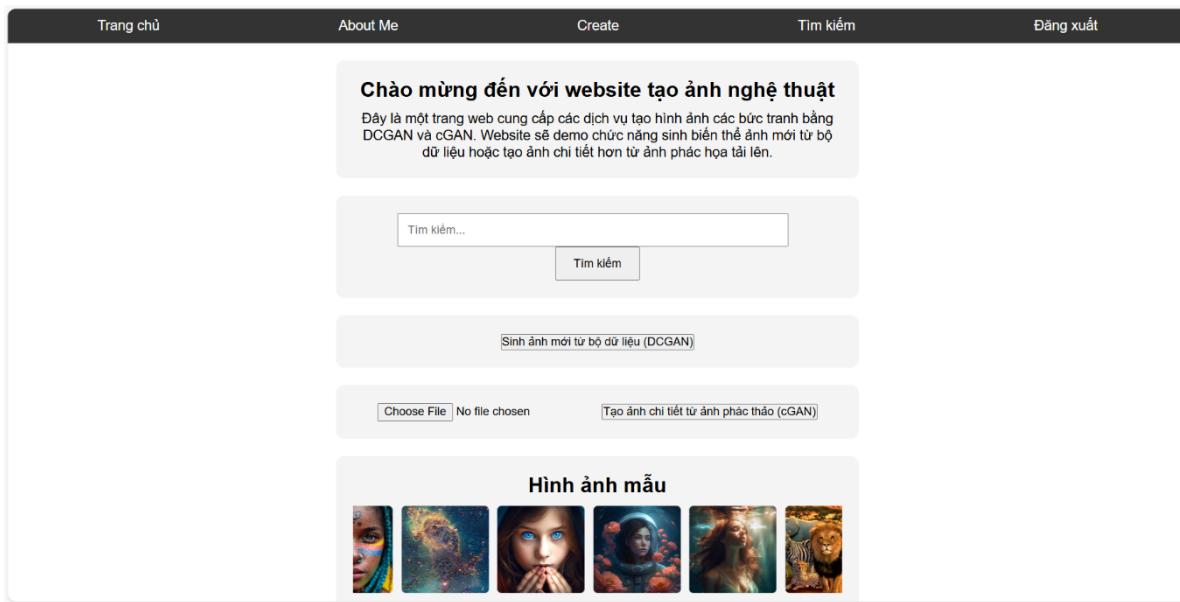
Bảng 4.5 Chi tiết bảng Image_tags

Tên cột	Kiểu dữ liệu	Null	Ràng buộc	Mô tả
Image_id	Int	Not null	Primary key, Foreign key bảng Images	Mã ảnh
Tag_id	Int	Not null	Primary key, Foreign key bảng Tags	Mã thẻ

