

Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:

протокол засідання кафедри

№ ____ від ____ . ____ .2025 р.

Завідувач кафедри ІПСіТ

(підпис) Олешко О.І.
(прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Розробка та дослідження штучних нейронних мереж для генерації
зображень»

Захищено на засіданні ЕК № _____
протокол № ____ від ____ . ____ .2025 р.
Оцінка _____ / _____
Голова ЕК

(підпис) Фесенко Г.В.
(прізвище та ініціали)

Виконав:

студент 4 курсу, групи КС-42

Спеціальності:

122 – Комп'ютерні науки

(шифр і назва спеціальності підготовки)

Літвіненко Дмитро Володимирович

(прізвище, ім'я та по батькові, підпис)

Керівник к.ф.-м.н., доцент Карась І.В.

(ступень, звання, прізвище та ініціали, підпис)

Рецензент к.е.н., доцент Чуб О.І.

(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра інтелектуальних програмних систем і технологій

Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр

Напрямок підготовки 12 Інформаційні технології

Спеціальність 122 – Комп'ютерні науки

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІПСіТ

_____ Олег ОЛЕШКО
підпис ім'я, прізвище

“ _____ ” _____ 2025 року

**З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Літвіненко Дмитро Володимирович
(прізвище, ім'я, по батькові студента)

1. Тема роботи: «Розробка та дослідження штучних нейронних мереж для генерації зображень»

керівник роботи Карась Ірина Вячеславівна, кандидат фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “16” квітня 2025 року № 4101-5/962

2. Строк подання студентом роботи _____ 02.06.2025 р. _____

3. Перелік питань, які потрібно розробити:

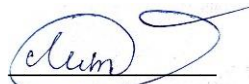
Розглянути задачу генерації зображень у сучасній сфері інформаційних технологій. Ознайомитися з науковими досягненнями в галузі створення штучних нейронних мереж (ШНМ) для генерації зображень. Провести огляд існуючих концепцій, типів моделей і методів розробки генеративних мереж, ознайомившись з архітектурами, алгоритмами і механізмами їх роботи. Спроекувати та програмно втілити низку розглянутих моделей мереж. Привести результати програмних реалізацій втілених моделей та сформулювати висновки щодо доцільності їх використання для вирішення поставленої задачі.

4. План роботи

№ з/п	Назви етапів роботи
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи (КР).
2	Пошук та аналіз інформаційних джерел за темою КР.
3	Огляд сучасного становища сфери інформаційних технологій у галузі генерацій зображень. Вивчення питань актуальності та проблематики вирішення поставленої задачі.
4	Дослідження сучасних наукових досягнень і праць за тематикою.
5	Вивчення концепцій, алгоритмів роботи та можливих архітектур генеративно-змагальних і дифузійних моделей ШНМ для генерації зображень.
6	Проектування архітектури та розробка простої генеративно-змагальної мережі (GAN).
7	Проектування архітектури та розробка неявної дифузійної моделі з усуненням шуму (DDIM).
8	Проектування архітектури та розробка векторно-квантової генеративно-змагальної мережі (VQ-GAN).
9	Програмна реалізація спроектованих моделей для генерації зображень на мові програмування Python за допомогою фреймворку TensorFlow.
10	Вибір набору даних (датасетів) для навчання моделей та його попередня обробка. Навчання програмно втілених генеративних моделей на представлених наборах зображень.
11	Тестування розроблених моделей для вирішення задачі генерації зображень. Аналіз отриманих результатів.
12	Формування загального підсумку щодо виконаної роботи та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ІІІ.

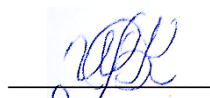
5. Дата видачі завдання 25.01.2025 р.

Студент


(підпис)

Літвіненко Д.В.
(ініціали, прізвище)

Керівник роботи


(підпис)

Карась І.В.
(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка до проекту бакалавра містить 91 сторінку, 37 рисунків, 6 формул, 30 посилань на джерела, 6 додатків.

Мета роботи полягає в дослідженні шляхів вирішення задачі генерування зображень на базі використання сучасних технологічних концепцій, алгоритмів і методів у сфері штучних нейронних мереж, що втілюються моделями генеративних мереж, а також в аналізі актуальності та проблематики реалізації завдання на поточному рівні розвитку інформаційних наук. Основне завдання визначається проведенням як теоретичного вивчення існуючих моделей генеративних мереж, так і програмним втіленням механізмів і принципів їх роботи, аналізу отримуваних результатів.

Об'єкт дослідження – процес генерування інформаційних об'єктів, що представляють собою цифрові растрові зображення.

Предмет дослідження – моделі генеративно-змагальних і дифузійних штучних нейронних мереж для створення зображень, основні механізми, фактори й можливості їх реалізації.

Основними методами досліджень є аналіз та порівняння інструментів та методів вирішення проблематики генерування зображень за рахунок втілення моделей генеративних мереж.

У роботі досліджено: задача генерування зображень; методологія генеративних мереж; генеративно-змагальні та дифузійні моделі, що використовуються для створення зображень.

Результати роботи можуть бути використані для подальшої розробки користуальницьких програмних продуктів, а також для кращого розуміння процесу та технік генерування зображень.

Ключові слова: ГЕНЕРУВАННЯ, ЗОБРАЖЕННЯ, ШТУЧНА НЕЙРОННА МЕРЕЖА, ГЕНЕРАТИВНО-ЗМАГАЛЬНА МОДЕЛЬ, ДИФУЗІЙНА МОДЕЛЬ, АРХІТЕКТУРА МЕРЕЖІ, ДАТАСЕТ, НАВЧАННЯ.

ABSTRACT

The explanatory note to the bachelor's project contains 91 pages, 37 figures, 6 formulas, 30 sources, 6 appendices.

The purpose of the work is to study ways to solve the problem of image generation based on the use of modern technological concepts, algorithms and methods in the field of artificial neural networks, which are embodied in generative network models, as well as to analyse the relevance and problems of the task at the current level of development of information sciences. The main task is determined by both theoretical study of existing models of generative networks and software implementation of mechanisms and principles of their operation, analysis of the results obtained.

The object of research is the process of generating information objects that are digital raster images.

The subject of the research is models of generative-competitive and diffusion artificial neural networks for image creation, the main mechanisms, factors and possibilities of their implementation.

The main research methods are the analysis and comparison of tools and methods for solving the problem of image generation through the implementation of generative network models.

The paper investigates: the task of generating images; the methodology of generative networks; generative-adversarial and diffusion models used to create images.

The results can be used for further development of custom software products, as well as for a better understanding of the process and techniques of image generation.

Keywords: GENERATION, IMAGE, ARTIFICIAL NEURAL NETWORK, GENERATIVE-ADVERSARIAL MODEL, DIFFUSION MODEL, NETWORK ARCHITECTURE, DATASET, TRAINING.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	7
ВСТУП.....	8
1 АНАЛІЗ ЗАДАЧІ ГЕНЕРУВАННЯ ЗОБРАЖЕНЬ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	11
1.1 Постанова задачі генерування зображень.....	11
1.2 Основи генеративних мереж. Особливості їх створення та розробки.....	12
1.3 Аналіз сучасного становища сфери інформаційних технологій у галузі створення ШНМ для генерування зображень.....	16
1.4 Дослідження проблематики створення генеративних мереж	23
1.5 Огляд існуючих програмних рішень для вирішення задачі генерування зображень.....	24
2 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРОЄКТУВАННЯ ГЕНЕРАТИВНИХ МЕРЕЖ.....	26
2.1 Аналіз вибору моделей генеративних мереж для втілення	26
2.2 Генеративно-змагальні мережі	27
2.2.1 Загальні концепції моделей генеративно-змагальних мереж.....	27
2.2.2 Модель простої генеративно-змагальної мережі (GAN).....	29
2.2.3 Модель векторно-квантової мережі (VQ-GAN)	30
2.3 Дифузійні мережі	33
2.3.1 Явище дифузії	33
2.3.2 Загальні концепції моделей дифузійних мереж.....	35
2.3.3 Неявна дифузійна модель з усуненням шуму (DDIM).....	36
3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛЕЙ ГЕНЕРАТИВНИХ МЕРЕЖ.....	39
3.1 Програмно-апаратні рішення	39
3.1.1 Середовище розробки та програмні інструменти.....	39
3.1.2 Особливості обчислювального середовища	40
3.2 Вибір набору даних (датасету).....	40
3.3 Програмне втілення моделі GAN.....	42
3.3.1 Попередня обробка датасету	42
3.3.2 Реалізація архітектури та механізму роботи.....	43
3.3.3 Навчання	46

3.4 Програмне втілення моделі DDIM.....	46
3.4.1 Попередня обробка датасету	46
3.4.2 Реалізація архітектури та механізму роботи.....	47
3.4.3 Навчання	52
3.5 Програмне втілення моделі VQ-GAN	53
3.5.1 Попередня обробка датасету	53
3.5.2 Реалізація архітектури та механізму роботи.....	55
3.5.3 Навчання моделі	61
4 АНАЛІЗ РЕЗУЛЬТАТІВ	62
4.1 Результати роботи генеративних мереж та згенеровані зображення	62
4.1.1 Проведення візуального аналізу	62
4.1.2 Об'єктивне дослідження результатів.....	64
4.2 Аналіз проблем і труднощів програмної реалізації	66
4.3 Рекомендації щодо майбутнього вдосконалення	67
4.4 Передбачувані області використання.....	68
ВИСНОВКИ	70
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	72
ДОДАТОК А. РЕЗУЛЬТАТИ ОЦІНКИ FID-МЕТРИКИ	75
ДОДАТОК Б. РЕЗУЛЬТАТИ ОЦІНКИ IS-МЕТРИКИ.....	76
ДОДАТОК В. РЕЗУЛЬТАТИ ОЦІНКИ LOSS-МЕТРИКИ.....	77
ДОДАТОК Г. ВИХІДНИЙ ПРОГРАМНИЙ КОД МОДЕЛІ ГЕНЕРАТИВНО-ЗМАГАЛЬНОЇ МЕРЕЖІ (GAN).....	78
Лістинг Г.1 – Побудова генератора.....	78
Лістинг Г.2 – Побудова дискримінатора.....	79
Лістинг Г.3 – Збірка моделі мережі (клас Model)	79
ДОДАТОК Д. ВИХІДНИЙ ПРОГРАМНИЙ КОД НЕЯВНОЇ ДИФУЗІЙНОЇ МОДЕЛІ З УСУНЕННЯМ ШУМУ (DDIM)	82
Лістинг Д.1 – Метод ResidualBlock	82
Лістинг Д.2 – Метод DownBlock	82
Лістинг Д.3 – Метод UpBlock	82
Лістинг Д.4 – Побудова мережі U-Net.....	83
Лістинг Д.5 – Збірка моделі мережі (клас DiffusionModel).....	83

ДОДАТОК Е. ВИХІДНИЙ ПРОГРАМНИЙ КОД МОДЕЛІ ВЕКТОРНО-КВАНТОВОЇ МЕРЕЖІ (VQ-GAN).....	86
Лістинг Е.1 – Клас VectorQuantizer	86
Лістинг Е.2 – Метод ResidualBlock	87
Лістинг Е.3 – Побудова кодувальника.....	87
Лістинг Е.4 – Побудова декодувальника	88
Лістинг Е.5 – Побудова дискримінатора.....	89
Лістинг Е.6 – Збірка моделі мережі (клас Model).....	89

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ГЗМ – генеративно-змагальна мережа

ГМ – генеративна мережа

МП – мова програмування

НД – набір даних

ПЗ – програмне забезпечення

ПП – програмний продукт

ПР – програмне рішення

СР – середовище розробки

ІІІ – штучний інтелект

ІІІМ – штучна нейронна мережа

Батч – набір з кількох прикладів (даних), який подається в нейронну мережу за один раз під час навчання чи передбачення

Датасет – спеціалізований набір даних для навчання моделей

Інтерфейс – спосіб взаємодії між усіма компонентами системи

Фреймворк – система правил і компонентів для створення програм

АМ – Autoregressive Models. Авторегресійна модель

DDIM – Denoising Diffusion Implicit Model. Неявна дифузійна модель з усуненням шуму

DM – Diffusion Model. Дифузійна модель.

GAN – Generative Adversarial Nets. Генеративно-змагальна мережа

GPU – Graphical Processing Unit. Графічний процесор

VAE – Variational Autoencoder. Варіаційний автокодувальник

VQ-GAN – Vector Quantized Generative Adversarial Network. Векторно-квантова генеративно-змагальна мережа

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким прогресом у галузі штучного інтелекту (ШІ), особливо у сфері штучних нейронних мереж (ШНМ). Одним із найбільш захоплюючих та перспективних напрямків їх застосування є вирішення задачі генерації зображень. Останніми роками стрімке зростання обчислювальних потужностей та розвиток методів машинного навчання, відкрили принципово нові можливості не лише для розпізнавання та класифікації вже наявних зображень, а й для створення високоякісних синтетичних картин. Завдяки цьому обробка та аналіз зображень набули неабиякої ваги в багатьох галузях інформаційних наук – від комп'ютерної обробки даних та робототехніки до медичної діагностики, архітектури й мистецтва. Також з'явилась величезна кількість застосувань – від генерації реалістичних фотографій людей і пейзажів за текстовими підказками до автоматизованої візуалізації розроблюваних проєктів і доповненої реальності.

Відповідно до цього, необхідність проведення дослідження зумовлена високим попитом на технології генерування зображень та швидким розвитком відповідних методів ШНМ. У даній роботі пропонується комплексне дослідження, що охоплює як теоретичний аналіз ключових сучасних підходів до генерації зображень (генеративно-змагальних та дифузійних мереж), так і їхню практичну програмну реалізацію та порівняльний аналіз. Основна ідея полягає у вивченні фундаментальних принципів роботи цих моделей через практичну реалізацію, дозволяючи сформулювати розуміння їхніх можливостей та обмежень.

Для виконання даної роботи визначено, що *об'єктом* дослідження виступає процес генерування інформаційних об'єктів, які представляють собою цифрові растрові зображення, за допомогою ШНМ.

Предметом дослідження є конкретні моделі генеративних мереж (ГМ) – генеративно-змагальних і дифузійних – для створення зображень, їхні основні механізми, алгоритми роботи, архітектурні особливості та фактори, які впливають на їхню ефективність та можливості практичної реалізації.

Мета роботи полягає в дослідженні та глибинному аналізі шляхів вирішення задачі генерування зображень на базі використання сучасних технологічних концепцій, алгоритмів і методів у сфері ШНМ, що втілюються моделями генеративних мереж, а також у вивченні актуальності та проблематики реалізації завдання на поточному рівні розвитку інформаційних наук.

Для досягнення поставленої мети було визначено такі основні завдання:

- проаналізувати постановку задачі генерування зображень та основи функціонування генеративних нейронних мереж;
- дослідити сучасний стан інформаційних технологій у галузі створення ШНМ для генерації зображень та виявити актуальні проблеми;
- провести теоретичне дослідження архітектур, принципів роботи та математичного апарату генеративних мереж;
- вивчити базові концепції генеративно-змагальних мереж (зокрема, моделей простої генеративно-змагальної та векторно-квантової мереж);
- вивчити теоретично явище дифузії та механізми роботи дифузійних моделей (імовірнісних дифузійних моделей з усуненням шуму);
- розробити програмні реалізації розглянутих моделей ГМ, включаючи етапи проєктування архітектур, вибору та підготовки наборів даних, налаштування обчислювального середовища, навчання та тестування роботи моделей;
- проаналізувати отримані результати, оцінити якість згенерованих зображень та порівняти ефективність досліджуваних моделей;
- здійснити оцінку доцільності використання ШНМ для вирішення задачі генерування зображень.

Основними методами досліджень, що використовуються в роботі, виступають теоретичний аналіз науково-технічної літератури для вивчення предметної області та існуючих рішень, методи системного аналізу для структурування інформації про архітектури та алгоритми ГМ, методи математичного моделювання для опису принципів їх роботи, експериментальне дослідження шляхом програмної реалізації та тестування моделей; інструменти порівняльного аналізу для оцінки результатів роботи.

У роботі досліджуються фундаментальні аспекти задачі генерування зображень, методологію побудови та навчання генеративних нейронних мереж, детально розглянуто архітектури, принципи функціонування та особливості генеративно-змагальних та дифузійних моделей, що використовуються для створення зображень.

Важливо, що актуальність вирішення задачі генерування зображень на базі використання ГМ визначається декількома факторами. По-перше, генерація зображень є однією з найпопулярніших тем у новітній галузі ШІ, що підтверджується величезною кількістю наукових публікацій та практичних розробок. По-друге, здатність створювати візуальний контент характеризується широким спектром потенційних застосувань у повсякденному житті інформаційного суспільства, охоплюючи творчі індустрії, розробку ігор та віртуальної реальності, медицину, наукові дослідження, а також покращення існуючих технологій обробки зображень. По-третє, незважаючи на значний прогрес в області генерування зображень, вкрай потрібним є виявлення нових можливостей для вдосконалення існуючих моделей і підвищення їх ефективності, таким чином роблячи дослідження у цій сфері надзвичайно важливими. Вивчення та порівняння різних підходів є необхідними для вибору оптимальних рішень для конкретних прикладних задач.

Наукова новизна роботи полягає у систематизації знань про сучасні ГМ на основі ШНМ, їхньому порівняльному аналізі на основі програмної реалізації та експериментального дослідження. Хоча окремі моделі добре висвітлені в літературі, комплексне дослідження, яке включає практичну реалізацію та порівняння декількох передових підходів у рамках однієї роботи, сприяє глибшому розумінню їх можливостей для вирішення задачі генерування зображень.

Отже, розробка та дослідження ШНМ для генерації зображень є новітньою та популярною тематикою у сучасній сфері інформаційних технологій і наук.

1 АНАЛІЗ ЗАДАЧІ ГЕНЕРУВАННЯ ЗОБРАЖЕНЬ У СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постанова задачі генерування зображень

Уявімо собі можливість, де завдяки натисканню кількох кнопок або введенням короткої текстової інформації можна миттєво створити абсолютно нове унікальне зображення, яке раніше не існувало в жодній формі. Цей, на перший погляд, простий приклад яскраво ілюструє фундаментальну суть задачі генерування цифрових об'єктів, зокрема зображень.

Задача генерації об'єктів (Object Generation Problem) у широкому сенсі є завданням, що полягає у створенні нових, правдоподібних та різноманітних об'єктів на основі аналізу та засвоєння закономірностей, присутніх у заданій навчальній вибірці [21]. Отримані об'єкти можуть бути використані як для прикладних цілей, так і для генерації об'єктів тренувальної вибірки, коли використовувати справжні дані неможливо. Залежно від того, з якою метою генеруються об'єкти, постановка задачі та методи її вирішення відрізняються.

Звісно, під час генерування об'єктів основне завдання в першу чергу полягає в тому, щоб навчитися створювати саме зображення, які людина не може відрізнити від реальних. При цьому основна мета такої задачі полягає в навчанні моделі розуміти глибинні закономірності, структуру та варіативність у великому наборі візуальних даних, а потім використовувати ці знання для синтезу нового контенту, вимагаючи не простого запам'ятовування навчальних прикладів, а й засвоєння абстрактних концепцій та їхніх візуальних проявів.

На найпростішому рівні задача генерування зображень уявляє проблему синтезу візуальних даних, що є дзеркальним відображенням завдань класичного комп'ютерного зору, оскільки працює у зворотному напрямку: вчиться створювати нові візуальні зразки з певного абстрактного представлення. Мета полягає в побудові алгоритму, здатного відтворювати зображення, які б максимально точно відповідали статистичним властивостям та візуальним характеристикам навчальної колекції або виходили за її межі, пропонуючи нові правдоподібні варіації.

Задачу генерування зображень можна класифікувати за типом вхідних даних та рівнем контролю над процесом генерації:

- безумовна генерація (unconditional generation) – генерування зображення, що відповідає загальному розподілу навчальних даних, без будь-яких специфічних вхідних умов, окрім випадкового вектору шуму (наприклад, з багатовимірного нормального розподілу), з метою отримання різноманітних і реалістичних зразків з вивченого домену (наприклад, обличчя людей, пейзажі або тварини);
- умовна генерація (conditional generation) – генерування зображень відбувається на основі певних вхідних умов або керуючих сигналів, апроксимуючи умовний розподіл;
- генерація за атрибутами/класами (attribute/class-conditional generation) – створення зображень, які належать до певного класу або мають певні дискретні чи неперервні атрибути;
- генерація за макетом або сегментаційною картою (layout/segmentation-map-to-image synthesis) – синтез зображення на основі заданої просторової структури або семантичної карти, що визначає розташування та типи об'єктів;
- інтерактивна генерація (interactive image generation) – на процес генерації можна впливати, надаючи зворотний зв'язок або модифікуючи керуючі сигнали.

Таким чином, розуміння сутності задачі генерування зображень надає змогу усвідомити головні аспекти, що необхідно враховувати при створенні та розробці ГМ для вирішення поставленої проблеми.

1.2 Основи генеративних мереж. Особливості їх створення та розробки

Відповідно до 1.1, фундаментальними підходами та інструментами, що лежать в основі процесу генерування зображень є генеративні мережі.

ГМ є класом алгоритмів машинного навчання, розроблених для вивчення розподілу ймовірностей вхідних даних з метою подальшого синтезу нових зразків, що належать до цього розподілу [13]. Загальна концепція полягає у навчанні моделі G (генератора) відображати деякий вхідний випадковий шум z (зазвичай з простого

апріорного розподілу – гаусівського або рівномірного) у простір зображень X . В ідеальному випадку, розподіл $P_g(x)$ згенерованих даних має бути якомога ближчим до реального розподілу даних $P_{data}(x)$:

$$P_{data}(x) \sim P_g(x), \quad (1.1)$$

де $P_g(x)$ – розподіл, що моделюється.

Центральним елементом багатьох сучасних підходів створення генеративних моделей є концепція латентного простору Z (latent space) – уявний внутрішній набір стислих ознак, який утворюється в результаті навчання генеративної НМ. Це, як правило, багатовимірний векторний простір значно меншої розмірності, ніж простір зображень X . Припускається, що кожна точка (або вектор), $z \in Z$, у цьому просторі кодує певні семантичні атрибути, характеристики або абстрактні риси майбутнього зображення.

Генеративна модель G виступає як детермінована (або стохастична) функція, яка реалізує відображення з латентного простору в простір зображень:

$$G: Z \rightarrow X \quad (1.2)$$

Враховуючи функціональний аспект наявності латентного простору, процес генерування нового зображення x' реалізується наступним алгоритмом:

- 1) здійснюється вибірка випадкового вектору z з деякого апріорного, зазвичай простого, розподілу $P(z)$ у латентному просторі, за рахунок використання багатовимірного нормального розподілу $N(0, I)$ або рівномірного розподілу $U(-1, 1)^d$, де d – розмірність латентного простору;
- 2) обраний латентний вектор z подається на вхід генеративної мережі G ;
- 3) мережа G (представляє глибоку нейронну мережу – зі згортковою або трансформерною архітектурою), перетворює вектор z у тензор зображення:

$$x' = G(z) \quad (1.3)$$

Ключова складність у втіленні ГМ полягає в навчанні параметрів (ваг) θ таким чином, щоб розподіл згенерованих зображень $P_g(x, \theta)$ якомога точніше відтворював розподіл реальних даних $P_{data}(X)$.

Залежно від специфіки цільового застосування та постановки задачі генерування зображень, методи та інструменти її вирішення можуть суттєво відрізнятися, вимагаючи різних моделей ШНМ.

Існує декілька основних парадигм та архітектур ГМ для створення зображень: генеративно-змагальні мережі, варіаційні автокодувальники, дифузійні моделі, авторегресійні моделі та моделі на основі потоків.

Генеративно-змагальні мережі (Generative Adversarial Networks – GAN), запропоновані Я. Гудфеллоу у 2014 році та методологія яких представлена у працях [12], складаються з двох нейронних мереж – генератора та дискримінатора, які навчаються у змагальному процесі. Генератор намагається створити реалістичні дані (зображення) з випадкового шуму, а дискримінатор старається відрізнити справжні дані з навчальної вибірки від згенерованих. Важливо, що обидві мережі навчаються одночасно, при цьому генератор прагне «обдурити» дискримінатор, а дискримінатор – стати кращим у виявленні підробок. Таким чином, з часом генератор навчається створювати все більш правдоподібні зображення.

Варіаційні автокодувальники (Variational Autoencoders – VAE), представлені Д. Кінгмою та М. Веллінгом у 2013 р. та викладені в роботі [16], є генеративними моделями, що базуються на принципах варіаційного виведення. VAE складається з кодувальника, який відображає вхідні дані у латентний простір, та декодувальника, що відновлює дані з латентного представлення. На відміну від стандартних автокодувальників, VAE навчається не просто відтворювати вхід, а моделювати розподіл ймовірностей у латентному просторі, дозволяючи генерувати нові дані шляхом семплювання розподілу та подальшого декодування.

Дифузійні моделі (Diffusion Models – DM), концепція та дослідження яких викладені в працях [8], представляють клас моделей, що працюють за принципом поступового додавання шуму до даних та подальшого навчання моделі обернути цей процес за рахунок поступового видалення шуму для генерування зображення з

повністю зашумленого стану. DM також відомі своєю здатністю генерувати високоякісні та різноманітні зображення, хоча процес генерації (семплювання) може бути обчислювально інтенсивним через велику кількість кроків.

Авторегресійні моделі (Autoregressive Models – AM), представлені у роботах [5, 27], генерують зображення за принципом «піксель за пікселем», моделюючи умовний розподіл кожного наступного пікселя на основі всіх попередньо згенерованих. AM засновані на ланцюговому правилі теорії ймовірностей і можуть явно обчислювати правдоподібність згенерованих даних. Звісно, хоча моделі й здатні генерувати дуже деталізовані зображення, процес генерації є послідовним і, як наслідок, повільним, особливо для зображень високої роздільної здатності.

Моделі на основі потоків (Flow-based Models), технологічний механізм яких був наведений у працях [20], використовують послідовність обернених перетворень (потоків) для відображення складного розподілу даних на простий апіорний розподіл. Завдяки оберненості перетворень, моделі дозволяють точно обчислювати правдоподібність даних та ефективно семплювати, а також пропонують добре структурований латентний простір.

Розглядаючи особливості створення та розробки наведених генеративних мереж, основними етапами виступають збір і підготовка даних, проєктування архітектури моделі та процес навчання.

Етап збору та підготовки даних зумовлюється тим, що якість та різноманітність навчальних даних є критично важливими для успіху реалізації будь-якої ГМ. Дані мають бути репрезентативними для домену, який модель має вивчити, причому алгоритм підготовки включає очищення даних (видалення артефактів і дублікатів), нормалізацію значень пікселів, зміну розміру зображень до єдиного стандарту та аугментацію даних.

Етап проєктування заснований на підборі необхідної архітектури моделі залежно від типу ГМ та специфіки вирішуваної задачі. На даному кроці втілення важливими є вибір функцій активації, методів нормалізації, механізмів уваги та визначення функції втрат, яка керує процесом навчання.

Етап навчання є складнішим і часто нестабільним процесом, що вимагає ретельного налаштування гіперпараметрів, таких як швидкість навчання, розмір оброблюваного батчу і параметри оптимізатора.

Сама процедура розробки ГМ є ітеративною, включаючи експерименти з архітектурами, функціями втрат, гіперпараметрами, а також аналіз згенерованих результатів на кожному етапі та поступове покращення моделі.

1.3 Аналіз сучасного становища сфери інформаційних технологій у галузі створення ШНМ для генерування зображень

Вирішення задачі генерування зображень за допомогою ШІ є одним з провідних та найбільш досліджуваних завдань у сучасній сфері інформаційних технологій, переживаючи період стрімкого розвитку. Технології, які лежать в основі цього прогресу, зокрема ШНМ, відкривають нові горизонти не тільки в інформаційних науках, але й в мистецтві, дизайні, індустрії розваг, наукових дослідженнях та багатьох інших сферах життя суспільства.

Сучасні генеративні моделі досягли вражаючої якості синтезованих зображень. Цей прогрес зумовлений як розробкою нових архітектур ШНМ, так і вдосконаленням алгоритмів навчання та доступом до великих наборів даних, представлених різноманітними наборами для навчання моделей.

Здатність генерувати фотореалістичні та стилізовані зображення на основі текстових описів, ескізів або навіть випадкового шуму відкриває низку новітніх можливостей для творчості та автоматизації. Саме тому розробники та науковці активно досліджують нові методи для покращення контрольованості, різноманітності та реалістичності генерованих зображень, а також для зменшення обчислювальних витрат, пов'язаних з навчанням та використанням цих моделей.

Одними з основних сучасних фундаментальних підходів щодо генерування зображень на базі ШНМ стали VAE, основна концепція яких була представлена в роботі Ц. Лю [17] як клас ГМ, що навчаються компактному представленню даних у латентному просторі та подальшій генерації нових зразків з нього.

Основою VAE є двокомпонентна архітектура з кодувальної та декодувальної частин. Кодувальник відображає вхідні зображення у латентний простір, який має ймовірнісну природу. Декодувальник, у свою чергу, намагається відновити вихідні дані з векторів, отриманих з латентного розподілу.

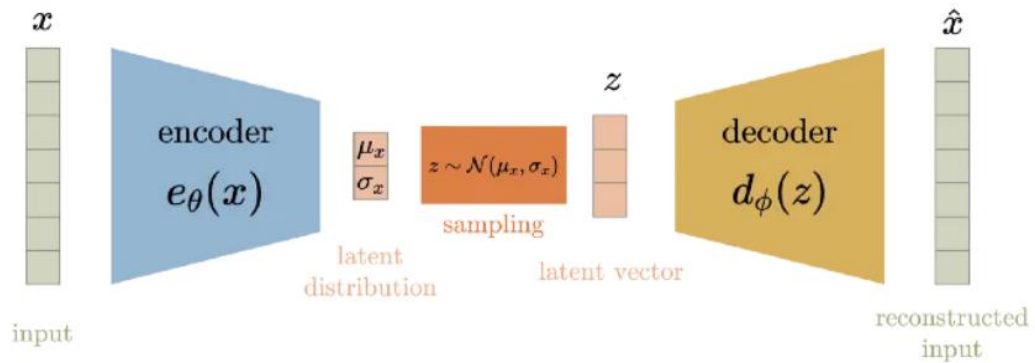


Рисунок 1.1 – Концептуальне представлення архітектури VAE [16]

Навчання VAE відбувається шляхом оптимізації так званої варіаційної нижньої межі – Evidence Lower Bound (ELBO) [17]. ELBO складається з двох членів: перший відповідає за точність реконструкції вхідних даних, а другий забезпечує відповідність розподілу латентних змінних до апіорного розподілу. Це сприяє формуванню гладкого латентного простору, придатного для генерації нових зразків. Звісно, одним з відомих обмежень ранніх VAE була тенденція до генерації розмитих зображень, що пов'язано з природою функції втрат, яка заохочує усереднення для мінімізації помилки реконструкції.

Для подолання деяких обмежень VAE були запропоновані векторні квантово-варіаційні автокодувальники (VQ-VAE) [28]. Ключовою ідеєю VQ-VAE є використання дискретного латентного простору, представленого у вигляді словника кодів (codebook) – набору вивчених векторів-ембедингів. Кодувальник відображає вхідні дані в неперервні вектори, які квантуються шляхом заміни на найближчий вектор зі словника кодів. Даний підхід допомагає уникнути «колапсу апостеріорного розподілу», коли латентні змінні ігноруються. Функція втрат VQ-VAE включає члени реконструкції, оновлення словника кодів та «зобов'язання», заохочуючи виходи кодувальника бути близькими до обраних векторів словника.

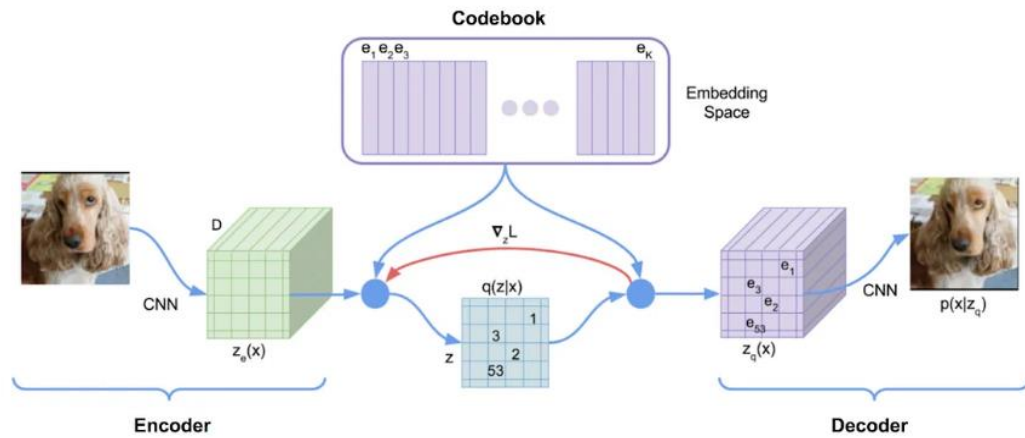


Рисунок 1.2 – Концептуальне представлення архітектури VQ-VAE [28]

Нещодавною розробкою для покращення VQ-VAE став метод «rotation trick» [23], який спрямований на покращення поширення градієнтів через недиференційовний шар квантування шляхом перетворення виходів кодувальника у відповідні вектори словника завдяки обертанню та масштабуванню, покращуючи якість реконструкції та ефективність використання словника кодів.

З метою вирішення задачі генерування зображень, головною концепцією для вивчення також стали GAN, які представляють клас ГМ, відомих своєю здатністю створювати високореалістичні зображення [2, 12].

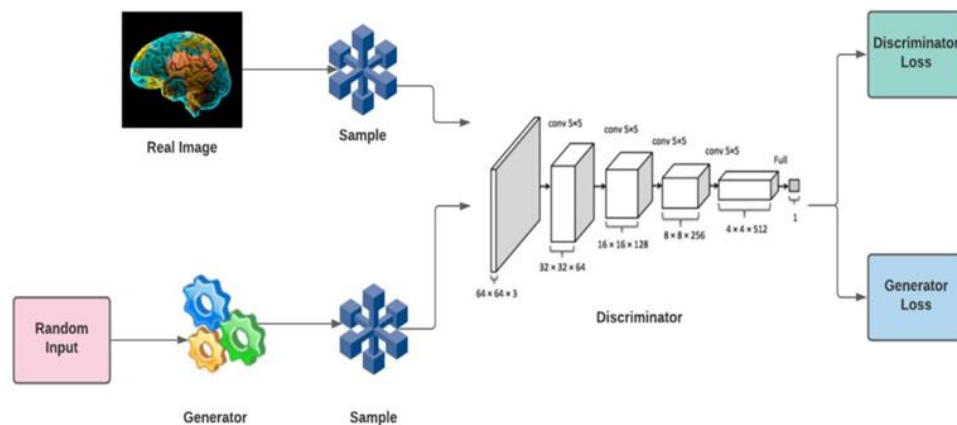


Рисунок 1.3 – Концептуальне представлення архітектури GAN [2]

Однією з перших успішних архітектур GAN стала DCGAN (Deep Convolutional GAN), що використовувала згорткові шари без шарів пулінгу, замінюючи їх на згортки із кроком у дискримінаторі та дробово-крокові згортки у

генераторі. DCGAN також активно використовувала пакетну нормалізацію для стабілізації навчання [22].

Покращенням DCGAN стала архітектура BigGAN [29], яка продемонструвала можливість масштабування GAN для генерування зображень високої роздільної здатності та якості на складних наборах даних. Ключовими елементами BigGAN є використання більших за розміром батчів, збільшення кількості параметрів моделі, застосування шарів самоуваги для захоплення глобальних залежностей, умовна пакетна нормалізація для інтеграції інформації про клас, та усічення латентного простору для покращення якості зразків за рахунок зменшення різноманітності.

Особливо важливою в сучасному розвитку стала лінійка моделей StyleGAN, StyleGAN2 і StyleGAN3 від NVIDIA [7], які внесли значний вклад у генерацію високоякісних зображень, особливо облич.

StyleGAN запровадив концепцію прогресивного зростання (початок генерування з низької роздільної здатності та її поступове збільшення) та мережу відображення для перетворення латентного коду на проміжний латентний код, який керує стилем на різних рівнях генератора через адаптивну нормалізацію екземплярів (AdaIN), дозволяючи краще виявляти атрибути зображень.

StyleGAN2 усунув деякі артефакти, присутні в StyleGAN, зокрема, краплеподібні артефакти, шляхом модифікації AdaIN на так звану вагову демодуляцію. Також відкидалася ідея прогресивного зростання на користь архітектури зі зв'язками пропуску або залишковими блоками.

StyleGAN3 був розроблений для усунення проблеми «прилипання текстур», коли текстури прив'язані до координат екрану, а не до поверхонь об'єктів, шляхом забезпечення еквіваріантності до зсувів та обертань через використання Фур'є ознак на вхідному шарі генератора та спеціально розроблених фільтрів.

Новітньою розробкою є впровадження моделей векторно-квантових мереж VQ-GAN [10, 30], що поєднує переваги дискретних латентних представлень VQ-VAE з архітектурою GAN. VQ-GAN використовує автокодувальник для вивчення словника візуальних компонентів, а потім навчається генерувати реалістичні зображення на основі цих компонентів.

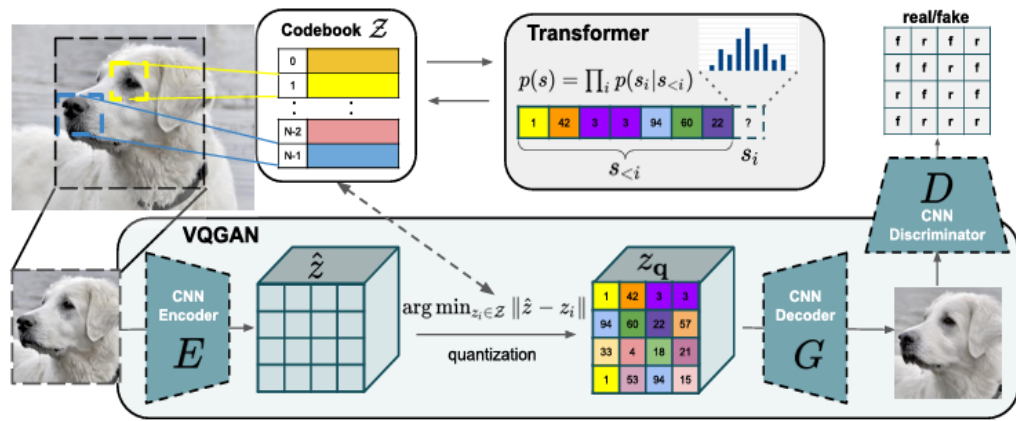


Рисунок 1.4 – Концептуальне представлення архітектури VQ-GAN [10]

Основними напрямками стали мережі ViT-VQGAN, SGC-VQGAN, Dual Codebook VQGAN та RAQ-VAE.

Архітектура ViT-VQGAN [30] замінює традиційні згорткові нейронні мережі (CNN) в автокодувальнику на Vision Transformers (ViT). Це дозволяє покращити обчислювальну ефективність на прискорювачах та якість реконструкції зображень. ViT-VQGAN також використовує вдосконалені методи навчання словника кодів, такі як факторні коди, передбачаючи проекцію виходів кодувальника в низьковимірний простір для пошуку індексів кодів, та L2-нормалізацію кодів.

Модель SGC-VQGAN (Semantic Guided Clustering VQGAN) [24] використовує семантичну інформацію, отриману з моделей сегментації, для керування процесом кластеризації словника кодів. Крім того, SGC-VQGAN застосовує конвеєр Pyramid Feature Learning для одночасного захоплення як дрібних деталей зображення, так і його семантичного змісту.

Підхід Dual Codebook VQGAN [9] пропонує розділення латентного представлення на глобальні та локальні компоненти, кожен з яких має власний словник кодів та механізм оновлення, причому глобальний словник може оновлюватися за допомогою легковагового трансформера.

Розробка RAQ-VAE (Rate-Adaptive Quantization VQ-VAE) [23] спрямована на адаптивне стиснення зображень, шляхом використання змінних бітрейтів словника кодів без необхідності повного перенавчання моделі. Це досягається за допомогою Seq2Seq моделей та спеціальних процедур навчання, таких як cross-forcing.

Крім того, комбінація VQ-GAN з авторегресійними трансформерами (наприклад, архітектури типу GPT-2) стали потужним підходом для умовного синтезу зображень. VQ-GAN створює дискретне представлення у вигляді послідовності візуальних токенів (індексів зі словника кодів), а потім трансформер навчається моделювати розподіл цих токенів, надаючи можливість генерувати нові зображення, обумовлені текстовими описами або семантичними картами [30].