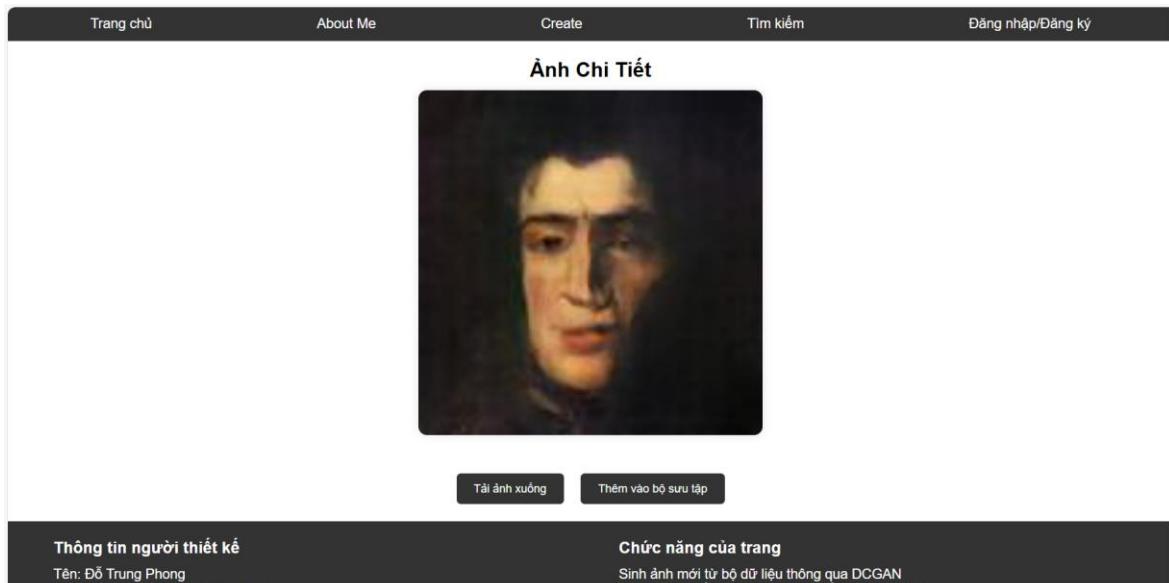
4.2.5. Thiết kế giao diện hệ thống

Hệ thống sẽ được chia thành 6 giao diện, gồm có:

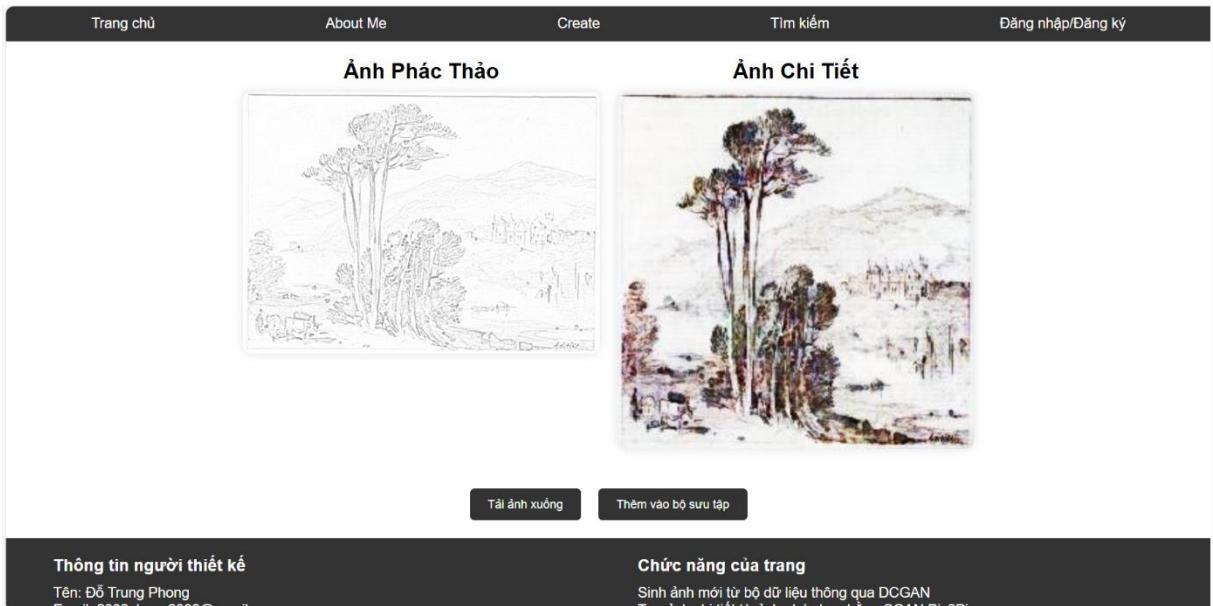
- Giao diện trang chủ: là giao diện chính của hệ thống, là nơi user có thể thực hiện tìm kiếm, sinh biến thẻ ảnh mới và tạo ảnh chi tiết từ ảnh phác họa tải lên.
- Giao diện trang kết quả tìm kiếm: là giao diện hiển thị kết quả tìm kiếm được nhập từ giao diện trang chủ.
- Giao diện trang hiển thị ảnh: là giao diện hiển thị ảnh được tạo từ dcgan_generator hoặc ảnh được mở từ trang tìm kiếm.
- Giao diện hiển thị ảnh chi tiết từ ảnh phác họa: là giao diện hiển thị ảnh được tạo từ cgan_generator và ảnh phác họa mà người dùng tải lên ở trang chủ.
- Giao diện đăng nhập: giao diện để người dùng đăng nhập vào hệ thống.
- Giao diện đăng ký: giao diện để người dùng đăng ký tài khoản.