Окрім VAE та GAN, провідною технологією в генерації зображень стали дифузійні моделі (DM), демонструючи здатність створювати високоякісні та різноманітні результати, часто перевершуючи наведені вище підходи.

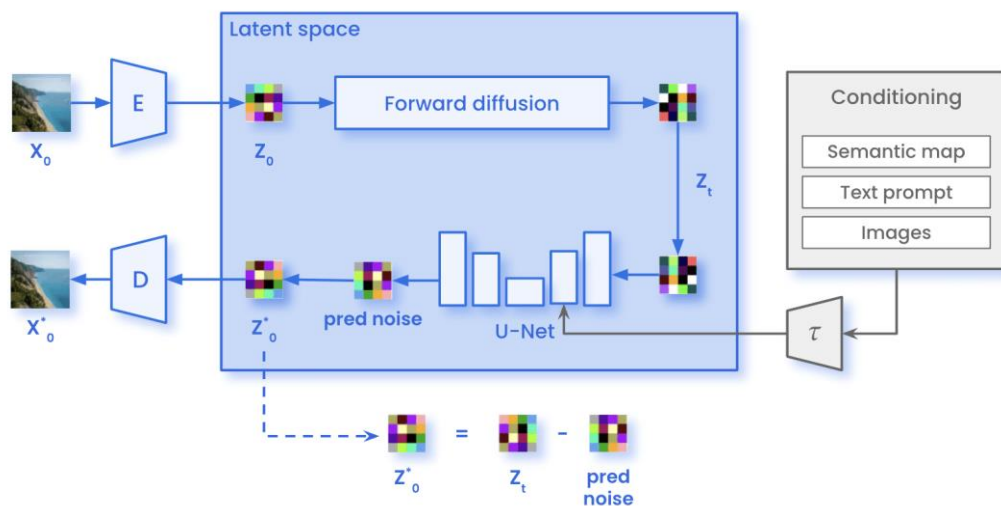


Рисунок 1.5 – Концептуальне представлення архітектури DM [19]

Основними сучасними концепціями DM є впровадження імовірнісних дифузійних моделей з усуненням шуму та моделей на основі балів.

Підхід DDPM (Denoising Diffusion Probabilistic Models) [15] базується на двох процесах – прямому (поступове додавання шуму до вихідного зображення протягом фіксованої кількості кроків) та зворотному (навчання НМ з метою покрокового усунення шуму, починаючи з чистого шуму).

Score-based Models [28] розглядає дифузійний процес у неперервному часі, описуючи його за допомогою стохастичних диференціальних рівнянь (SDE) для прямого процесу (зашумлення) та відповідних обернених SDE для процесу генерації (знешумлення). Ключовим елементом є функція оцінки, яка є градієнтом

логарифму щільності ймовірності зашумлених даних на визначеному часовому кроці. Процес генерування може відбуватися шляхом розв'язання оберненого SDE або відповідного звичайного диференціального рівняння (ODE), що має ті ж самі маргінальні розподіли. Навчання часто базується на методі зіставлення оцінок з усуненням шуму.

Проблемою традиційних DM є велика кількість кроків для генерування якісних зображень, роблячи сам процес вкрай повільним. Відповідно до цього, значні зусилля були спрямовані на прискорення семплінгу за допомогою неявних дифузійних моделей з усуненням шуму.

DDIM (Denoising Diffusion Implicit Models) [25] узагальнює DDPM, вводячи немарковські процеси дифузії, що дозволяє використовувати потенційно детерміністичну та значно швидшу модель. Головною властивістю є послідовна генерація з одного й того ж латентного вектора з різною кількістю кроків, надаючи схожі високорівневі ознаки для інтерполяції зображень.

Крім того, з метою прискорення семплінгу, були розроблені підходи Consistency Models та Rectified Flow.

Consistency Models [4] навчаються безпосередньо відображати точки вздовж траєкторії ODE на її початок, надаючи можливість генерувати високоякісні зображення за один або декілька кроків. Тренування реалізується шляхом дистиляції з попередньо навчених дифузійних моделей (Consistency Distillation) або з нуля (Consistency Training).

Rectified Flow [11] спрямований на навчання ГМ, де траєкторії від шуму до даних є прямими лініями, що описуються ODE, у просторі ознак. Це дозволяє значно зменшити кількість кроків, необхідних для генерації, порівняно з традиційними DM, які можуть мати більш викривлені траєкторії. Даний алгоритм використовується для ітеративного «випрямлення» потоків.

Важливим напрямком вивчення стали латентні дифузійні моделі (LDM) [14], що впроваджують двоетапний підхід, де спочатку автокодувальник навчається стискати зображення у низьковимірний латентний простір, зберігаючи перцептивно важливу інформацію, а потім дифузійна модель навчається генерувати

дані в цьому стисненому латентному просторі. Таким чином зменшуються обчислювальні витрати, оскільки дифузійний процес відбувається в просторі меншої розмірності. LDM також використовують механізм cross-attention для гнучкого кондиціонування на різні модальності, такі як текст або семантичні карти, даючи змогу створювати моделі для text-to-image та layout-to-image синтезу.

Отже, проведений аналіз сучасних технологій створення генеративних мереж виявляє низку ключових тенденцій щодо необхідності й доцільності дослідження вирішення задачі генерування зображень, використовуючи інструменти ІІІ.

1.4 Дослідження проблематики створення генеративних мереж

Незважаючи на досягнуті успіхи у галузі створення ГМ, розглянутими в 1.3, існує велика кількість невирішених проблеми, пов'язаних зі стабільністю навчання, контрольованістю процесу генерування, високими обчислювальними вимогами та оцінкою якості згенерованих зображень.

У першу чергу основна проблематика полягає у здійсненні процесу навчання, який вимагає ретельного налаштування гіперпараметрів. Якщо розглядати GAN моделі основна складність характеризується забезпеченням стабільності навчання. Наприклад, генератор може почати продукувати лише обмежений набір схожих зразків, не охоплюючи всього різноманіття даних з навчальної вибірки (mode collapse), або ж дискримінатор може стати занадто сильним і не надавати корисного градієнта генератору (vanishing gradients).

Крім того, навчання сучасних ГМ, особливо для зображень високої роздільної здатності, потребує значних обчислювальних потужностей, зокрема графічних (GPU) або тензорних (TPU) процесорів з великим обсягом пам'яті, а сам процес може займати значний проміжок часу, залежно від розміру моделі, обсягу даних та доступних обчислювальних ресурсів.

Важливо, що специфічні особливості та виклики у розробці ГМ можуть полягати у встановленні балансу між якістю та різноманітністю, коли часто існує дилема між генерацією якісних, але менш різноманітних зображень, і навпаки.

Вагомою є проблематика забезпечення можливості керування атрибутами згенерованих зображень та масштабування до високої роздільної здатності, вимагаючи прогресивних технік навчання або багатоетапних підходів. Хоча умовна генерація значно просунулася, особливо з дифузійними моделями, досягнення тонкого контролю над усіма бажаними аспектами зображення залишається складним завданням. Також, не варто забувати про етичні аспекти, пов'язані з можливістю створення дипфейків, вимагаючи розробки відповідних механізмів контролю та протидії.

Таким чином, вирішення розглянутих комплексних проблем є першочерговою задачею, яка виникає в процесі створення та розробки ШНМ для вирішення задач генерування зображень.

1.5 Огляд існуючих програмних рішень для вирішення задачі генерування зображень

Теоретичні досягнення в галузі ГМ знаходять своє втілення у практичних програмних рішеннях (ПР), які варіюються від потужних відкритих інструментів та бібліотек до комерційних платформ для створення зображень. Існуючі програмні продукти (ПП) відіграють ключову роль у демократизації доступу до технологій генерації зображень, сприяючи інноваціям і швидкому поширенню нових методів і забезпечуючи зручність використання та високу якість результатів.

Stable Diffusion [26] є одним із найвпливовіших ПР, розроблене компанією Stability AI. Дана модель базується на LDM [14], дозволяючи генерувати високоякісні зображення з текстових описів, а також виконувати перетворення одного зображення на інше за текстовою підказкою, заповнювати відсутні частини зображення та виконувати його розширення.

Ключовими перевагами Stable Diffusion є представлення моделі та її ваги як «open source», надаючи можливість для вільного використання, модифікування та розширення. Також модель характеризується високою якістю генерування, а завдяки роботі в латентному просторі може працювати на споживчих GPU, потребуючи відносно низькі обчислювальні вимоги.

Midjourney [18] постає однією з найвідоміших та найпопулярніших комерційних платформ для генерування зображень з текстових описів, доступ до якої здійснюється переважно через бота в Discord.

Особливостями ГП є висока художня якість, представлена естетичними, сюрреалістичними й деталізованими зображеннями, а простота використання інтерфейсу та висока швидкість розвитку є головними перевагами, оскільки модель постійно оновлюється, пропонуючи покращену якість, нові стилі та функції.

Одним із провідних ГП для вирішення задачі генерування зображень також є модель DALL-E 3 [6], розроблена компанією OpenAI та є наступником попередніх версій DALL-E. DALL-E 3 відома своєю здатністю дуже точно інтерпретувати складні текстові запити та генерувати зображення з високим ступенем відповідності опису, включаючи дрібні деталі та взаємодію об'єктів. Важливим аспектом є обмеження генерації шкідливого, упередженого або захищеного авторським правом контенту.

Adobe Firefly [1] є ГМ від Adobe. Основна характеристика – навчання на базі Adobe Stock (відкритий ліцензований контент і контенті суспільного надбання), роблячи модель комерційно безпечною для використання. Також дозволяється використовувати генеративні функції моделі безпосередньо в інструментах для дизайну, інтегруючись у робочі процеси та надаючи великий спектр користувальницьких функцій (наприклад, генерація зображень з тексту, текстові ефекти, генеративне заповнення та генеративна зміна кольору).

Таким чином, можна підсумувати, що створення та розробка ГМ є динамічною галуззю, яка поєднує глибокі теоретичні знання з практичними програмними навичками, де кожен тип моделі пропонує унікальний набір інструментів та підходів для вирішення задачі синтезу даних, і вибір конкретного методу залежить від специфічних вимог задачі, наявних даних та ресурсів.

2 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРОЄКТУВАННЯ ГЕНЕРАТИВНИХ МЕРЕЖ

2.1 Аналіз вибору моделей генеративних мереж для втілення

Вибір конкретних моделей ГМ для теоретичного дослідження в рамках даної роботи є ключовим етапом, адже надає змогу вивчити базові алгоритми, архітектури та принципи роботи, а також спроектувати низку ГМ з метою подальшої практичної реалізації та аналізу отриманих результатів.

Для виконання дослідження було обрано моделі генеративно-змагальних і дифузійних мереж – просту генеративно-змагальну мережу (Generative Adversarial Nets – GAN), векторно-квантову генеративно-змагальну мережу (Vector Quantized Generative Adversarial Network – VQ-GAN) та неявну дифузійну модель з усуненням шуму (Denoising Diffusion Implicit Model – DDIM).

Обраний набір моделей представляє собою збалансований спектр підходів, що охоплює як класичні, так і передові технології для вирішення проблематики генерування зображень. Це дозволяє провести комплексне дослідження, порівняти різні парадигми та зробити обґрунтовані висновки щодо практичності й доцільності використання ШНМ для генерування зображень.

Проста модель GAN є однією з основоположних архітектур у сфері генерації зображень, розуміння принципу роботи якої є критично важливим аспектом для подальшого створення вдосконалених механізмів ГМ, дозволяючи кількісно та якісно оцінити прогрес у галузі.

Гібридний підхід VQ-GAN поєднує ідеї векторного квантування дискретних латентних представлень варіаційних автокодувальників (VQ-VAE) з потужністю змагального навчання звичайної GAN моделі, що надає змогу вивчити дискретні латентні представлення (візуальні токени) високої якості.

DDIM представляє прискорену дифузійну модель, яка спрямована на вирішення проблеми повільного процесу семплінгу, що в свою чергу значно пришвидшує процедуру генерування зображень для реальних практичних застосувань.

Обраний набір моделей забезпечує комплексне охоплення ключових парадигм генерації зображень, надаючи можливість провести аналіз етапів створення, дослідити фундаментальні принципи та оцінити ефективність ГМ практично. Порівняння класичного підходу GAN з новітньою дифузійною моделлю DDIM, а також дослідження гібридного підходу VQ-GAN, надасть змогу вивчити основні концепції вирішення задачі генерування зображень за допомогою підходів ІІІ та НМ.

2.2 Генеративно-змагальні мережі

2.2.1 Загальні концепції моделей генеративно-змагальних мереж

Як відомо з 1.2-1.3., генеративно-змагальна мережа (ГЗМ) є класом алгоритмів машинного навчання, виступаючи основою для генеративного ІІІ та вирішення задачі генерування зображень. В ядрі ГМЗ лежить ідея змагального навчання двох нейронних мереж – генератора та дискримінатора.

Генератор має на меті навчитися створювати дані, які максимально схожі на реальні. Він приймає на вхід випадковий шум (зазвичай, вектор з простого розподілу) і перетворює його на зображення. Головне завдання – «обдурити» дискримінатор, змусивши його повірити, що згенерований зразок є справжнім.

Дискримінатор, навпаки, навчається відрізняти справжні дані від створених генератором, отримуючи на вхід зразок і видаючи ймовірність того, що він є справжнім, а не згенерованим.

Мета навчання ГЗМ полягає в підвищенні рівня помилок дискримінантної мережі, а сам процес навчання [12] відбувається як гра з нульовою сумою («*minimax game*») між генератором (G) та дискримінатором (D), які конкурують між собою у формі, де виграв один агент є програшем іншого.

Кожен простір ймовірностей (Ω, μ_{ref}) визначає гру ГЗМ, де стратегією генератора є $P(\Omega)$ – множина всіх ймовірнісних мір Ω , а стратегією дискримінатора – множина марковських ядер:

$$\mu_D: \Omega \rightarrow P[0, 1], \quad (2.1)$$

де $P[0, 1]$ – множина ймовірнісних мір на $[0, 1]$.

Гра з нульовою сумою відбувається за наступною цільовою функцією:

$$L(\mu_G, \mu_D) := E_{x \sim \mu_{ref}, y \sim \mu_D(x)} [\ln(y)] + E_{x \sim \mu_G, y \sim \mu_D(x)} [\ln(1 - y)] \quad (2.2)$$

Завдання генератора полягає в тому, щоб наблизитися до $\mu_G \approx \mu_{ref}$ (мінімізувати цільову функцію), максимально наближуючи власний розподіл виходу до еталонного розподілу. Завдання дискримінатора – виводити значення близьке до 1 (максимізувати цільову функцію), коли вхідні дані мають еталонний розподіл, і близьке до 0, якщо вхідні дані мають вигляд генераторного розподілу.

Концептуально важливо, що дискримінатору подаються як справжні зразки з навчального набору, так і згенеровані генератором до тих пір, поки він не досягне прийнятної точності, навчаючись максимізувати її у розрізненні двох типів даних. Генератор, у свою чергу, заповнюється випадковими вхідними даними, які вибираються із заздалегідь визначеного латентного простору, та навчається мінімізувати ймовірність правильної класифікації дискримінатором згенерованих зразки як «підроблені», максимізуючи його помилку щодо власних виходів.

Незалежні процедури зворотного поширення застосовуються до обох мереж таким чином, що генератор створює кращі зразки, тоді як дискримінатор стає більш досвідченим у позначенні синтезованих зразків.

Процес навчання ГЗМ відбувається ітеративно: на кожному кроці спочатку оновлюються ваги дискримінатора, а потім – ваги генератора (або навпаки, в залежності від конкретної реалізації ГЗМ). Теоретично, цей процес має привести до стану рівноваги, коли генератор створює настільки реалістичні дані, що дискримінатор не може відрізнити їх від справжніх.

У загальному, конкуренція між двома НМ спонукає їх взаємно вдосконалюються, у кінцевому підсумку призводячи до здатності генератора створювати високоякісні та реалістичні зразки даних.

2.2.2 Модель простої генеративно-змагальної мережі (GAN)

Розуміючи базові принципи роботи ГЗМ, виконаємо проектування архітектури простої генеративно-змагальної мережі GAN, яка складатиметься з двох компонентів – генератора та дискримінатора.

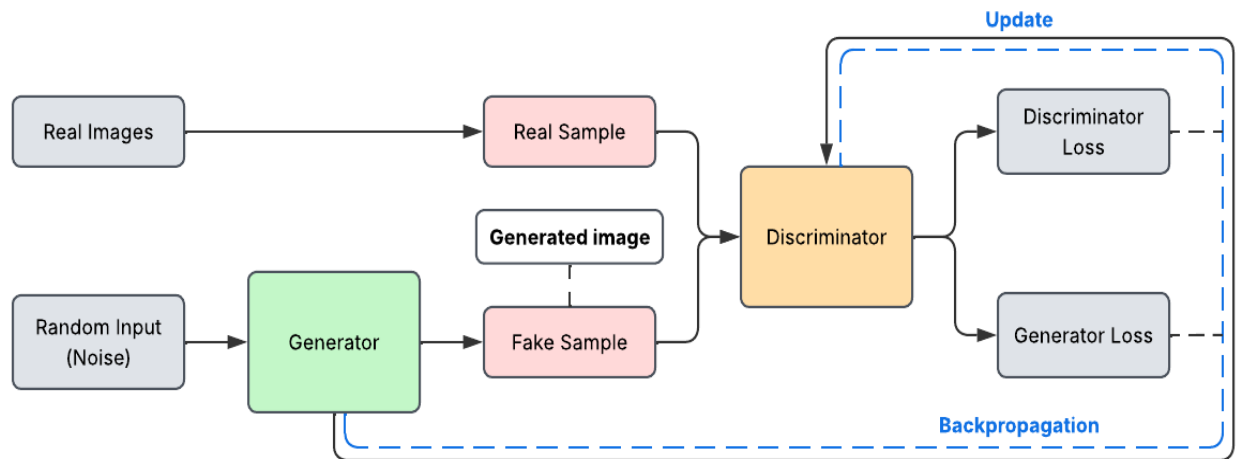


Рисунок 2.1 – Схематичне представлення проектованої простої моделі GAN

У своїй найпростішій формі GAN прийматиме на вході випадковий шум, який генератор перетворюватиме на осмислений вихід. Вводячи шум, можна змусити модель виробляти різноманітні дані, беручи вибірки з різних місць цільового розподілу. Для зручності, простір, з якого вибирається шум, матиме меншу розмірність, ніж розмірність вихідного.

Метою генератора є створення зразків фальшивих даних, змушуючи дискримінатор класифікувати їх як справжні. Саме тому побудову доцільно представити глибокою НМ, яка використовуватиме випадковий шум як вхідний сигнал для створення реалістичних зразків зображень та вивчатиме базовий розподіл даних, налаштовуючи свої параметри шляхом зворотного поширення.

Крім того, мережа генератора має перетворювати вхідний випадковий латентний вектор на зображення. Даний механізм можна реалізувати за допомогою згортки, пакетної нормалізації та підвищеної дискретизації шарі, причому на виході доцільно використати функцію активації, яка гарантуватиме, що значення пікселів знаходяться в межах очікуваного діапазону.

Дискримінатор, у свою чергу, діє як бінарний класифікатор, розрізняючи реальні та згенеровані дані. Він покращуватиме свою класифікаційну здатність шляхом навчання та уточнення параметрів, з метою точнішого виявлення фальшивих зразків.

Мережева архітектура дискримінатора має відповідати типу даних, що класифікуватимуться. Відповідно, при роботі з зображеннями необхідно використовувати згорткові шари, які допомагатимуть виділити ознаки та покращити здатність моделі розрізняти реальні та згенеровані зразки. Крім того, структура повинна мати шари пакетної нормалізації, відсіювання та активації для покращення стабільності навчання.

Отже, у загальному випадку генератор можна представити звичайною деконволюційною нейронною мережею, а дискримінатор – згортковою.

Також важливо, що навчальні дані для дискримінатора мають надходити з двох джерел – як реальні дані, в якості позитивних прикладів, так і як створені генератором, слугуючи негативними прикладами протягом навчання. Таким чином, вихід генератора безпосередньо має бути під'єднано до входу дискримінатора, де завдяки зворотному поширенню класифікація дискримінатора буде надавати сигнал, який генератор використовуватиме для оновлення своїх ваг.

Не слід забувати, що GAN намагатиметься відтворити розподіл ймовірностей, тому проєктована модель повинна використовувати дві функції втрат – окремо для навчання генератора та дискримінатора – які відображатимуть відстань між розподілом згенерованих і реальних даних.

2.2.3 Модель векторно-квантової мережі (VQ-GAN)

Векторно-квантова генеративно-змагальна мережа (VQ-GAN) є гібридним підходом, який поєднує принципи векторного квантування моделей типу Vector Quantised-Variational AutoEncoder (VQ-VAE) зі змагальним навчанням ГЗМ.

Класична архітектура GAN, яка була описана в 2.2.1, складається з двох основних компонентів – генератора та дискримінатора. Концепція VQ-GAN

вносить суттєві зміни до цієї базової побудови, використовуючи структурований підхід, який включає елементи автокодувальника з дискретним латентним простором, сформованого з кодового словника завдяки квантуванню.

На відміну від прямого процесу генерування зображень з шуму, притаманний моделям GAN, мережа VQ-GAN втілює двоетапний підхід за допомогою автокодувальника, механізму квантування, дискримінатора та додаткової мережі трансформера, яка створюватиме випадкові зображення.

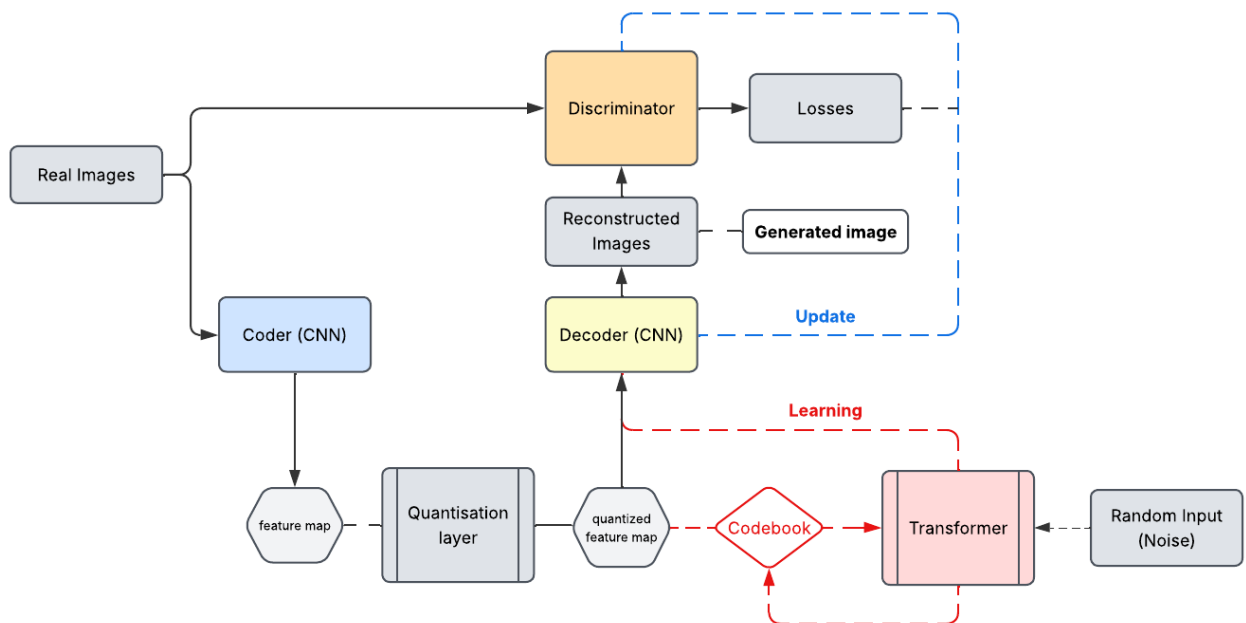


Рисунок 2.2 – Схематичне представлення проєктованої моделі VQ-GAN

Для побудови архітектури втілюваної моделі, замість генеративної частини використаємо автокодувальник, представлений парою кодувальника й декодувальника, між якими знаходитиметься шар квантування.

На даному етапі важливо впровадити механізм латентного простору, що представляє собою дискретний словник кодів (codebook), кожен вектор (ембедінг) у якому відтворює певний візуальний «атом» чи «патерн».

Реалізується й сам процес векторного квантування (vector quantization), де неперервне латентне представлення відображається на найближчий вектор з навченого дискретного словника кодів, а результатом є дискретне представлення зображення у вигляді послідовності індексів (токенів) цього словника.

З огляду на це, проєктована модель повинна містити кодувальник, який приймає вхідне зображення та стискає його, перетворюючи у компактне неперервне латентне представлення та відображаючи у набір індексів векторів кодової книги, і декодувальник, що відновлюватиме зображення з квантованого латентного представлення. Разом із механізмом квантування, кодувальник і декодувальник де-факто виконуватимуть роль першого етапу генерації – створення якісних дискретних представлень для подальшого навчання впровадженої мережі трансформеру, що власне й генеруватиме зображення.

Практично втілити цей апарат можливо, якщо кодувальник і декодувальний доцільно уявити як згорткові нейронні мережі (CNN). Кодувальна складова реалізовуватиметься з послідовності згорткових шарів, блоків зниження роздільної здатності (downsampling block) і залишкових блоків (residual block) для глибшого вивчення ознак, включаючи шари нормалізації та функції активації. Декодувальна частина буде симетричною до кодувальної та використовуватиме шари підвищення роздільної здатності (upsampling block), залишкові блоки, шари нормалізації та функції активації.

Механізм квантування, що є центральним для VQ-GAN, допустимо втілювати завдяки наступного алгоритму:

- 1) вхідне зображення подається на вхід кодувальника, що створює карту ознак нижчої просторової роздільної здатності;
- 2) створюється навчений кодовий словник, який складається з низки ембедінгів, представляючи дискретні прототипи візуальних ознак;
- 3) кожен вектор сформованої карти ознак, замінюється на найближчий вектор кодового словника, у результаті чого створюється квантована карта ознак.

Доцільно, що сам процес навчання кодової книги та кодувальника можна реалізувати наступним способом:

- оскільки операція знаходження найближчого вектора не є диференційовною, для передачі градієнтів від декодувальника до кодувальника можна використати прямий оцінювач, який копіюватиме градієнти;

- кодовий словник оновлюватиметься за допомогою функції втрат, яка наближає вектори самого словника до виходів кодувальника;
- впроваджується втрата зобов'язання, щоб виходи кодувальника були близькими до обраних векторів кодового словника й не зростали довільно.

З метою покращення якості реконструйованих зображень, в архітектуру моделі VQ-GAN вводиться патч-орієнтований дискримінатор, що навчається відрізняти реальні зображення від реконструйованих, змушуючи автокодувальник створювати більш реалістичні та деталізовані реконструкції.

Важливо, що мережа VQ-GAN є лише автокодувальником, який навчається реконструювати зображення. Отримувані дискретні латентні представлення необхідно використати як вхідні значення для трансформера з метою створення згенерованих зображень, втілюючи другий етап генерації.

Трансформер навчається моделювати послідовність дискретних кодів (токенів), отриманих від автокодувальника, і таким чином генерувати нові послідовності, які потім декодуються VQ-GAN у зображення.

Важливо, що класичне представлення мережі трансформера також можна замінити дифузійною моделлю (див. у 3.5.2).

2.3 Дифузійні мережі

2.3.1 Явище дифузії

Окрім розглянутих ГМ (GAN і VQ-GAN), практичного застосування для вирішення проблематики генерування зображень також набули дифузійні моделі, в особливості DDIM. Щоб зрозуміти загальні концепції DM та мати змогу до подальшого проєктування обраної моделі, згадаємо методологію дифузійного процесу при створенні генеративних мереж.

Дифузійний процес у контексті ГМ концептуально схожий на фізичне явище дифузії, коли частинки переміщуються з областей високої концентрації до областей низької, проте трансформується в генерацію нових даних шляхом обернення процесу втрати інформації, спричиненої втручанням шуму [3, 19].

Це метод створення нових даних (зображень) шляхом двоетапного підходу – прямої та зворотної дифузії.

У ході прямої дифузії вхідні дані (чисте зображення) спотворюються шляхом послідовного додавання до них гаусівського шуму на кожному кроці, доки вони не перетворяться на повністю випадковий шум.

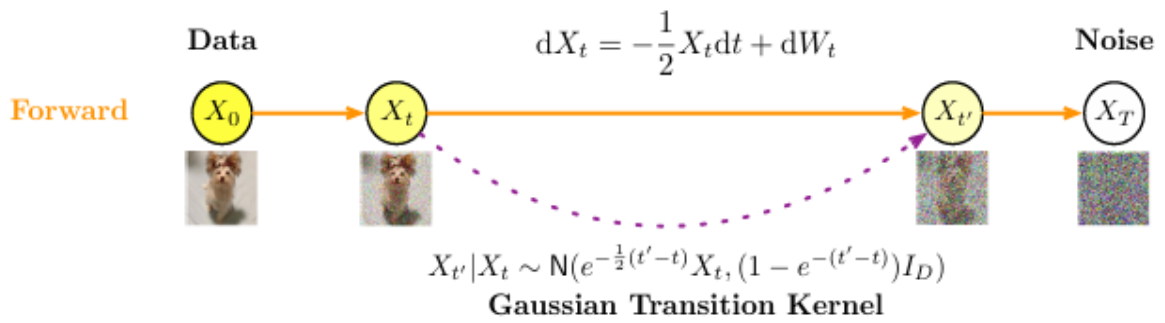


Рисунок 2.3 – Схематичне представлення процесу прямої дифузії [3]

Цей процес описується як марковський ланцюг, де стан зображення на поточному кроці залежить виключно від його стану на попередньому, причому кількість доданого шуму контролюється заздалегідь визначеним розкладом дисперсій. Наприкінці процедури вхідне зображення перетворюється на ізотропний гаусівський шум.

Процес зворотної дифузії є оберненим до процесу прямої дифузії: починаючи з чистого шуму, втілюється механізм покрокового його видалення із відновленням структури та деталей, поки не буде згенеровано новий, осмислений зразок даних, схожий на вихідний.

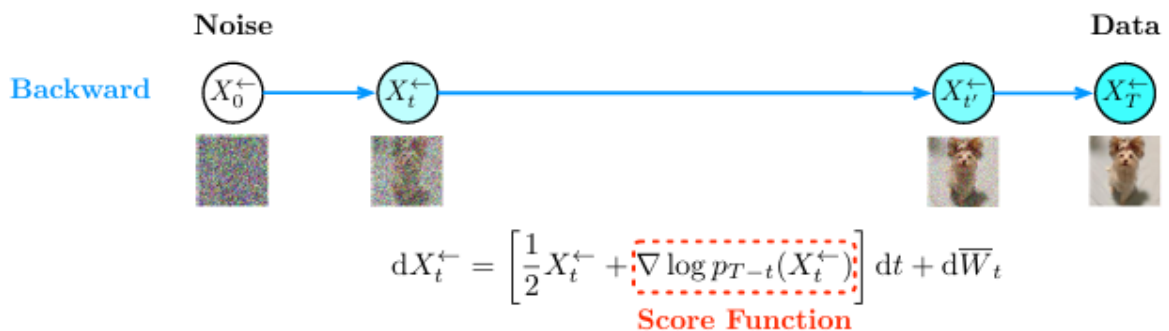


Рисунок 2.4 – Схематичне представлення процесу зворотної дифузії [3]

Фактично, процес зворотної дифузії надає можливість передбачати саме шум (випадковий чи доданий), замість безпосереднього прогнозу очищеного зображення.

Окрім звичайних процесів прямої та зворотної дифузії, у ГМ мають місце явища направленої і латентної дифузії. Направлена дифузія є своєрідним способом керування у процесі зворотної дифузії. Замість безумовної генерації випадкових зображень, можна надавати додаткову інформацію (умову), яка спрямовуватиме процес створення зображень. Такою умовою можуть бути текстові описи, мітки класів, семантичні карти або навіть інші зображення. Основна ідея латентної дифузії полягає в застосовуванні ресурсомісткого дифузійного процесу не до самих пікселів зображення, а до їхніх компактних представлень у стисненому латентному (прихованому) просторі.

2.3.2 Загальні концепції моделей дифузійних мереж

Розуміючи концептуальне представлення явища дифузії для втілення генеративного процесу, розглянемо поняття дифузійної мережі.

ДМ представляють клас імовірнісних ГМ (засновані на математичній концепції марковських ланцюгів), які здатні створювати зображення, шляхом перетворення цифрового шуму на деякий репрезентативний зразок [3, 7, 8].

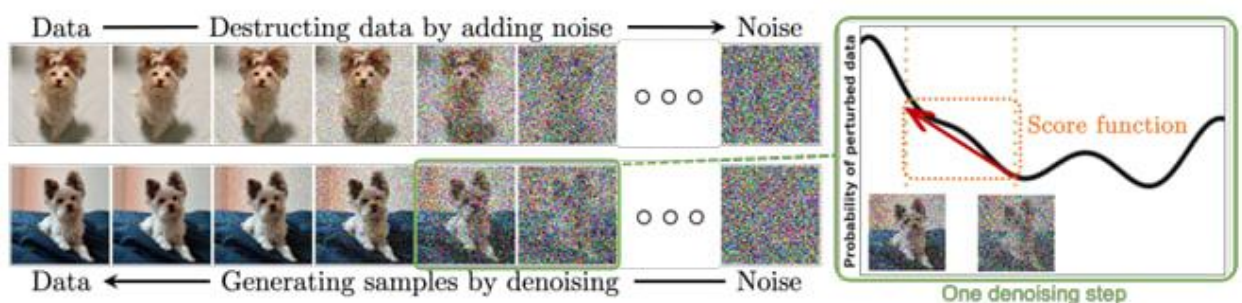


Рисунок 2.5 – Ілюстрація принципу роботи моделей дифузійних мереж [8]

Основна концепція даних моделей полягає у втіленні процесу генерування шляхом поступового обернення перебігу зашумлення. Даний механізм реалізується систематичним руйнуванням структури вхідних даних, шляхом ітеративного

додавання шуму, а потім відновленням втраченої структури, в кінцевому результаті призводячи до генерування нових зразків. Інакше кажучи, DM реалізують процеси прямої та зворотної дифузії.

На відміну від ГМ, які намагаються створити згенерований зразок за один крок, DM розбивають дане завдання на послідовність простіших, керованих етапів дозволяючи легше реалізовувати механізм навчання реалізованої моделі.

Властивістю DM також є здатність генерувати зображення як безумовно, створюючи випадковий зразок із вивченого розподілу, так і умовно, коли процес генерації керується додатковою інформацією – текстовий опис чи мітка.

Для навчання DM типовою цільовою функцією є мінімізація середньоквадратичної помилки (MSE) між справжнім та передбаченим шумом, представлена цільовою функцією:

$$L = E_{t, x_0, \epsilon} [\|\epsilon - \epsilon_0(\sqrt{\bar{a}_t} \cdot x_0 + \sqrt{1 - \bar{a}_t} \cdot \epsilon, t)\|^2] \quad (2.3)$$

2.3.3 Неявна дифузійна модель з усуненням шуму (DDIM)

Розуміючи сутність генеративного процесу дифузії та базові принципи роботи DM, виконаємо проєктування архітектури неявної дифузійної моделі з усуненням шуму – DDIM.

DDIM є ГМ, які здатні відтворювати розподіл даних шляхом поступового обернення процесу зашумлення, втілення якого базується на застосуванні стохастичного диференціального рівняння, за рахунок використання звичайного диференціального рівняння для опису зворотного дифузійного процесу.

Логічно, що центральним компонентом DDIM, відповідальним за навчання зворотного процесу дифузії, є НМ, здатна передбачати шум. Саме тому для проєктування неявної дифузійної моделі за основу доцільно використовувати архітектурні принципи мережі U-Net.

У нашому випадку, основне завдання втілення U-Net полягатиме в передбаченні шуму на кожному часовому кроці зворотного процесу, отримавши на

вхід зашумлене зображення та інформацію про рівень шуму, який був доданий до чистого зображення протягом прямого процесу дифузії.

Загальна архітектура проєктованої мережі матиме характерну U-подібну форму, складаючись з двох основних симетричних частин – кодувальної та декодувальної, між якими знаходиться «вузьке місце» (bottleneck).

Кодувальна частина буде послідовно зменшувати просторову роздільну здатність вхідного зображення, одночасно збільшуючи кількість каналів ознак і дозволяючи вивчити ієрархічні представлення даних (від низькорівневих до абстрактних семантичних текстур). Декодувальна частина симетрично відновлюватиме просторову роздільну здатність, поступово зменшуючи кількість каналів ознак, щоб у кінцевому підсумку згенерувати передбачене з шуму зображення тієї ж розмірності, що й вхідне зашумлене.

Для проєктованої моделі, кодувальник доцільно почати зі згорткового шару для вилучення просторових ознак, що оброблятиме вхідне зашумлене зображення. Далі слідує послідовність блоків зниження роздільної здатності, які включають шар нормалізації, функцію активації та шар зниження дискретизації, з метою зменшення просторової роздільної здатності карт ознак.

Декодувальник буде симетричним до кодувальника і відповідатиме за поступове підвищення просторової роздільної здатності карт ознак до вихідного розміру зображення. Дана частина мережі представлятиме послідовність блоків підвищення роздільної здатності, включаючи шар підвищення дискретизації, а також шари згортки, нормалізації і функцію активації.

Важливо, що після останнього блоку підвищення роздільної здатності необхідно додати вихідний згортковий блок, який матиме таку ж кількість вихідних каналів, як і вхідне зображення, бо саме цей шар генеруватиме остаточне передбачення параметрів шуму.

Bottleneck слугуватиме центральною частиною мережі, де карти ознак матимуть найменшу просторову роздільну здатність, але потенційно найбільшу глибину (кількість каналів). Ця частина складатиметься з декількох згорткових шарів, шарів нормалізації та активації.

Для того, щоб з'єднати шари кодувальної частини з відповідними за просторовою роздільною здатністю шарами декодувальної, втілюються зв'язки пропуску (skip connections). Дані зв'язки дозволятимуть декодувальнику використовувати детальну інформацію, яка могла бути втрачена під час проходження через глибокі шари кодувальника та bottleneck, передаючи низькорівневі ознаки безпосередньо.

Оскільки модель DDIM повинна обробляти зображення з різним рівнем шуму (відповідно до різних часових кроків дифузійного процесу), їй необхідно надавати інформацію про поточний рівень зашумлення. Для цього вхідне скалярне значення слід перетворити на векторне представлення (ембедінг) високої розмірності, використовуючи синусоїдальне позиційне кодування. Завдяки цьому, отриманий часовий ембедінг можна інтегрувати в необхідні блоки, шляхом простого додавання до проміжних активацій, що дозволяє проєктованій моделі ефективно адаптувати свою поведінку.

Звісно, при втіленні моделі DDIM, мережа U-Net виступає лише в якості інструменту, за допомогою якого реалізується покроковий процес знешумлення. Для представлення цілісної моделі необхідно впровадити також функціональні блоки, які реалізують власно дифузійний процес – пряму та зворотну дифузію.