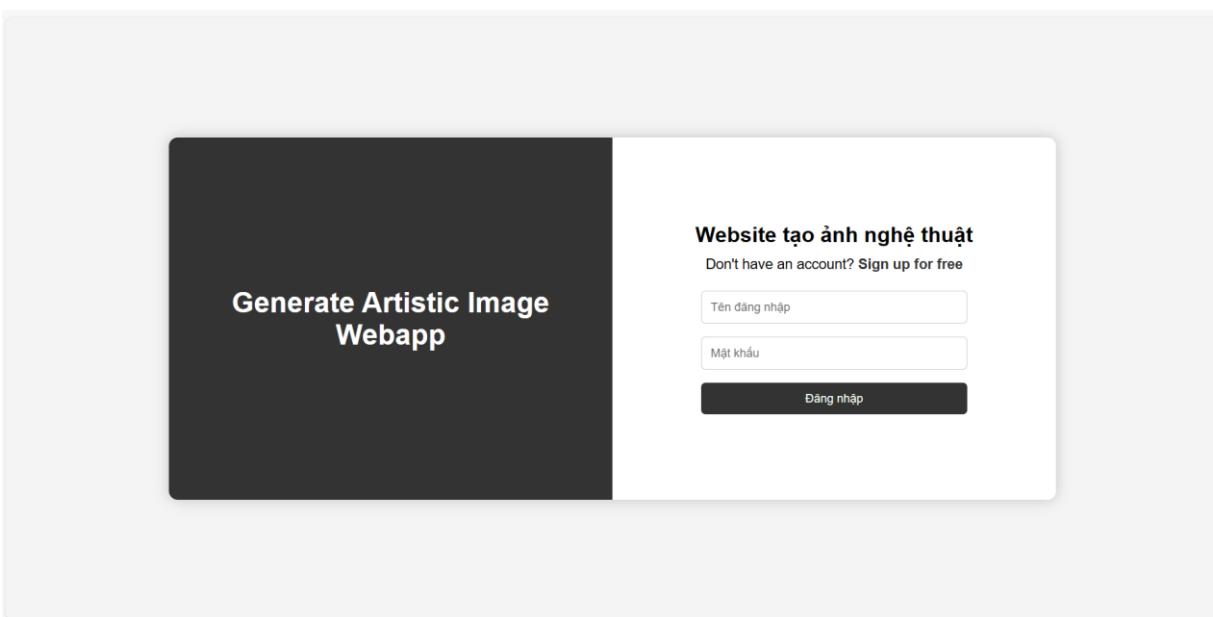
Hình 4.18 Giao diện trang chủ



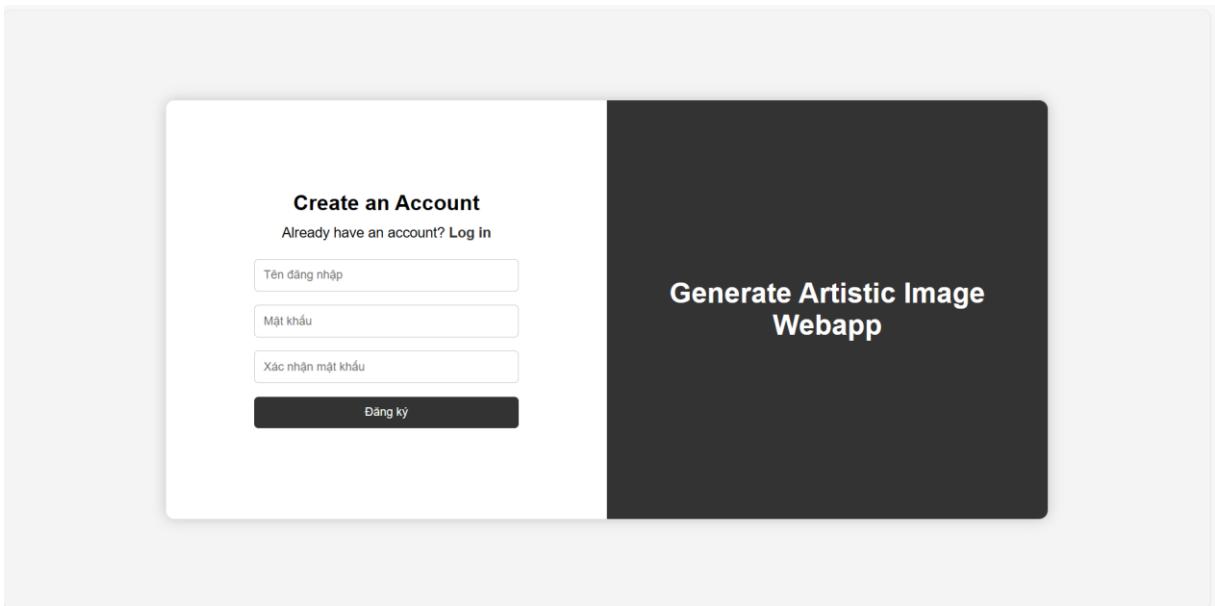
Hình 4.19 Giao diện trang sinh biến thể ảnh mới