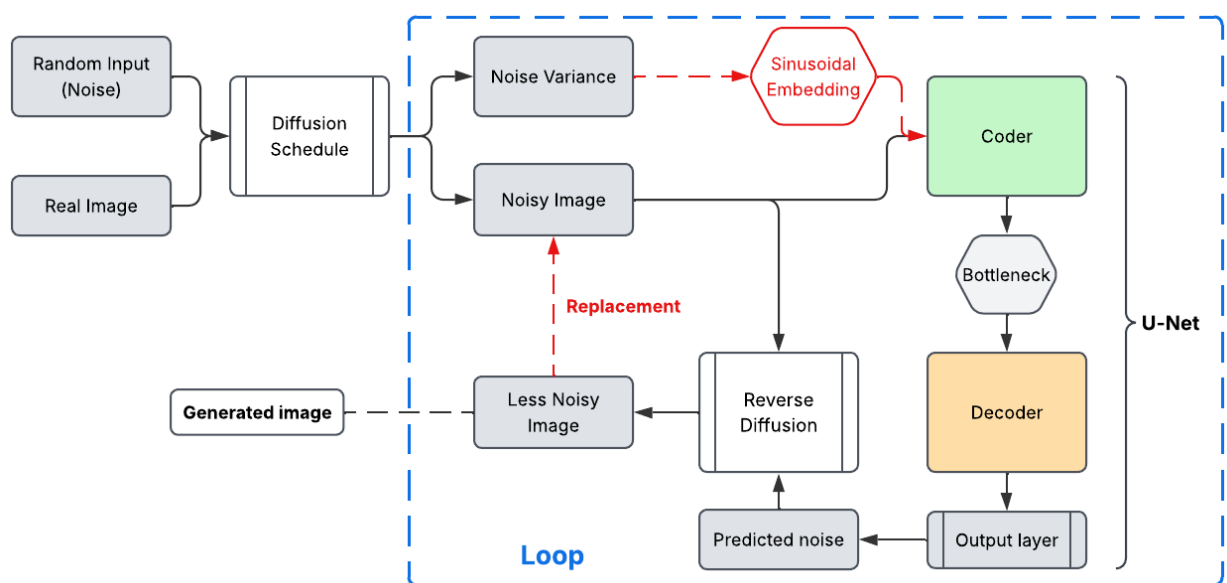


Рисунок 2.6 – Схематичне представлення проєктованої моделі DDIM

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛЕЙ ГЕНЕРАТИВНИХ МЕРЕЖ

3.1 Програмно-апаратні рішення

3.1.1 Середовище розробки та програмні інструменти

З метою здійснення програмної розробки генеративних моделей GAN, VQ-GAN та DDIM, в якості програмних інструментів обрані мова програмування (МП) Python та функціональні можливості фреймворку TensorFlow.

Вибір Python як основної МП для втілення ГМ зумовлений наявністю великого інструментарію – спеціалізованих бібліотек (розширень), реалізацій алгоритмів і методів навчання для розробки НМ. Важливою характеристикою також є простота синтаксису, що значно спрощує процес написання, тестування та налагодження алгоритмів ГМ, а кросплатформенність дозволяє легко інтегрувати код, написаний на інших МП, даючи змогу оптимізувати критичні за продуктивністю частини моделей.

Фреймворк TensorFlow надає гнучкі інструменти для побудови та тренування ГМ, оптимізуючи обчислювальні процеси та зменшуючи споживання пам'яті. Особливе значення мають модулі для визначення шарів моделі та процесів навчання (Keras), забезпечення гнучкості тонкого налаштування операцій і втрат (Core), моніторингу процесу навчання та налагодження (TensorBoard) та реалізації механізмів збереження та експорту натренованих моделей (Model Garden).

В якості середовища розробки (СР) використовується програмне забезпечення (ПЗ) Google Colaboratory (Colab) – безкоштовне хмарне середовище, яке надає доступ до обчислювальних ресурсів системи Google, де необхідні бібліотеки та розширення Python вже встановлені.

Таким чином, створюється оптимальне програмне середовище для практичної реалізації ГМ з метою вирішення задачі генерування зображень.

3.1.2 Особливості обчислювального середовища

Вагомим фактором практичної реалізації ГМ є процедура навчання моделей, що є ресурсоємним процесом, потребуючи значних обчислювальних потужностей. Відповідно, постала необхідність використання високопродуктивного обчислювального середовища графічного процесора (GPU), здатного забезпечити ефективне та прискорене тренування реалізовуваних моделей.

З цією метою застосовувався графічний процесор NVIDIA A100, що повністю підтримує TensorFlow, демонструє виняткову продуктивність при роботі з великими обчислювальними графами, характерними для впроваджуваних ГМ, і дозволяє паралельно обробляти значну кількість операцій при роботі з тензорами та матрицями великих розмірів.

Важливо, що A100 має 40 ГБ оперативної пам'яті, надаючи змогу завантажувати моделі та обробляти пакети зображень без необхідності поділу на підгрупи та обмеження розміру, а також дозволяє тренувати розроблені моделі на повноцінних зображеннях, а не на зменшених копіях.

Отже, використання GPU NVIDIA A100 є не лише доцільним, а й необхідним технічним рішенням, без якого реалізація створюваних ГМ була б значно обмеженою або неможливою.

3.2 Вибір набору даних (датасету)

Для навчання та порівняльного аналізу роботи втілюваних ГМ використовується датасет «Cats Faces», що містить 15.000 кольорових зображень мордочок котів розміром 64×64 пікселі, спеціально підготовлених для задач генеративного моделювання. Вибір саме цього набору даних (НД) зумовлюється низкою технічних факторів.

По-перше, зображення в датасеті мають фіксовану роздільну здатність, надаючи типовий стандарт для відносно простих за реалізацією ГМ та забезпечуючи баланс між достатнім рівнем візуальної складності та доступністю обчислень при обмежених ресурсах.

По-друге, зображення належать до єдиної категорії і мають переважно стандартизовані пози та ракурси об'єктів, даючи змогу зосередитися на вивченні генеративних можливостей моделей, мінімізуючи вплив надмірної варіативності вхідних даних за рахунок зменшення параметрів складності та неоднорідності.

Важливим фактором є використання НД для вирішення задачі генерування зображень, свідчаючи про придатність і релевантність порівнювати отримані результати генерування з наявними реалізаціями.

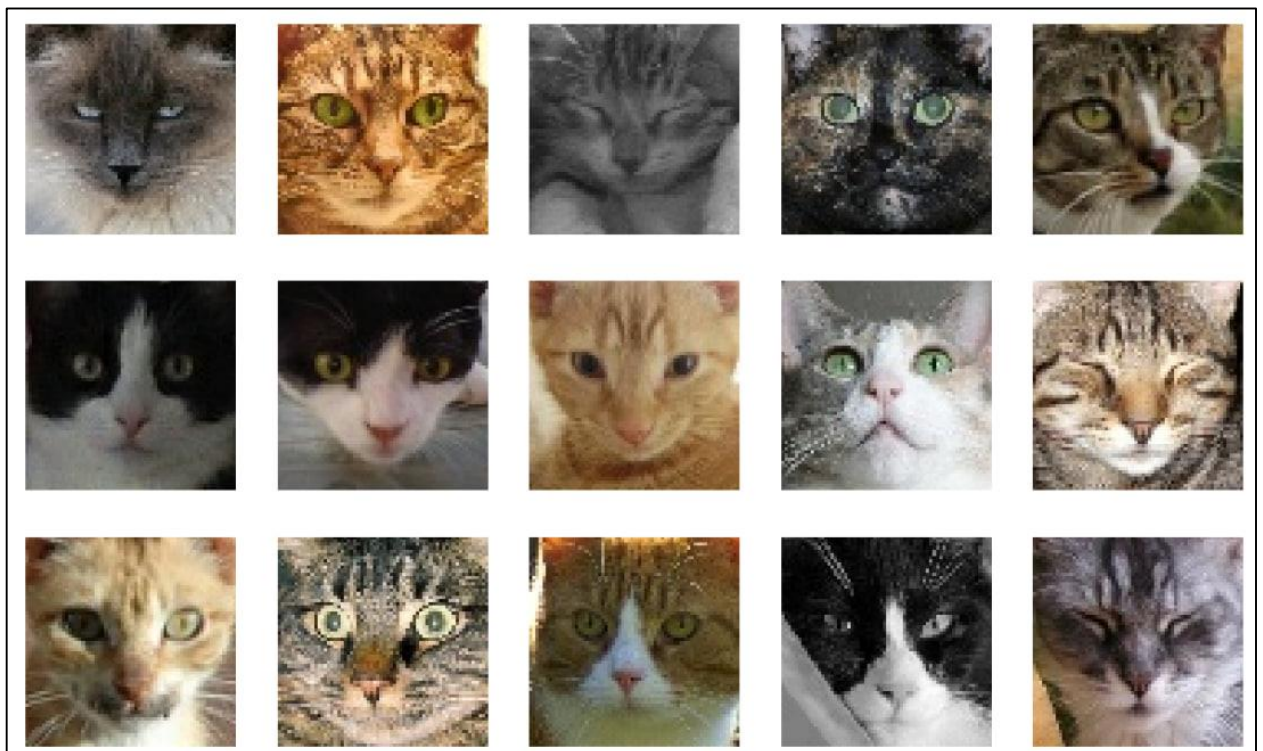


Рисунок 3.1 – Приклад зображень з датасету «Cats Faces»

Зазначимо, що усі ГМ навчатимуться лише на цьому датасеті. Такий підхід було обрано з метою забезпечення об'єктивного порівняння якості генерування, оскільки за незмінним тренувальним набором можна точно оцінити, наскільки кожна з моделей справляється з відтворенням структури та візуальних особливостей об'єктів. Також надається можливість для визначення якості відновлення образу, аналізу стабільності навчання, проведення візуальної та метричної оцінки якості згенерованих зображень, досліджуючи здатність моделей до узагальнення в межах обмеженого домену вхідних даних.

3.3 Програмне втілення моделі GAN

3.3.1 Попередня обробка датасету

Початковим етапом реалізації спроектованої моделі простої генеративно-змагальної мережі GAN – є завантаження даних (датасету) з локального середовища комп'ютера у СР і їх попередня обробка.

Оскільки НД представляє собою архів формату ZIP, спочатку виконується процедура розархівування з метою отримання зображень. Для цього використовуються функціональні можливості пакетів *zipfile* та *os*.

```
import zipfile
import os

# Розархівування завантаженого датасету
with zipfile.ZipFile("cats.zip", "r") as zip_ref:
    zip_ref.extractall("cats")

# Перевірка розархівування
os.listdir("cats")
```

Рисунок 3.2 – Вихідний програмний код розархівування даних

З метою подальшого використання датасету виконуються препроцесінг та оптимізація даних. Для цього кожне зображення автоматично масштабується до необхідного розміру пікселі з трьома кольоровими каналами (модель RGB). Крім того, дані «перемішуються» з метою уникнення упередженості моделі в процесі навчання та встановлюється фіксований розмір батчу, дозволяючи ефективно оброблювати дані невеликими групами.

Нормалізацію зображень виконується за допомогою шару *Rescaling*, який переводить значення пікселів із діапазону $[0, 255]$ у стандартний $[0, 1]$, причому нормалізаційний шар застосовується до кожного зображення, отримуючи послідовності нормалізованих батчів зображень без міток.

Оптимізація продуктивності втілюється завдяки методам *cache()*, який дозволяє зберігати дані в пам'яті після першої епохи, та *prefetch()*, що автоматично вибирає оптимальний розмір буфера для асинхронної підготовки даних у фоновому режимі протягом навчання моделі.

```

bs = 128
dataset = tf.keras.utils.image_dataset_from_directory(
    "cats",
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size=bs,
    image_size=(64, 64),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation='area',
    follow_links=False,
    crop_to_aspect_ratio=False,
)

# Обробка даних датасету (label -> непотрібні)
normalization_layer = tf.keras.layers.Rescaling(1./255)
dataset = dataset.map(lambda x, y: (normalization_layer(x)))
AUTOTUNE = tf.data.AUTOTUNE
dataset = dataset.cache().prefetch(buffer_size=AUTOTUNE)

```

Рисунок 3.3 – Вихідний програмний код препроцесінгу та оптимізації даних для моделі GAN

3.3.2 Реалізація архітектури та механізму роботи

Перейдемо до етапу втілення моделі GAN (див. Додаток Г). Згадаємо, що в основі проєктованої архітектури (відповідно до 2.2.2) знаходяться генератор і дискримінатор, взаємодія яких надає змогу отримати згенероване зображення.

Генератор реалізуємо як розширювальну згорткову нейронну мережу, яка поступово трансформує випадковий вектор (шум) у повноцінне зображення фіксованого розміру з трьома кольоровими каналами.

Побудова починається з вхідного шару, що приймає на вхід вектор випадкових чисел, який виконує роль латентного представлення (простору, з якого здійснюється генерація зображень). Далі використовується повнозв'язний (*Dense*) шар, який представляє вхідний вектор у тензор, розміром початкового зображення. Після цього результат нормалізується за допомогою шару *BatchNormalization*, а для додавання нелінійності застосовується функція активації *LeakyReLU*, яка дозволяє негативні значення, зменшуючи ризик «загибелі» нейронів.

Наступним етапом є використання послідовності згорткових транспонованих шарів (*Conv2DTranspose*), що поступово збільшують розмір просторового представлення зображення та виконують обернену операцію згортки: розширюють зображення на кожному рівні генератора. Кожен такий рівень включає оптимальну кількість фільтрів, дозволяючи моделі послідовно навчитися абстрактним ознакам, а потім – більш деталізованим особливостям зображення.

Завершується генератор вихідним шаром, який виконує згорткову транспоновану операцію з трьома фільтрами і функцією активації *sigmoid*, надаючи змогу обмежити значення пікселів у межах діапазону $[0, 1]$ для роботи з нормалізованими зображеннями.

Дискримінатор має виконувати роль класифікатора, тому його побудова втілюється за принципом зменшуючої згорткової мережі, яка поступово стискає просторову інформацію, витягуючи з неї найбільш релевантні ознаки.

Вхідний шар дискримінатора отримує кольорове зображення, яке відповідає справжньому (з набору) чи згенерованому генератором.

Основна частина складається з кількох згорткових шарів (*Conv2D*), кожен із яких виконує зменшення просторового розміру зображення, причому після згортки застосовується функція активації *LeakyReLU* та регуляризаційний шар (*Dropout*), який випадково обнуляє частину нейронів для уникнення явища перенавчання. Ці рівні дозволяють моделі витягувати ієрархічні ознаки – від простих ліній і форм до більш абстрактних структур, характерні для об'єктів на зображеннях.

Після серії згорток, вихідний тензор перетворюється у вектор за допомогою шару *Flatten*, після чого використовується щільний повнозв'язний шар, який стискає інформацію.

Останній вихідний шар дискримінатора має лише один нейрон з активацією *sigmoid*, дозволяючи моделі розрахувати ймовірність того, що зображення є реальним (значення функції близьке до 1) або згенерованим (значення функції близьке до 0). Завдяки цьому дискримінатор виконує бінарну класифікацію.

Збірка функціональних елементів відбувається в якості об'єктів моделі Keras, об'єднуючи вхідні та вихідні шари.

Щоб об'єднати реалізовані частини мережі в повноцінну модель GAN, створюється окремий клас *Model* (спадкоємець від *keras.Model*), функціонал якого виконує інтеграцію архітектур обох мереж і містить функціональну логіку, необхідну для процесу навчання.

У конструкторі класу виконується ініціалізація двох окремих моделей генератора та дискримінатора, а для кожної функціональної частини створюються оптимізатори типу *Adam* з невеликою швидкістю навчання, даючи змогу моделі оновлювати ваги плавно та гарантуючи стабільну процедуру тренування.

Основними методами класу є *gan_loss()* та *training_step()*.

Функція *gan_loss()* використовується для обчислення генеративної помилки, що представляє значення бінарної крос-ентропії між передбаченням дискримінатора і справжніми мітками. Дана функція допомагає однаково оцінювати помилку моделі, використовуючи єдиний критерій правдоподібності.

Функція *training_step()* є виконує один крок навчання генератора та дискримінатора, реалізуючи послідовний алгоритм:

- 1) генерується тензор випадкового шуму;
- 2) використовується контекстний менеджер *GradientTape* для генератора та дискримінатора, щоб роздільно відстежувати їх градієнти і оновлювати ваги;
- 3) генератор створює зображення з шуму, а дискримінатор робить передбачення як для згенерованих, так і для справжніх зображень;
- 4) обчислюються значення втрат для дискримінатора та генератора;
- 5) розраховуються градієнти обох мереж, після чого оптимізатори оновлюють ваги відповідно до отриманих значень.

У результаті виконання функції повертається значення помилки дискримінатора, що надалі використовується для моніторингу навчання.

Створення об'єкта моделі GAN відбувається за допомогою ініціалізації екземпляру класу *Model*, який об'єднує генератор і дискримінатор в одну систему для навчання ГМ.

3.3.3 Навчання

Після побудови загальної моделі, запускається процес навчання, що представляє багаторазове проходження через початковий набір зображень.

Тренування моделі GAN концептуально реалізується в змагальному стилі, де генератор намагається «обдурити» дискримінатор, який відрізняє справжні зображення від згенерованих.

Практично, навчання триває протягом заданої кількості епох, кожна з яких втілює один повний прохід по всіх зображеннях датасету. Для визначеного батчу викликається метод *training_step()*, який генерує зображення з випадкового шуму, пропускає справжні й згенеровані зображення через дискримінатор, розраховує генеративну помилку та оновлює ваги через обчислені градієнти.

Візуалізація результатів навчання відбувається шляхом виведення інформації щодо поточної епохи навчання та значення функції втрат дискримінатора, а також будується графік втрат (див. Додаток В). Додатково демонструються згенеровані зображення, з метою візуальної оцінки якості роботи ГМ у динаміці.

3.4 Програмне втілення моделі DDIM

3.4.1 Попередня обробка датасету

Аналогічно до практичного втілення генеративної моделі GAN, для програмної реалізації спроектованої дифузійної моделі типу DDIM здійснюється підготовка зображень обраного датасету.

Процедури завантаження і розархівування даних із набору, а також операції препроцесінгу й оптимізації є практично ідентичними до тих, які були розглянуті в 3.3.1. Відмінність підходу попередньої обробки зображень полягає лише в зміні процедурі нормалізації.

Для моделі DDIM розроблена окрема користувацька функція, функціонал якої здійснює поділ значення пікселів на 256. У даному випадку, підхід є зовнішнім по відношенню до втілюваної ГМ, оскільки процедура нормалізації виконується до подачі даних датасету на вхід генеративної мережі.

Крім того, до потоку даних додатково застосовується алгоритм «перемішування» після виконання нормалізації, дозволяючи в значній мірі усереднити розподіл зображень у самих батчах та зменшуючи ймовірність запам'ятовування моделлю порядку зображень з вхідного НД.

```
# Виконуємо препроцесінг даних (зображень)
@tf.function
def process(x):
    # приводимо значення пікселів до [0; 1]
    return (x/255.0)

dataset = dataset.map(process)
dataset = dataset.shuffle(500).cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

Рисунок 3.5 – Вихідний програмний код препроцесінгу та оптимізації даних для моделі DDIM

3.4.2 Реалізація архітектури та механізму роботи

Відповідно до проєктування у 2.3.3, програмно реалізуються функціональні елементи та генеративний механізм моделі DDIM (див. Додаток Д).

Враховуючи природу DM, у процесі генерування зображень обробляються дані, що знаходяться на різних часових стадіях дифузійного процесу. На кожному такому кроці рівень шуму змінюється, тому модель повинна розуміти, на якому етапі генерування вона перебуває, щоб мати можливість правильно реконструювати згенероване зображення. Тому звичайне скалярне значення часу (одновимірний тензор) перетворюється на вектор високої розмірності (ембедінг), шляхом здійснення синусоїдального позиційного кодування.

Програмно реалізується це за допомогою методу, що приймає на вхід тензор скалярної величини часових кроків, створює набір у логарифмічному масштабі для визначеної кількості частот та застосовує періодичні функції (синус і косинус) до вхідного тензору, формуючи багатовимірний ембедінг. У кінцевому результаті отримується вектор з фіксованою кількістю значень для кожного входу, який кодує поточний крок у дифузійному процесі.

```

embedding_dims = 32 # кiл-ть чисел, що будуть на виході
embedding_max_frequency = 1000.0

def sinusoidal_embedding(x): # передаємо час з початку дифузії
    embedding_min_frequency = 1.0
    frequencies = tf.exp(
        tf.linspace(
            tf.math.log(embedding_min_frequency),
            tf.math.log(embedding_max_frequency),
            embedding_dims // 2,
        )
    )
    angular_speeds = 2.0 * math.pi * frequencies
    embeddings = tf.concat(
        [tf.sin(angular_speeds * x), tf.cos(angular_speeds * x)], axis=3
    )
    return embeddings

```

Рисунок 3.6 – Вихідний програмний код функції синусоїдального позиційного кодування для перетворення одновимірної скалярної тензори значення часу

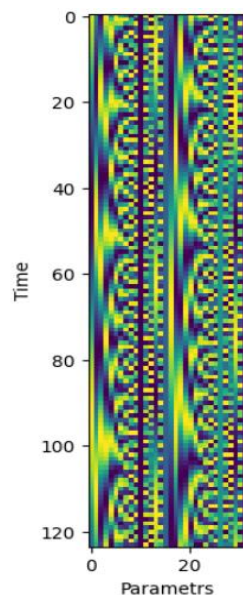


Рисунок 3.7 – Візуалізація функції синусоїдального позиційного кодування

Втілювана модель передбачає, що вхідне зображення поступово зашумлюється в процесі прямої дифузії. Для того, щоб керувати даним перебігом, встановлюються значення рівнів сигналів – найнижчий, що залишається після максимального зашумлення, та найвищий, який існує на початку дифузійного процесу. Ці параметри визначають діапазон, у якому змінюватиметься співвідношення сигналу до шуму, а згодом також використовуються для обчислення ваг компонентів моделі при додаванні шуму й відновленні зображення.

Побудова DDIM здійснюється завдяки архітектурі U-Net, що має в якості основних компонентів кодувальну та декодувальну частини, зв'язані за допомогою пропускних з'єднань, а також bottleneck.

На початковому етапі реалізації визначаються глобальні параметри `widths` і `block_depth`. Перший встановлює кількість фільтрів (каналів) на кожному рівні мережі, а другий – вказує на число разів застосовування основного розрахункового блоку на кожному рівні кодувального та декодувального компонента, а також у bottleneck мережі.

Основний розрахунковий блок (*ResidualBlock*) є базисом U-Net, що складається з двох згорткових шарів, які обробляють вхідний тензор. З метою унеможливлення втрати інформації, вхідна активація в блоці зберігається як залишкове з'єднання та додається до виходу. У випадку незбіжності кількості каналів, використовується додатковий згортковий шар для узгодження розмірності та реалізується *Swish*-активація для подальшого покращення навчання моделі.