Hình 4.20 Giao diện tạo ảnh chi tiết từ ảnh phác thảo



Hình 4.21 Giao diện trang đăng nhập



Hình 4.22 Giao diện trang đăng ký



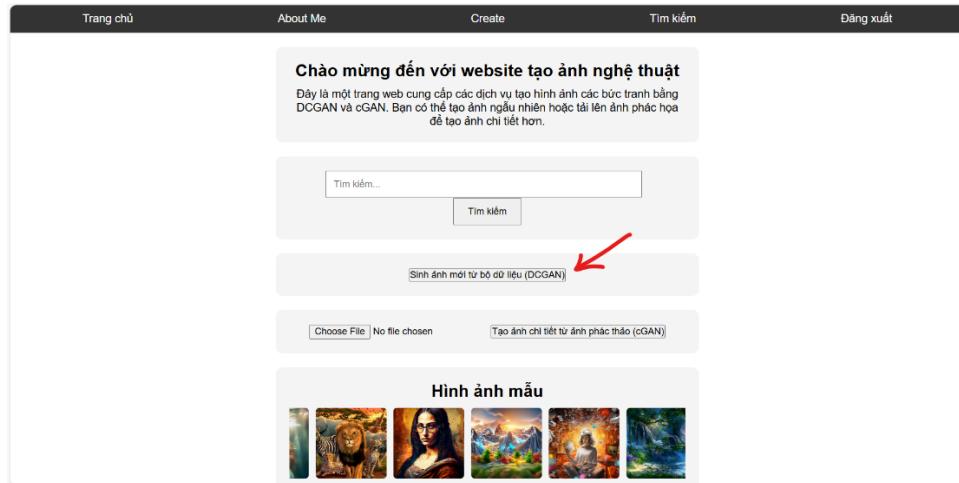
Hình 4.23 Giao diện trang kết quả tìm kiếm