```
# Головний "розрахунковий" блок
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = layers.Conv2D(width, kernel_size=1)(x)
        x = layers.BatchNormalization(center=False, scale=False)(x)
        x = layers.Conv2D(width, kernel_size=3, padding="same", activation=keras.activations.swish)(x)
        x = layers.Conv2D(width, kernel_size=3, padding="same")(x)
        x = layers.Add()([x, residual])
        return x
    return apply
```

Рисунок 3.8 – Вихідний програмний код *ResidualBlock*

Задля втілення кодувального компоненту мережі, реалізується знижувальний блок (*DownBlock*), який характеризує один рівень кодувальника та відповідає за зменшення просторового розміру зображення. Даний блок додає до зображення визначений часовий ембедінг, розтягнутий до потрібного розміру, послідовно застосовує декілька *ResidualBlock*, зберігає проміжні активації, що

використовуватимуться в підвищуючих блоках декодувальної частини, та застосовує усереднений пулінг для зменшення розміру тензора.

```
# Знижувальний блок -> зменшує розмір зображення та збільшує кіл-ть фільтрів
def DownBlock(width, block_depth):
    def apply(x):
        x, skips, emb = x
        height = x.shape[1]
        # використовуємо часовий "embedding" разом із конкатинацією зображення
        e = layers.UpSampling2D(size=height, interpolation="nearest")(emb)
        x = Concatenate()([x, e])
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = layers.AveragePooling2D(pool_size=2)(x)
        return x
    return apply
```

Рисунок 3.9 – Вихідний програмний код *DownBlock*

Заради впровадження декодувального компоненту, втілюється підвищуючий блок (*UpBlock*), що описує один рівень декодувальника та відновлює просторову роздільність зображення. Його функціональні можливості полягають у збільшенні розміру тензора та додаванні збережених активацій з відповідного знижувального блоку, за допомогою циклічного виклику *ResidualBlock*.

```
# Підвищуючий блок
def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)
        for _ in range(block_depth):
            x = layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x
    return apply
```

Рисунок 3.10 – Вихідний програмний код *DownBlock*

Для реалізації принципу роботи мережі U-Net з попередньо створених блоків, втілюється програмний метод, що приймає два тензори – зображення з накладеним шумом та скаляр, який відповідає за кодування рівня зашумлення (часовий крок дифузійного процесу):

- 1) до вхідного значення часового кроку використовується синусоїдальне позиційне кодування для отримання ембедінгу, який масштабується до просторових розмірів зображення та додається до вхідного тензора на кожному рівні мережі;
- 2) зашумлене зображення проходить визначене число *DownBlock*, що зменшують його розмірність і зберігають проміжні активації у списку;
- 3) отриманий тензор обробляється в центральній частині мережі – bottleneck з декількох послідовних *ResidualBlock*;
- 4) виконується зворотне масштабування тензору блоками *UpBlock*, де відновлюються просторові розміри, а до поточних активацій поступово додаються запам'ятовані пропускні з'єднання.

Наприкінці, до вихідного тензора повторно додаються дані щодо вхідного зображення та ембедінгу часу, після чого застосовується згортковий шар з активацією *Swish*, у такий спосіб допомагаючи покращити передавання ознак. Вихідним є шар згортки із 3 каналами та нульовою ініціалізацією ваг, який видає остаточне передбачення шуму.

Такий підхід дозволяє почати тренування з нульового передбачення і поступово вдосконалити результати для стабільного навчання моделі DDIM.

Збірка U-Net реалізується шляхом виклику конструктору відповідного класу, який формує модель, визначаючи її вхідними (зашумлене зображення і часовий крок) і вихідним (містить передбачення залишкового шуму) тензорами.

Повноцінна модель DDIM реалізується класом *DiffusionModel*, що представляє високорівневу обгортку над архітектурою U-Net та організовує повний цикл генерації та навчання моделі, засновуючись на ідеї поступового очищення зображення від шуму.

У конструкторі створюються шар нормалізації, мережа U-Net, яка відповідає за передбачення залишкового шуму на кожному етапі дифузії, та оптимізатор із вбудованим згасанням.

Інтерфейс класу реалізує низку основних функцій для:

- виконання процедури денормалізації, повертаючи параметри зображення до діапазону $[0, 1]$;

- визначення того, як сигнал і шум змінюються протягом дифузійного процесу, переводячи часовий параметр у шумову й сигнальну компоненти;
- втілення основного методу генерування, де випадковий шум поступово очищується, використовуючи передбачення залишкового сигналу на кожному кроці (час рухається у зворотному напрямку і кожен новий стан будується на основі попереднього);
- генерування нових зображень з чистого шуму, використовуючи основний метод та виконуючи денормалізацію;
- навчання моделі, коли для кожного батчу зображень виконується нормалізація, додається випадковий шум, реалізується передбачення значення та розраховується функція втрат між справжнім шумом і передбаченим, що мінімізується оптимізатором.

Ініціалізація DDIM виконується завдяки виклику конструктора класу *DiffusionModel*, у наслідок чого створюється екземпляр моделі, готової до навчання та подальшого процесу генерування зображень.

3.4.3 Навчання

Важливим етапом перед тренування DDIM є початкова процедура адаптації нормалізатора. Пояснюється це нездатністю відповідного шару НМ одразу виконати процес нормалізації, оскільки спершу необхідно обчислити середнє значення та відхилення пікселів на окремих прикладах зображень навчальної вибірки (визначена кількість репрезентативних батчів), у такий спосіб практично запам'ятавши статистику даних, що використовуватиметься в процесі тренування моделі та генерування зображень із випадкового цифрового шуму.

```
# Ініціалізація нормалізатора
for a in dataset.take(10):
    model.normalizer.adapt(a)
```

Рисунок 3.11 – Вихідний програмний код процедури адаптації нормалізатора

Сам алгоритм навчання є схожим на процес тренування моделі GAN, однак має на меті надати можливість DDIM запам'ятати процедуру поступового очищення шуму задля генерування зображень.

Процедура триває протягом заданої кількості епох, кожна з яких характеризується поступовим проходом по всьому датасету, розбитому на батчі (розмір батчу щоразу перевіряється, щоб уникнути неповних партій).

Над кожним батчем здійснюється один крок навчання шляхом нормалізації вхідних зображень, генерування випадкового шуму та дифузійних кроків, створення зашумленої версії зображення, виконання прогнозування мережі (оцінка значення доданого шуму), обчислення втрат (абсолютна різниця між справжнім і передбаченим шумом) та застосування градієнтного спуску для оновлення ваг.

Візуалізація результатів навчання втілюється за рахунок виведення інформації стосовно поточної епохи навчання, середнього значення функції втрат та побудови належного графіку втрат у режимі реального часу (див. Додаток В).

Таким чином, кожна тренувальна епоха поступово вдосконалює здатність реалізованої моделі генерувати нові зображення, виконуючи процес зворотної дифузії, який починається з випадкового шуму.

3.5 Програмне втілення моделі VQ-GAN

3.5.1 Попередня обробка датасету

Відомо, що VQ-GAN напряду може працювати з дискретним латентним простором зображень, застосовуючи механізм квантованого ембедінгу.

Щоб спростити та прискорити процедури розробки й навчання втілюваної моделі, доцільно попередньо підготувати латентний НД із вхідного датасету зображень, замість повної обробки навчальної вибірки кодувальником.

Задля отримання набору компактних за формою зображень, використовується попередньо реалізована й натренована модель автокодувальника (формальний опис програмного втілення наведено у 3.5.2).

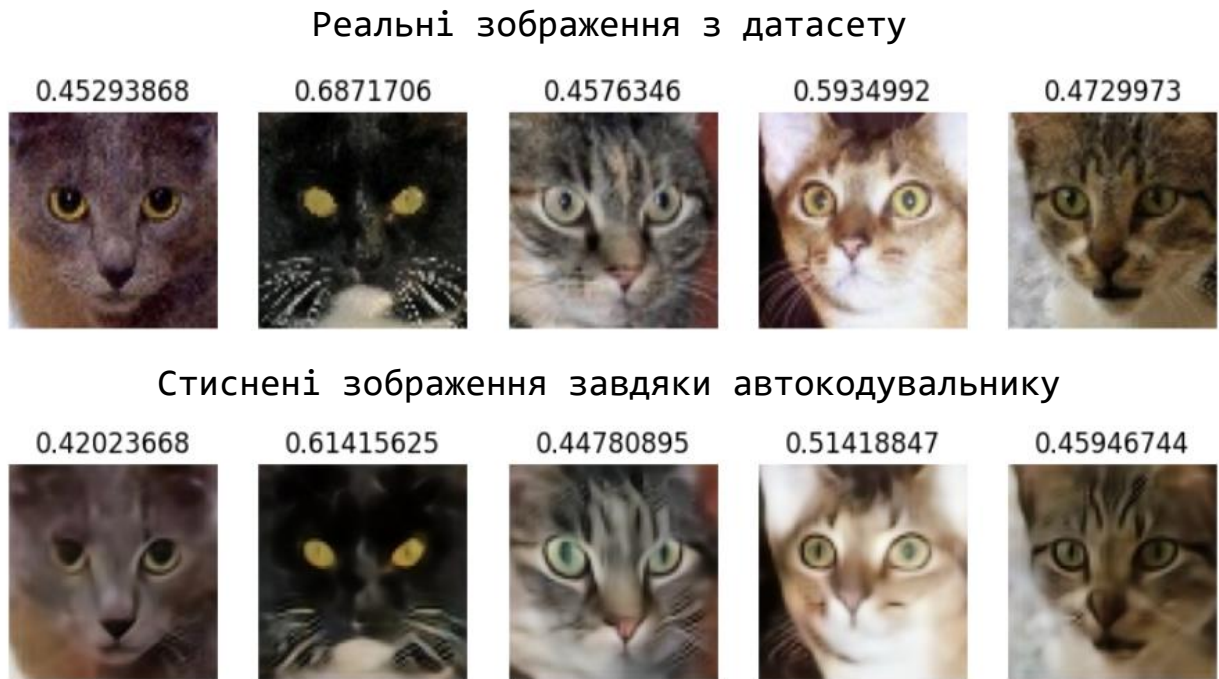


Рисунок 3.12 – Демонстрація стиснених зображень обробленого датасету

Сама процедура попередньої обробки починається з препроцесінгу зображень, включаючи нормалізацію піксельних значень до діапазону $[0, 1]$. Кожне нормалізоване зображення передається до кодувальника, який стискає вхідні дані та генерує відповідне латентне представлення – тензор меншого розміру, що містить узагальнену інформацію про зображення. Цей алгоритм застосовується до всього датасету, даючи змогу ефективно обробити всі зображення та зберегти їх у вигляді латентних векторів.

```
# Виконуємо препроцесінг даних
@tf.function
def process(x):
    # приводимо значення пікселів до [0...1]
    l = encoder(x/256.0)
    return l

dataset = dataset.map(process)
```

Рисунок 3.13 – Вихідний програмний код збереження латентного НД

Після отримання набору латентних представлень, оброблений датасет зберігається у TFRecord-файлі (формат для великих обсягів структурованих даних).

Для цього створюється об'єкт *TFRecordWriter*, який у циклі по датасету відбирає фіксовану кількість латентних представлень зображень, кожне з яких перетворюється на байтовий формат, та записує дані у файл.

Фактично, втілюється єдиний компактний запис, який містить задану кількість латентних представлень, що використовуватимуться моделлю VQ-GAN.

3.5.2 Реалізація архітектури та механізму роботи

Функціональні елементи й цілісний алгоритм генерування VQ-GAN (див. Додаток Е) програмно реалізуються відповідно до спроектованого представлення моделі (наведене у 2.2.3), що зображує об'єднання автокодувальника і латентної дифузійної моделі в єдину ГМ.

Базовим компонентом автокодувальника є апарат векторного квантування простору латентних ознак, що втілюється через інтерфейс класу *VectorQuantizer*. Його основна задача полягає в заміні безперервних вихідних векторів із кодувальника на дискретні вектори словника кодів.

На початковому етапі задаються параметри розмірності словника і вектору ознак у латентному просторі, а також гіперпараметр для керування балансом між зміною кодувальника і підлаштуванням словника кодів (codebook). Важливо, що самі вектори codebook визначаються випадковим чином і зберігаються як навчувані параметри, що оновлюватимуться протягом тренування моделі.

Процедура прямого проходу тензору через шар квантувальника здійснюється за визначеним алгоритмом:

- 1) зберігається форма вхідного тензора, а сам тензор розгладжується у двовимірний масив, кожен рядок якого є окремим вектором ознак;
- 2) реалізується принцип квантування, шляхом обчислення для кожного вектору ознак індексу найближчого вектора словника кодів, що перетворюється у one-hot вектор і множиться на матрицю codebook, з метою заміни виходів кодувальника на найближчі дискретні вектори;
- 3) отриманий тензор повертається до початкової форми вхідного тензору;

4) визначаються функції втрат, які змушують кодувальник краще узгоджуватись із дискретними векторами словника кодів і дозволяють кодовому словнику адаптуватися до нових вихідних даних;

5) виконується обхід зворотного проходу градієнтів, і квантований тензор комбінується з вхідним, щоб передати градієнти лише кодувальнику, використовуючи значення дискретного представлення.

Пошук найближчого вектору словника для кожного вхідного вектору втілюється окремою функцією, суть якої заснована на обчисленні схожості між вхідними векторами та всіма векторами словника кодів, за рахунок чого розраховується відстань за формулою евклідової метрики і для кожного вектору вхідного простору вибирається індекс найближчого вектору у codebook.

Створення шару векторного квантування реалізується як екземпляр класу *VectorQuantizer*, що реалізує механізм заміни безперервних векторів ознак на дискретні представлення зі словника кодів.

Маючи реалізований шар квантування, втілюються й інші компоненти автокодувальної моделі, головною складовою для створення яких виступає основний розрахунковий блок (*ResidualBlock*), що використовує механізм прямого з'єднання (*skip connection*).

```
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = layers.Conv2D(width, kernel_size=1, activation=keras.activations.swish)(x)
        x = layers.Conv2D(width // 2, kernel_size=1, activation=keras.activations.swish)(x)
        x = layers.Conv2D(width // 2, kernel_size=3, padding="same", activation=keras.activations.swish)(x)
        x = layers.Conv2D(width, kernel_size=1, activation=keras.activations.swish)(x)
        x = layers.Add()([x, residual])
        return x
    return apply
```

Рисунок 3.14 – Вихідний програмний код *ResidualBlock*

Принцип його роботи визначається тим, що на вхід подається тензор, який проходитиме через послідовність згорткових шарів, сприяючи кращому потоку градієнтів і формуючи кінцевий результат блоку.

Використовуючи *ResidualBlock*, будується модель кодувальника, яка відповідає за перетворення вхідного зображення у стиснене латентне представлення, що дискретизується через векторний квантувальник.

Фактично, структура кодувальника характеризується послідовними *ResidualBlock* із поступовим збільшенням кількості фільтрів, через які проходить вхідне зображення. Після кожного етапу виконуються операції пулінгу (зменшення розмірності) для ефективного стиснення інформації, а на виході кількість каналів скорочується до визначеної кількості фільтрів за допомогою згорткового шару, узгоджуючись із розмірністю ембедінгів у векторному квантувальнику.

```
# Реалізуємо енкодер
def get_encoder(image_size):
    image = keras.Input(shape=(image_size, image_size, 3))

    x = image

    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = layers.AveragePooling2D(pool_size=2)(x)

    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = layers.AveragePooling2D(pool_size=2)(x)

    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)

    # додаємо шар, що "сильно" зменшує кіл-ть фільтрів (до 4-х)
    x = layers.Conv2D(4, kernel_size=1, activation=keras.activations.swish)(x)

    # додаємо шар квантування
    x = vq_layer(x)

    return keras.Model(image, x, name="encoder")
```

Рисунок 3.15 – Вихідний програмний код побудови кодувальника

Отримане латентне представлення надалі подається на вхід векторного квантування (шар дискретизації), який замінює безперервні вектори на найближчі дискретні коди з кодового словника.

Завдяки тих самих послідовно визначених *ResidualBlock*, будується декодувальник, що виконує обернений до кодувальника процес відновлення зображення зі стисненого дискретного представлення.

По суті, латентне представлення проходить через кілька глибоких *ResidualBlock*, причому дані поетапно відновлюються у розмірність зображення з використанням білінійної інтерполяції. На виході застосовується згортковий шар із трьома фільтрами та функцією активації *sigmoid*, який приводить значення до діапазону $[0, 1]$ і формує фінальне зображення з трьома кольоровими каналами.

```
# Реалізуємо декодер
def get_decoder(latent_size):
    latent = keras.Input(shape=(latent_size, latent_size, 4))

    x = latent
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)

    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)

    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)

    # перетворюємо дані знову в картинку (3 фільтри, 0...1 - sigmoid)
    x = layers.Conv2D(3, kernel_size=1, activation="sigmoid")(x)

    return keras.Model(latent, x, name="decoder")
```

Рисунок 3.16 – Вихідний програмний код побудови декодувальника

Таким чином, декодувальник працює як генератор, створюючи зображення, орієнтуючись лише на дискретну інформацію із кодового словника.

Аналогічно, реалізується компонент дискримінатору, який виконує оцінку реальності зображення, допомагаючи покращити результати декодування.

Його структура представляє послідовність *ResidualBlock* із поступовим зменшенням розмірності, а дані піддаються глобальному усередненню, з метою виділення ключових характеристик всього зображення. На виході блоку, додається повнозв'язний шар з активацією *sigmoid*, який видає ймовірність того, що зображення є справжнім.

```
def get_discriminator(image_size, discriminator_widths):
    image = keras.Input(shape=(image_size, image_size, 3))

    x = image

    for width in discriminator_widths:
        x = ResidualBlock(width)(x)
        x = layers.AveragePooling2D(pool_size=2)(x)
        x = ResidualBlock(width)(x)

    x = keras.layers.GlobalAveragePooling2D()(x)
    x = Flatten()(x)
    x = Dropout(0.3)(x)
    x = Dense(1, activation='sigmoid')(x)

    return keras.Model(image, x, name="discriminator")
```

Рисунок 3.17 – Вихідний програмний код побудови дискримінатора

Для повноцінної збірки моделі автокодувальника реалізується клас *Model*, що об'єднує реалізовані компоненти мережі, а також втілює ЕМА-моделі (представляють метод оновлення ваг, який згладжує результати генерації та зменшує шум) й механізми навчання.

У конструкторі класу додатково ініціалізуються оптимізатори з однаковою швидкістю навчання (*learning rate*) та обмеженням норми градієнтів, у такий спосіб покращуючи стабільність навчання.

Основними методами класу є *gan_loss()* і *train_step()*.

Функція *gan_loss()* реалізує бінарну кросентропію як функцію втрат, яка повертає значення залежно від того, чи має дискримінатор сприймати зображення як реальне чи згенероване.

Функція *train_step()* втілює один крок навчання моделі:

- 1) зображення кодується в латентний простір, а потім декодується вже у реконструйованому вигляді;

2) обчислюється середньоквадратична помилка між вхідним та реконструйованим зображенням, а також додається помилка квантувального шару для покращення якості представлення;

3) дискримінатор оцінює реальні та згенеровані зображення, обчислюються втрати генератора й дискримінатора, і відповідно розраховується значення загальної втрати;

4) виконується оновлення ваг через оптимізатори за рахунок роздільного обчислення градієнтів для кодувальника, декодувальника та дискримінатора;

5) реалізується оновлення ЕМА-моделей, наближаючись до значень основних моделей кодувальника й декодувальника.

Ініціалізується модель VQ-GAN як екземпляр класу *Model* із заданими розмірами зображень та латентного простору.

Важливо, що для подальшого генерування зображень за допомогою втілюваної моделі латентної дифузії, автокодувальна частина надається вже навченою (див. пояснення у 3.5.3).

Реалізація латентної дифузійної моделі, що заміщує собою трансформер, є схожою до втілення моделі DDIM. Основні відмінності полягають лише у втіленні механізму латентної дифузії та процедурі тренування моделі.

Механізм латентної дифузії реалізується як у прямому, так і в зворотному напрямках, і ґрунтується на поступовому перетворенні початкового зображення в шум протягом заданої кількості кроків та відповідному інверсному процесі – поетапному відновленню зображення з шуму. Для цього використовуються гаусівський розподіл і лінійно змінювані коефіцієнти дисперсії на кожному кроці.

У ході тренування втілювана модель латентної дифузії навчається відновлювати шум, який був доданий до зображення. Для кожного зображення у батчі випадковим чином обирається часовий крок, що визначає рівень доданого шуму, після чого генерується випадковий гаусівський шум, перетворюючи зображення в зашумлену версію відповідно до заданої дисперсії у процесі прямої дифузії. Дане зашумлене зображення подається разом із часовим кроком на вхід моделі, яка намагається передбачити доданий шум.

Різниця між справжнім і передбаченим шумом розраховується як середньоквадратична похибка, що використовується для оновлення ваг через зворотне поширення. Також паралельно оновлюється й ЕМА-модель, що зберігає більш стабільні ваги для генерування зображень.

Генерування зображень моделлю латентної дифузії починається з випадкових латентних шумів. На кожному зворотному часовому кроці модель (у даному випадку, ЕМА-модель) передбачає, який шум присутній у поточному зразку, і застосовує формулу зворотної дифузії для поступового очищення цього зразка. Такий процес повторюється для всіх кроків у зворотному порядку – від максимальної зашумленості до потенційно чистого латентного представлення. Наприкінці, очищені латентні вектори декодуються у кінцеві зображення через заздалегідь навчений декодувальник.

3.5.3 Навчання моделі

Безпосередня процедура навчання автокодувальника моделі VQ-GAN полягає в оптимізації трьох компонентів – втрати реконструкції, квантування та втрати дискримінатора. На кожній епосі модель обробляє батчі зображень, що кодуються, квантуються і декодуються. Паралельно, дискримінатор вчиться відрізняти реальні зображення від відновлених, а також оптимізуються ваги компонентів моделі.

Процес тренування моделі латентної дифузії реалізується вже у поступовому навчанню НМ відновлювати початкове зображення із випадкового шуму. Епоха включає алгоритм проходження по датасету, де для кожного батчу спершу генерується шум і зображення зашумлюється відповідно до поточного кроку. Модель навчається передбачати цей шум, і оптимізується за допомогою MSE-втрати між справжнім шумом та передбаченим.

У ході навчання як автокодувальника, так і моделі латентної дифузії, втілюваних у загальній концепції моделі VQ-GAN, додатково виводяться значення поточної втрати та графік змін помилки у часі (див. Додаток В).

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Результати роботи генеративних мереж та згенеровані зображення

4.1.1 Проведення візуального аналізу

Процес тестування роботи розроблених генеративних моделей надає можливість отримати велике різноманіття згенерованих зображень для дослідження якості генерації та ефективності використання тієї чи іншої моделі для вирішення задачі генерування зображень.

Для первинної оцінки результатів моделей проводиться візуальне порівняння згенерованих зображень, у такий спосіб інтуїтивно оцінюючи реалістичність, чіткість, деталізацію та відповідність структури об'єктів.

На представлених нижче прикладах демонструються типові зображення, що були згенеровані кожною з моделей після завершення етапу навчання.

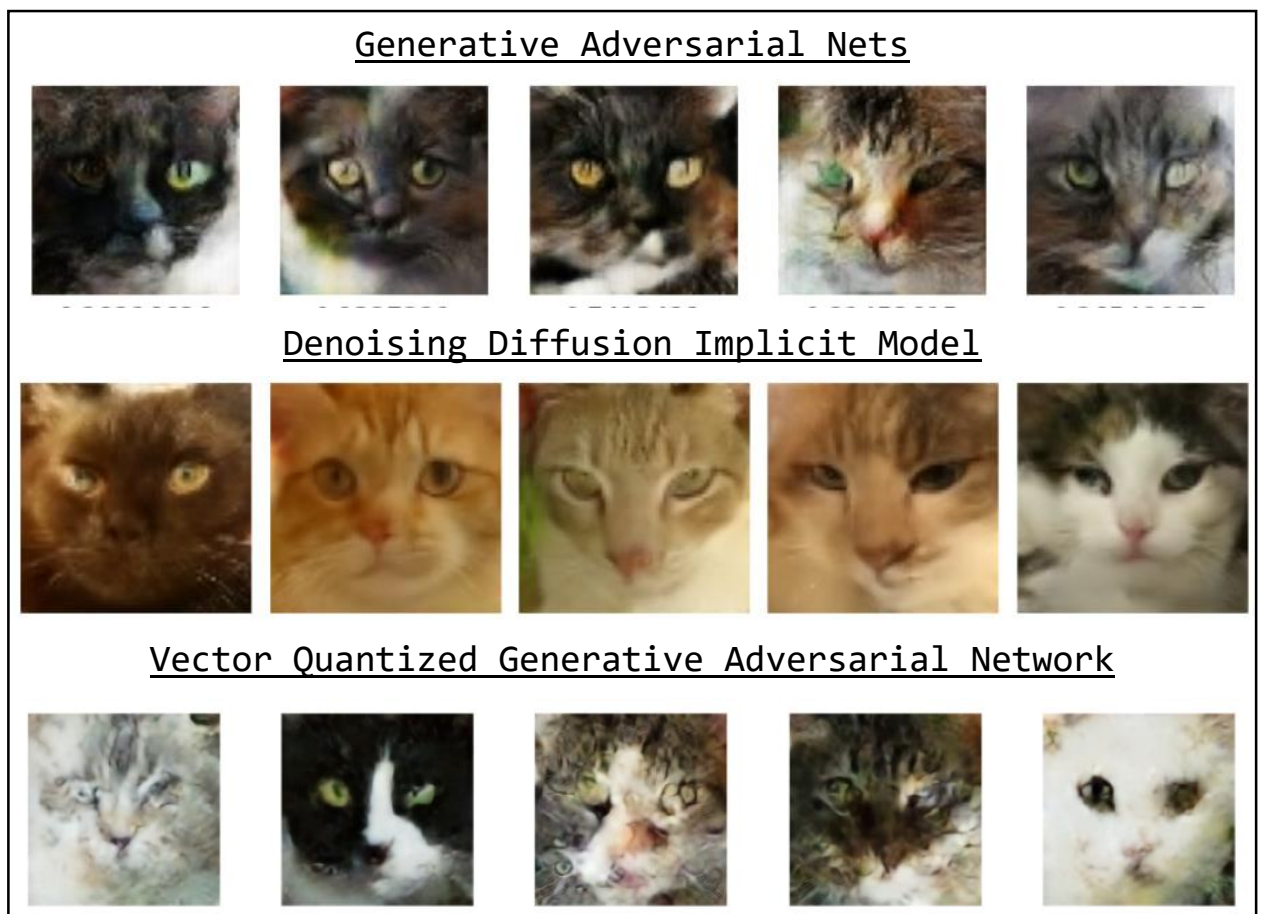


Рисунок 4.1 – Результати генерування зображень розробленими моделями

Так зображення, отримані за допомогою простої моделі GAN, демонструють базову здатність до формування загального вигляду об'єкта. У більшості прикладів збережено основну структуру мордочки кота, що демонструється присутністю очей, вух та контурів хутра. Водночас, чітко помітні явні недоліки результатів генерування. Наприклад, конкретні зображення мають розмиті ділянки й контури, випадкові артефакти, спотворення пропорцій та нестабільну текстуру. Крім того, на деяких прикладах можна побачити деформацію очей та спотворення форми голови, через що зображення виглядають неприродно. Таким чином, ілюструється типова ознака обмежених генеративних можливостей втіленої простої моделі GAN при роботі з комплексними структурами.

Результати моделі DDIM демонструють якісніші й більш фотореалістичні зображення, які значно перевершують згенеровані результати моделі GAN за візуальною якістю. Більшість зображень надають правильно збережені пропорції, завдяки чому структура мордочок наближена до вигляду справжніх котів. Додатково можна відзначити покращену деталізацію очей, текстури хутра та загальну цілісність зображень. У багатьох прикладах досягнуті також стабільність освітлення та природність розташування анатомічних рис. Проте певна частина результатів містить помітні артефакти або надлишкову згладженість, що фактично вказує на залежність якості генерації від параметрів дифузійного процесу. Незважаючи на це, модель DDIM показує природність згенерованих зразків та хорошу здатність до поетапного «очищення» шуму, репрезентуючи так само загально вищу якість зображень та підвищену стабільність процедури генерування.

Втілювана мережа VQ-GAN, у поєднанні з моделлю латентної дифузії, натомість не змогла досягнути очікуваної якості. У наведених прикладах переважають сильно викривлені зображення з нечіткими рисами, неприродною текстурою та наявністю випадкових артефактів. Крім того, відсутність збереженої структури мордочок, поява сторонніх кольорів та симетричних спотворень свідчать про те, що для навчання моделі було використано занадто малий за обсягом латентний набір, або модель була недотренованою чи неправильно параметризованою.

У результаті, візуальний аналіз демонструє, що найвищу якість зображень надає розроблена дифузійна модель DDIM, яка змогла досягти найкращого балансу між природністю генерування, структурною цілісністю та деталізацією зображень. Модель GAN показала базову якість з незначними спотвореннями, що є характерним для ГМ без складної латентної структури та поетапного відновлення даних. Найгірші ж результати генерування надала модель VQ-GAN, що може бути пов'язано з технічними складнощами під час навчання або недостатньою роздільною здатністю векторного представлення.

4.1.2 Об'єктивне дослідження результатів

Для виконання об'єктивного дослідження роботи моделей використовуються дві найпоширеніші метрики оцінки ГМ – Frechet Inception Distance (FID) та Inception Score (IS).

Метрика FID вимірює статистичну відстань між розподілами ознак, які були отримані з реальних та згенерованих зображень за допомогою проміжних шарів попередньо навченої НМ InceptionV3. Ці розподіли моделюються як багатовимірні нормальні, відстань між якими обчислюється за допомогою формули Фреше. Низьке значення FID свідчить про те, що згенеровані зображення є подібними до реальних як з точки зору візуального стилю, так і розподілу ознак.

Метрика IS, у свою чергу, оцінює якість та різноманіття згенерованих зображень, адже базується на ймовірнісних оцінках класифікаційної моделі InceptionV3, яка прогнозує належність кожного зображення до певного класу. Зростання значення IS означає, що ГМ поступово генерує зображення, які чітко розпізнаються та водночас належать до різних класів, тим самим дозволяючи виявити моделі, які схильні до надто одноманітної генерації або втрати чіткості.

Сам процес аналізу ґрунтується на тому, що протягом навчання моделей GAN, DDIM і VQ-GAN були виконані розрахунки значень відповідних метрик на конкретно визначених епохах тренування, надаючи змогу вивчити динаміку змін метричних показників за допомогою побудованих графіків.

Отже, розглядаючи результати метрики FID (представлені графіками у Додатку А), доцільно зазначити, що:

- модель GAN на початку тренування характеризується відносно високим значенням FID і демонструє поступове коливальне зниження показника протягом усього навчання, досягаючи мінімум у діапазоні 60-70 одиниць; це свідчить про те, що якість генерованих зображень та їх реалістичність поступово та стабільно покращуються з процесом тренування;

- модель DDIM демонструє найшвидше та найзначніше зниження FID на початкових етапах, після чого значення метрики стабілізується і продовжує коливатися в діапазоні 30-60 одиниць до кінця процесу тренування, досягаючи найнижчих значень серед усіх моделей; дані результати ілюструють, що модель швидко навчається генерувати зображення, які є найбільш схожими на реальні дані;

- модель VQ-GAN також показує зниження показника FID протягом навчання, але відбувається це повільніше, порівняно з іншими моделями, досягаючи мінімальних значень близько 100-120 одиниць після повного циклу тренування; таким чином, кінцеві значення метрики FID залишаються вищими, ніж у GAN та DDIM, вказуючи, що генеровані моделлю зображення можуть бути менш схожими на реальні.

Аналізуючи результати метрики IS (представлені графіками у Додатку Б) для розроблених ГМ, доцільно сказати, що для навчання моделей використовувався лише один єдиний датасет з однотипних зображень, через що параметр різноманіття надавав занижені оцінки метрики.

Незважаючи на це, можна зробити наступні висновки:

- модель GAN демонструє поступове зростання значення IS протягом перших епох навчання, після чого спостерігається певна стабілізація з наявними коливаннями до кінця процесу тренування; це означає, що після навчання модель здатна генерувати досить різноманітні та чіткі зображення, які можна розпізнати;

- модель DDIM показує швидке зростання IS на початкових етапах навчання, однак згодом спостерігається поступове зниження значень метрики; така динаміка може вказувати на те, що хоча модель швидко навчається генерувати

якісні на початковому етапі зображення, з часом різноманітність генерованих семплів може погіршуватися;

- модель VQ-GAN демонструє найбільш плавне та стабільне зростання значення IS протягом усього процесу навчання; не зважаючи на те, що максимальна оцінка метрики є нижчою порівняно з піковими значеннями для моделей GAN та DDIM, VQ-GAN показує стабільний процес навчання та поступове покращення якості й різноманітності генерованих зображень.

Таким чином, результати об'єктивного дослідження розроблених ГМ підтверджують висновки щодо виконаної візуальної оцінки зображень.

4.2 Аналіз проблем і труднощів програмної реалізації

Важливо, що неоднозначність представлених результатів генерування зображень (згідно до 4.1), може пояснюватися низкою виявлених технічних і практичних труднощів у процесі реалізації ГМ, пов'язаних як із обмеженнями обчислювальних ресурсів, так і зі складністю самих алгоритмів.

Насамперед, навчання моделей здійснювалося в СР, що надає лише обмежену кількість ресурсів оперативної та відеопам'яті графічного процесора, суттєво впливаючи на можливості тренування складних ГМ, таких як VQ-GAN та DDIM.

Обмеження ресурсів неодноразово призводили до вимушеного зменшення розмірів оброблюваних батчів, негативно впливаючи на стабільність і швидкість навчання моделей. Окремі тренувальні цикли переривалися через перевищення доступної пам'яті, тим самим змушуючи адаптувати параметри тренування, шукати компроміси між якістю й продуктивністю, та повторно запускати обчислення після збоїв. У випадку з VQ-GAN це стало однією з причин зниження якості згенерованих зображень.

Ще однією проблемою у ході реалізації виявилася тривалість навчання моделей, особливо для дифузійного підходу DDIM, де кожна епоха займала значний проміжок часу. Через складну послідовну природу генеративного процесу, а також через обчислювально-затратні зворотні проходи, модель вимагала великої

кількості епох для досягнення задовільної якості генерування зображень, таким чином ускладнюючи налаштування гіперпараметрів, оскільки навіть мінімальні зміни потребували годин тренування для оцінювання їх ефективності.