4.3. Các chức năng của hệ thống

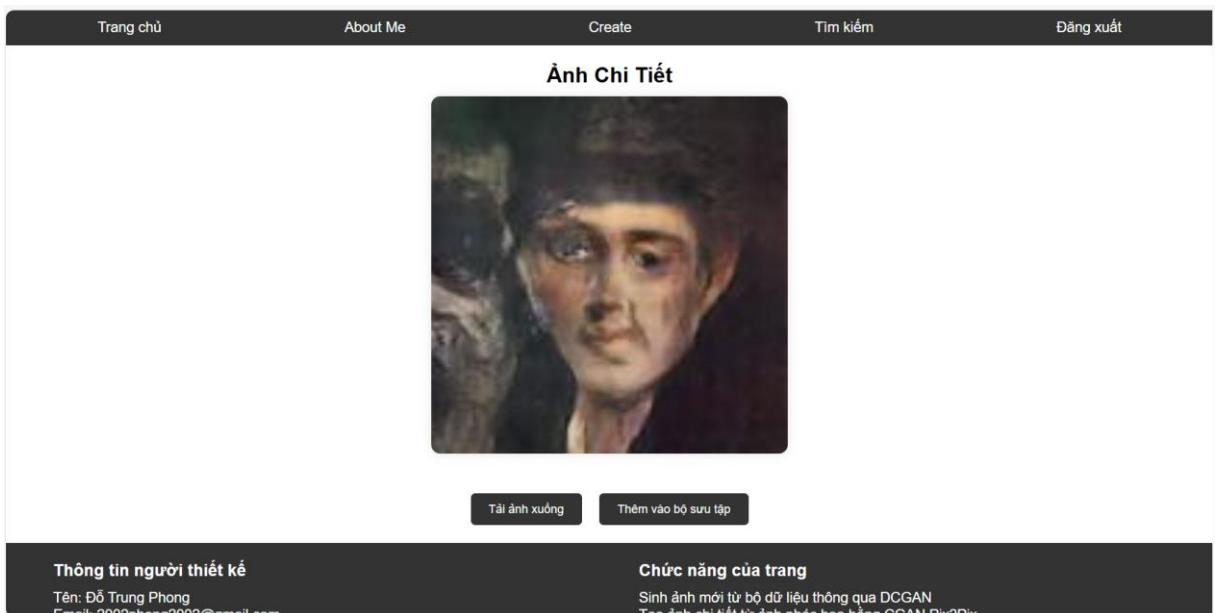
4.3.1. Chức năng chính

Hệ thống có chức năng chính là sinh biến thế ảnh mới từ bộ ảnh và tạo ảnh chi tiết từ ảnh phác họa. Với chức năng sinh biến thế ảnh mới từ bộ ảnh, hệ thống cung cấp tính năng sinh biến thế ảnh mới từ bộ ảnh bằng cách sử dụng mô hình DCGAN. Hình ảnh được tạo ra sau đó được lưu vào thư mục tạm và hiển thị cho người dùng.

Ở màn hình trang chủ, người dùng click vào nút “Sinh ảnh mới từ bộ dữ liệu (DCGAN)”:



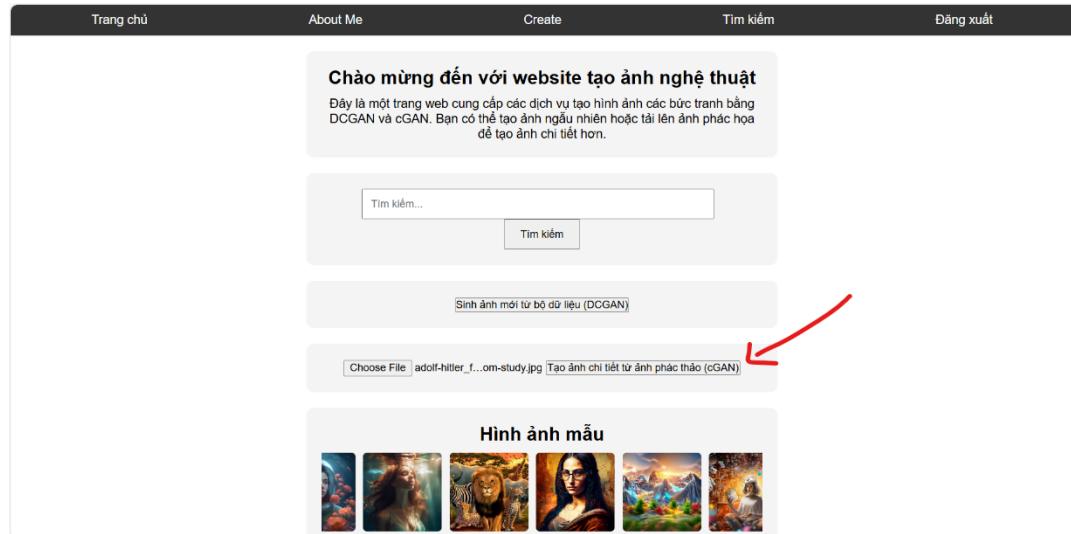
Hình 4.24 Test chức năng Sinh biến thể ảnh mới từ bộ dữ liệu



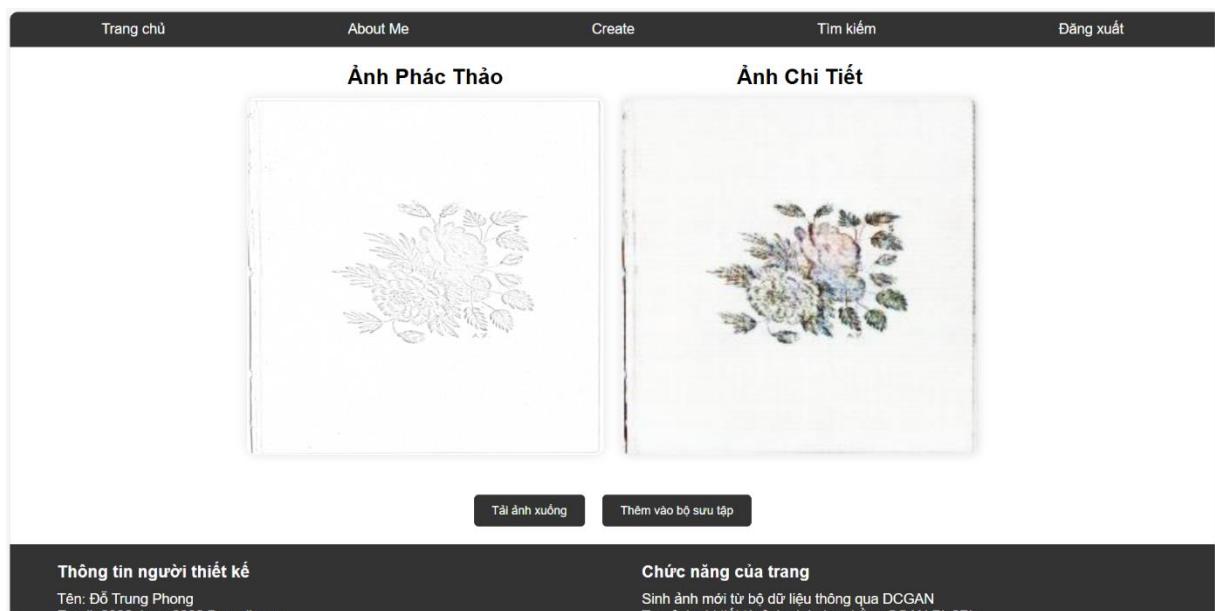
Hình 4.25 Kết quả test chức năng Sinh biến thể ảnh mới từ bộ dữ liệu

Với chức năng tạo ảnh chi tiết từ ảnh phác họa, hệ thống sẽ sử dụng ảnh phác họa mà người dùng tải lên làm đầu vào cho mô hình CGAN pix2pix để sinh ra hình ảnh chi tiết tương ứng. Hình ảnh được tạo ra sau đó được lưu vào thư mục tạm và hiển thị cho người dùng.

Ở màn hình trang chủ, người dùng click vào nút “Choose file” để tải ảnh phác họa lên, sau đó click nút “Tạo ảnh chi tiết từ ảnh phác thảo (cGAN)”:



Hình 4.26 Test chức năng tạo ảnh chi tiết từ ảnh phác thảo



Hình 4.27 Kết quả test chức năng tạo ảnh chi tiết từ ảnh phác họa

4.3.2. Một số chức năng khác

Ngoài chức năng chính là tạo ảnh, hệ thống còn cung cấp các chức năng tìm kiếm hình ảnh, tạo bộ sưu tập và tải ảnh.

KẾT LUẬN

Thời gian thực hiện đồ án tốt nghiệp vừa qua là một trải nghiệm thú vị đối với bản thân em. Việc nghiên cứu và học hỏi một mô hình mạng thuộc loại Generative trong một khoảng thời gian khá ngắn là một thách thức không nhỏ đối với em. Em đã tìm hiểu nhiều biến thể khác nhau của mô hình học sâu GAN nhằm giải quyết bài toán tạo ảnh nghệ thuật, từ đó học được những kỹ năng quý giá như kỹ năng làm việc độc lập, quản lý thời gian, kỹ năng xây dựng kế hoạch hay các kỹ năng chuyên ngành. Những kỹ năng này sẽ trở thành những trợ lực không nhỏ trong sự nghiệp tương lai của em.

Em đã tìm hiểu và nghiên cứu về các biến thể của mô hình học sâu GAN cũng như những tài liệu và những nghiên cứu từng được công bố để hoàn thành đề tài Tìm hiểu về mô hình học sâu GAN (Generative Adversarial Network) và ứng dụng vào tạo ảnh nghệ thuật. Ngoài ra, em cũng đã thiết kế được một hệ thống tích hợp đơn giản sử dụng flask framework để tạo ảnh và quản lý hình ảnh đã tạo.

Tuy đã xây dựng thành công 2 mô hình cụ thể là DCGAN để sinh biến thể ảnh mới từ bộ ảnh và CGAN để tạo ảnh chi tiết từ hình ảnh phác họa, nhưng bởi giới hạn về thời gian và tài nguyên hạn chế nên đầu ra vẫn còn chưa tốt, thời gian huấn luyện rất lâu và cần phải cải thiện nhiều trong tương lai.

Sau đây là những điểm có thể cải thiện để có thể xây dựng được mô hình tốt hơn: nâng cấp chất lượng và số lượng của dữ liệu đầu vào, nâng cấp tài nguyên phần cứng giúp tối ưu hóa hiệu xuất huấn luyện và nâng cấp mô hình để huấn luyện tốt hơn.

Em xin được gửi lời cảm ơn chân thành nhất tới thầy giáo, Thạc sĩ Nguyễn Thanh Hải đã hướng dẫn em thực hiện đề tài này. Em xin chúc thầy luôn luôn mạnh khỏe và thành công trong những nghiên cứu sắp tới.

Em xin trân trọng cảm ơn!

Đỗ Trung Phong

TÀI LIỆU THAM KHẢO

Tài liệu tiếng Việt:

- [1] Phùng Đức Hòa (Chủ biên); Hoàng Quang Huy; Hoàng Văn Hoành, 2019
Nguyễn Đức Lưu; Trịnh Bá Quý, *Giáo trình Nhập môn công nghệ phần mềm*,
Thống kê.
- [2] Nguyễn Thanh Tuấn, 2020, *Deep Learning Cơ Bản*.
- [3] Vũ Hữu Tiệp, 2020, *Machine Learning Cơ Bản*.

Tài liệu tiếng Anh:

- [4] Kevin P. Murphy, 2012, *Machine Learning: A Probabilistic Perspective*.
- [5] Ian J. Goodfellow, 2014, *Generative Adversarial Networks*.

Website:

- [6] Tìm hiểu về mạng tích chập (CNN),
URL: <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>, truy cập gần nhất: 17/05/2024.
- [7] Pix2Pix – Image-to-Image Translation Neural Network,
URL: <https://neurohive.io/en/popular-networks/pix2pix-image-to-image-translation/>, truy cập gần nhất: 17/05/2024.
- [8] Overview of GAN Structure
URL: https://developers.google.com/machine-learning/gan/gan_structure, truy cập gần nhất: 17/05/2024.
- [9] Generative Adversarial Network,
URL: <https://nttuan8.com/category/gan/>, truy cập gần nhất: 17/05/2024.
- [10] Series GAN model trong Khoa học dữ liệu – Khanh’s blog,
URL: <https://phamdinhkhanh.github.io/2020/07/13/GAN.html>, truy cập gần nhất: 17/05/2024.

[11] Python Flask Là Gì?,

URL: <https://www.mcivietnam.com/blog-detail/flask-python-la-gi-so-sanh-flask-va-django/>, truy cập gần nhất: 17/05/2024.

[12] GAN series,

URL: <https://viblo.asia/p/gan-series-1-co-ban-ve-gan-trong-deep-learningbWrZnE4YKxw>, truy cập gần nhất: 17/05/2024.