Крім технічних складнощів, сам процес програмної реалізації моделей обраних генеративних мереж виявився концептуально непростю задачею. Робота з VQ-GAN потребувала роздільного моделювання як частин автокодувальника, так і моделі латентної дифузії, а також синхронізованого підходу до тренування всіх компонентів. DDIM вимагала точного відтворення складних процедур прямої і зворотної дифузії, концепція яких базувалася на низці математичних обчислень. Навіть базова модель GAN вимагала налаштування операції коректного балансування між генератором і дискримінатором, аби уникнути колапсу режиму та нестабільної динаміки навчання.

4.3 Рекомендації щодо майбутнього вдосконалення

Проведені дослідження та результати реалізації генеративних моделей GAN, VQ-GAN та DDIM підтвердили їх ефективність у вирішенні задач генерації зображень. Утім, для подальшого вдосконалення, підвищення якості та універсальності пропонується реалізувати низку технічних удосконалень.

Насамперед, у поточній реалізації процедура навчання відбувалася на одному НД, у такий спосіб обмежуючи здатність моделей до генералізації та варіативності результатів. Відповідно, доцільно розширити набір навчальних даних, здійснюючи тренування на декількох різноманітних датасетах, які охоплюють ширший спектр візуальних характеристик, об'єктів та стилів, таким чином дозволяючи створювати більш різноманітні зображення.

Крім того, перспективним рішенням є інтеграція з мовними моделями, зокрема з використанням архітектури BERT, що забезпечить можливість генерації зображень на основі текстових описів. Такий підхід фактично надасть змогу для реалізації мультимодальних систем, де текст і візуальні дані поєднуються для генерування змістовних й осмислених зображень.

Окрему увагу варто приділити подальшій оптимізації архітектур моделей, зокрема за рахунок впровадження покращених блоків нормалізації, механізмів самоуваги, глибших латентних просторів та ефективніших стратегій навчання. Це дозволить суттєво підвищити якість згенерованих зображень, забезпечуючи кращу деталізацію, структурованість, чіткість та узгодженість.

Також перспективними є розробка та впровадження механізмів контрольованої генерації, даючи можливість задавати специфічні параметри, такі як колірна палітра, композиція, стиль, розмір об'єкта тощо.

Звісно, оптимізація обчислень та споживання ресурсів, шляхом впровадження оптимізацій на рівні обчислень (наприклад, через змішану точність, моделі-дистиляти або pruned-архітектури), дозволить ефективно використовувати ГМ навіть у середовищах з обмеженими обчислювальними можливостями, зокрема в мобільних або браузерних застосунках.

Таким чином, зазначені рішення щодо вдосконалення сприятимуть подальшому розвитку створених моделей, підвищуючи їх гнучкість, якість та функціональність та роблячи їх більш придатними для реального практичного застосування в різноманітних галузях – від науки до індустрії.

4.4 Передбачувані області використання

Незважаючи на можливі проблеми щодо реалізації та генерувальних можливостей, розроблені та досліджені моделі відкривають широкий спектр потенційних застосувань, що базуються на здатності генерувати нові, унікальні зразки даних, схожі на ті, на яких вони навчалися.

Наприклад, усі втілені моделі (GAN, DDIM і VQ-GAN) можуть бути використані для створення додаткових зображень, які розширюють існуючі набори, що вкрай необхідно в ситуаціях, коли збір великої кількості реальних даних є дорогим, трудомістким або неможливим процесом. Збільшені таким чином вибірки можуть покращити якість та стійкість інших моделей машинного навчання (наприклад, класифікаторів або детекторів об'єктів).

Крім того, створені моделі можуть слугувати основою для розробки графічного інструментарію для художників, дизайнерів та творців контенту в галузі інформаційної культури. Наприклад, для генерації унікальних текстур, фонів, абстрактних зображень, концепт-артів для ігор чи фільмів. У промисловому дизайні, архітектурі або розробці продуктів, покращені ГМ є здатними надавати можливість створювати візуальні прототипи різних концепцій, дозволяючи оцінити їхній вигляд без необхідності створення фізичних моделей, прискорюючи процес розробки та знижуючи ресурсні витрати.

Якщо моделі навчити на специфічних типах даних (медичні зображення, супутникові знімки або обличчя людей), вони потенційно можуть бути використані у відповідних вузькоспеціалізованих галузях. При цьому, ефективність тієї чи іншої моделі в кожній з областей буде залежати від якості та репрезентативності даних навчання, налаштованих гіперпараметрів, а також можливих подальших доопрацювань та оптимізацій архітектур мереж.

Важливо, що розроблені моделі є цінним ресурсом для подальших досліджень у галузі генеративного моделювання. Вони дозволяють вивчати особливості різних архітектур і методів навчання, їхній вплив на кінцевий результат, а також можуть використовуватися як навчальні інструменти для демонстрації можливостей ШІ для вирішення задачі генерування зображень.

ВИСНОВКИ

У межах виконаної роботи було здійснено комплексне дослідження проблематики та можливостей реалізації задачі генерації зображень із використанням сучасних моделей ШНМ.

Для досягнення поставленої мети було успішно виконано низку як теоретичних, так і практичних завдань. Зокрема, детально проаналізовано постановку задачі генерування зображень та фундаментальні основи функціонування генеративних нейронних мереж. Проведено дослідження сучасного стану інформаційних технологій у галузі створення ШНМ для генерації зображень, включаючи огляд існуючих програмних рішень та виявлення актуальної проблематики щодо їх втілення й використання. Особливу увагу було приділено математичному апарату та архітектурним рішенням низки обраних генеративних моделей.

У ході теоретичного дослідження було виявлено, що генеративні нейромережеві підходи не лише демонструють значний науковий і прикладний потенціал, але й стають дедалі доступнішими не тільки для широкого кола розробників і дослідників, але й для суспільства в цілому.

Ключовим етапом роботи стала розробка програмних реалізацій досліджуваних моделей генеративно-змагальних і дифузійних мереж на прикладах архітектур GAN, DDIM та VQ-GAN. У процесі враховувались етапи проєктування структури та механізми роботи, вибору та підготовки наборів даних, налаштування обчислювального середовища, а також процедури безпосереднього навчання та тестування розроблених моделей. Генеративні можливості щодо ефективності створення зображень були порівняні за допомогою як візуального аналізу, так і об'єктивних метрик.

Незважаючи на обмеженість ресурсів, реалізовані моделі продемонстрували здатність генерувати нові зразки, що передають ключові візуальні характеристики вхідного простору. Особливу ефективність, з точки зору якості результатів, продемонструвала латентна дифузійна модель DDIM.

Виконана робота демонструє, що навіть прості за побудовою генеративні моделі здатні створювати високоякісні та різноманітні зображення, відкриваючи широкі можливості для їх практичного застосування у різноманітних сферах життя. Додатково сформульовано рекомендації та визначено перспективні напрямки щодо майбутнього вдосконалення розроблених моделей.

У загальному підсумку, отримані результати підтверджують як теоретичну ефективність ШНМ у генеративних задачах, так і практичну доцільність їх використання та перспективність для створення зображень.

Таким чином, поставлена мета роботи – дослідження та глибинний аналіз способів розв’язання задачі генерації зображень засобами сучасних технологічних підходів у сфері ШНМ – реалізована у повному обсязі, а проведене дослідження закладає основу для подальшого розвитку та практичного впровадження генеративних підходів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Adobe Firefly. URL: <https://www.adobe.com/de/products/firefly.html>. Дата звернення: 12.03.25.
2. Aggarwal A., Mittal M., Battineni G. Generative adversarial network: An overview of theory and applications. International Journal of Information Management Data Insights. 2021. Vol. 1. P. 9. URL: <https://gclnk.com/GGA0Vgi3>.
3. An Overview of Diffusion Models: Applications, Guided Generation, Statistical Rates and Optimization / Chen M., Mei S., Fan J., Wang M. // arXiv preprint. URL: <https://arxiv.org/abs/2404.07771>. Дата звернення: 28.02.25.
4. Conditional Consistency Guided Image Translation and Enhancement // arXiv preprint. URL: <https://arxiv.org/abs/2501.01223>. Дата звернення: 07.03.25.
5. Conditional Image Generation with PixelCNN Decoders / A. Van den Oord et al. Advances in Neural Information Processing Systems 29 : 30th Conference on Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, 5–10 Dec. 2016 : in 7 pt. Curran Associates, Inc., 2017. Pt. 7. P. 4790–4798.
6. DALL-E 3. URL: <https://openai.com/index/dall-e-3/>. Дата звернення: 11.03.25.
7. Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis. Advances in Neural Information Processing Systems 34 : 35th Conference on Neural Information Processing Systems (NeurIPS 2021), 6–14 Dec. 2021 : in 36 pt. Curran Associates, Inc., 2022. Pt. 34. P. 8780–8794.
8. Diffusion Models: A Comprehensive Survey of Methods and Applications // arXiv preprint. URL: <https://arxiv.org/abs/2209.00796>. Дата звернення: 28.02.25.
9. Dual Codebook VQ: Enhanced Image Reconstruction with Reduced Codebook Size // arXiv preprint. URL: <https://arxiv.org/abs/2503.10832>. Дата звернення: 05.03.25.
10. Esser P., Rombach R., Ommer B. Taming Transformers for High-Resolution Image Synthesis. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2021), Nashville, USA, 20–25 Jun. 2021 : in 21 pt. Curran Associates, Inc., 2021. Pt. 15. P. 12873–12883.

11. Fast-Slow Co-advancing Optimizer: Toward Harmonious Adversarial Training of GAN // arXiv preprint. URL: <https://arxiv.org/abs/2504.15099>. Дата звернення: 09.03.25.
12. Generative Adversarial Nets / I. J. Goodfellow et al. Advances in Neural Information Processing Systems 27 : 28th Annual Conference on Neural Information Processing Systems (NeurIPS 2014), Montreal, Canada, 8–13 Dec. 2014 : in 4 pt. Curran Associates, Inc., 2015. Pt. 2. P. 2672–2680.
13. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA: MIT Press, 2016. 800 с.
14. High-Resolution Image Synthesis with Latent Diffusion Models / R. Rombach et al. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2022), New Orleans, USA, 18–24 Jun. 2022 : in 25 pt. Curran Associates, Inc., 2023. Pt. 10. P. 10684–10695.
15. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. Advances in Neural Information Processing Systems 33 : 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada, 6–12 Dec. 2020 : in 27 pt. Curran Associates, Inc., 2021. Pt. 24. P. 6840–6851.
16. Kingma D. P., Welling M. Auto-Encoding Variational Bayes // Proceedings of the ICLR 2014: 2nd International Conference on Learning Representations (14-16 April 2014, Banff, Canada). Banff, 2014. URL: <https://arxiv.org/abs/1312.6114>.
17. Liu J. Research on the Application of Variational Autoencoder in Image Generation. ITM Web of Conferences Volume 70, Nanjing, China, 6–8 Dec. 2024 / Ulster College, Shaanxi University of Science and Technology. EDP Sciences, 2025. P. 1–8.
18. Midjourney. URL: <https://www.midjourney.com/>. Дата звернення: 10.03.25.
19. Neural Network Diffusion // arXiv preprint. URL: <https://arxiv.org/abs/2402.13144>. Дата звернення: 23.02.25.
20. NICE: Non-linear independent components estimation // arXiv preprint. URL: <https://arxiv.org/abs/1410.8516>. Дата звернення: 02.03.25.
21. Object generation problem. URL: <https://gclnk.com/UCDl1Yrr>. Дата звернення: 20.02.25.

22. Radolf A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks// Proceedings of the ICLR 2016: 4th International Conference on Learning Representations (2-4 May 2016, San Juan, Puerto Rico). San Juan, 2022. URL: <https://arxiv.org/abs/1511.06434>.
23. Restructuring Vector Quantization with the Rotation Trick / Fifty C. et al. // Proceedings of the ICLR 2025: 13th International Conference on Learning Representations (24-28 April 2025, Singapore). Singapore, 2022. URL: <https://arxiv.org/pdf/2410.06424>.
24. SGC-VQGAN: Towards Complex Scene Representation via Semantic Guided Clustering Codebook // arXiv preprint. URL: <https://arxiv.org/abs/2409.06105>. Дата звернення: 05.03.25.
25. Song J., Meng C., Ermon S. Denoising Diffusion Implicit Models // Proceedings of the ICLR 2021: 9th International Conference on Learning Representations (4 May 2021, Vienna, Austria). Vienna, 2021. URL: <https://gclnk.com/n1mkypTg>.
26. Stable Diffusion. URL: <https://gclnk.com/weJDeu9C>. Дата звернення: 10.03.25.
27. Van den Oord A., Kalchbrenner N., Kavukcuoglu K. Pixel Recurrent Neural Networks. Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML 2016), New York City, USA, 19–24 Jun. 2016 : in 6 pt. Curran Associates, Inc., 2016. Pt. 4. P. 3477–3486.
28. Van den Oord A., Vinyals O., Kavukcuoglu K. Neural Discrete Representation Learning. Advances in Neural Information Processing Systems 30 : 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, USA, 4–9 Dec. 2017 : in 10 pt. Curran Associates, Inc., 2018. Pt. 9. P. 6306–6315.
29. Yan W. BigGAN: Advances, Applications, and Challenges in Generative Image Processing. Highlights in Science, Engineering and Technology : 3rd International Conference on Computer, Machine Learning and Artificial Intelligence (CMLAI 2025), San Francisco, USA, 29–30 May 2025 : in 138 pt. Darcy & Roy Press, 2025. Pt. 138.
30. Yu J., Li X., Koh J. Y. Vector-Quantized Image Modeling with Improved VQGAN // Proceedings of the ICLR 2022: 10th International Conference on Learning Representations (25-29 April 2022). 2022. URL: <https://gclnk.com/VSOmu9uM>.

ДОДАТОК А. РЕЗУЛЬТАТИ ОЦІНКИ FID-МЕТРИКИ

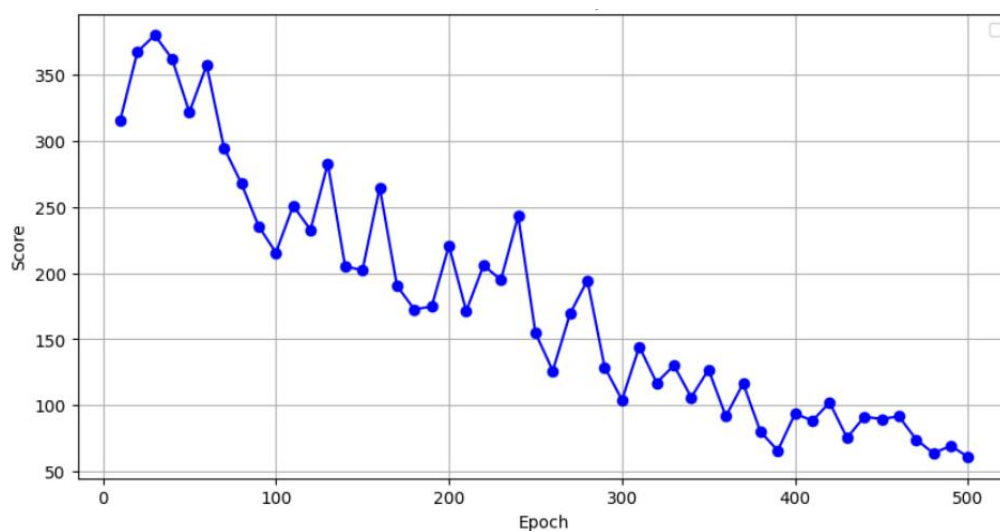


Рисунок А.1 – Графік оцінки FID-метрики моделі GAN протягом навчання

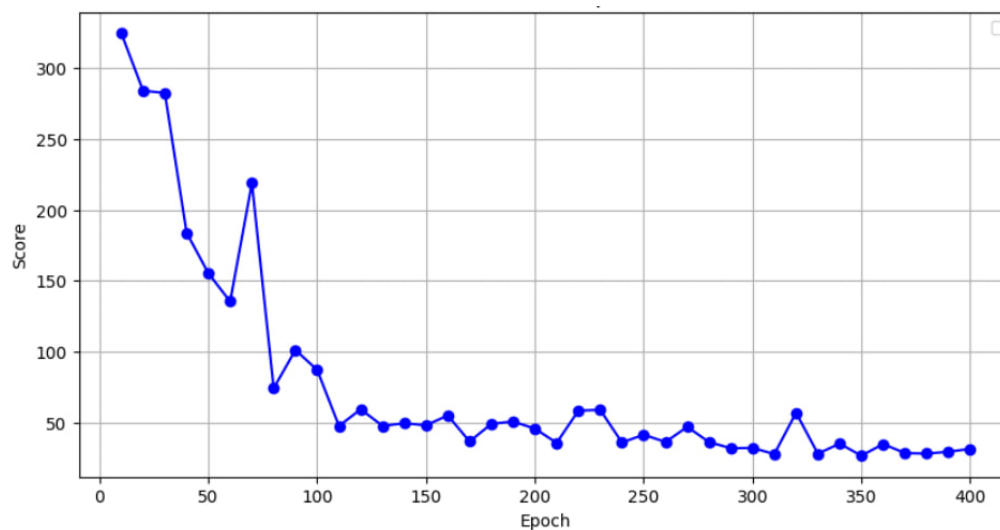


Рисунок А.2 – Графік оцінки FID-метрики моделі DDIM протягом навчання

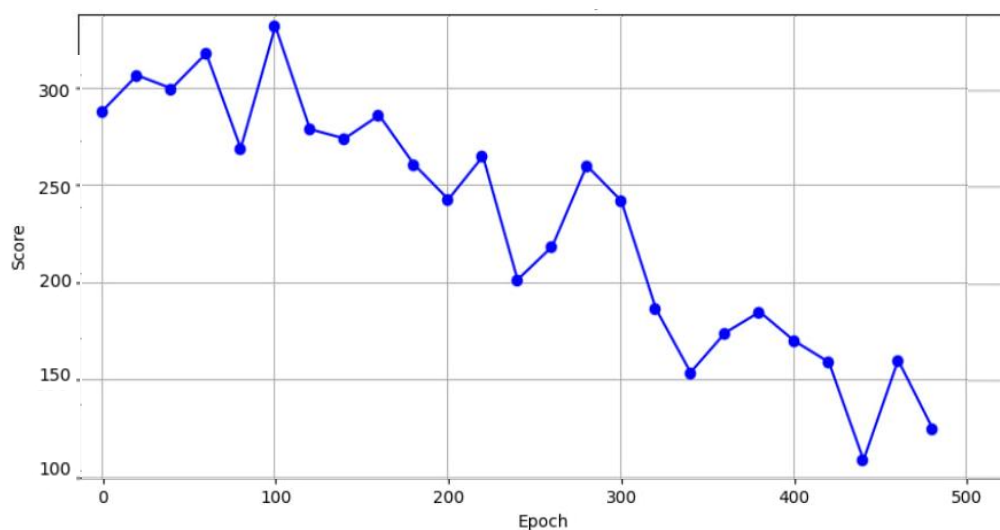


Рисунок А.3 – Графік оцінки FID-метрики моделі VQ-GAN протягом навчання

ДОДАТОК Б. РЕЗУЛЬТАТИ ОЦІНКИ IS-МЕТРИКИ

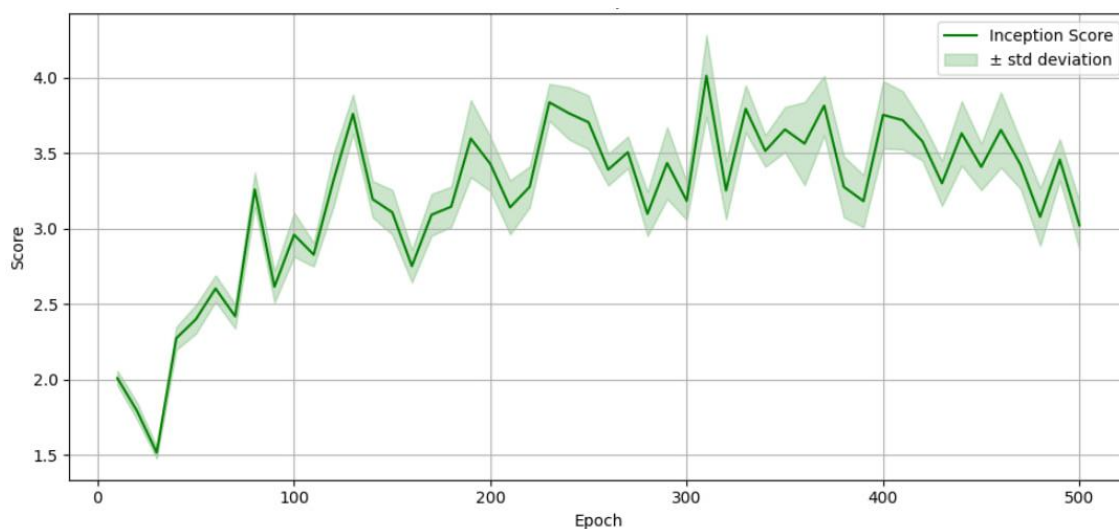


Рисунок Б.1 – Графік оцінки IS-метрики моделі GAN протягом навчання

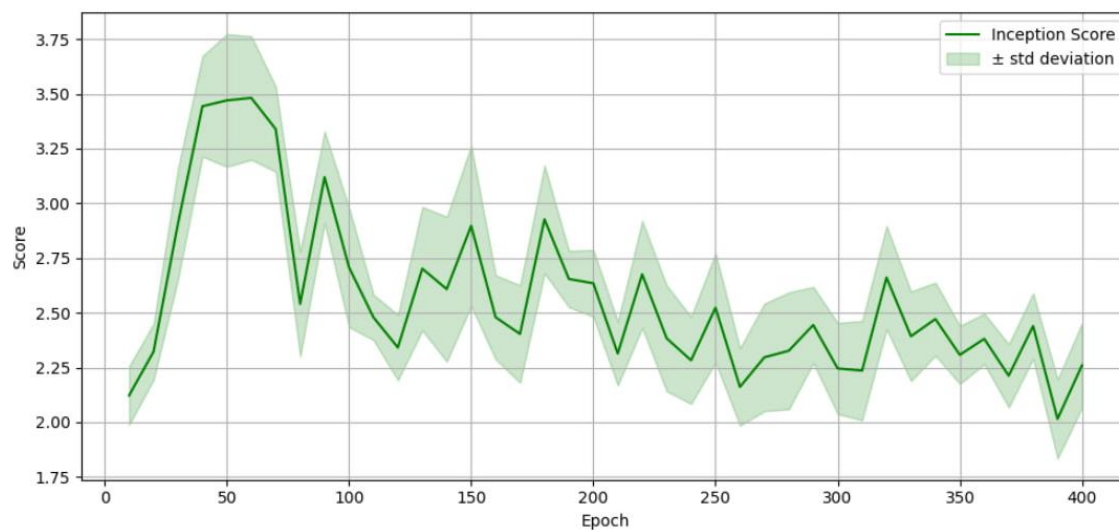


Рисунок Б.2 – Графік оцінки IS-метрики моделі DDIM протягом навчання

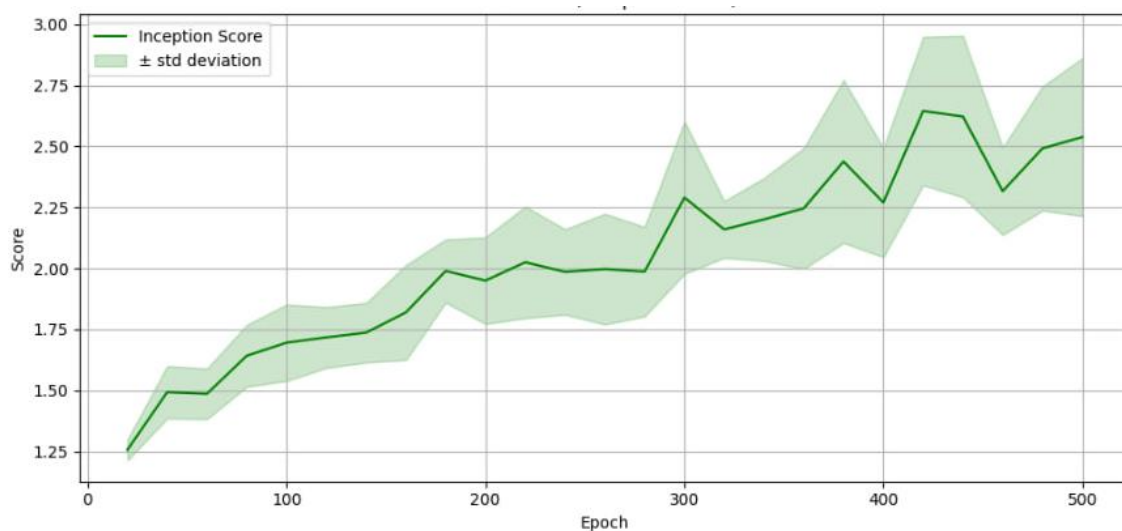


Рисунок Б.3 – Графік оцінки IS-метрики моделі VQ-GAN протягом навчання

ДОДАТОК В. РЕЗУЛЬТАТИ ОЦІНКИ LOSS-МЕТРИКИ

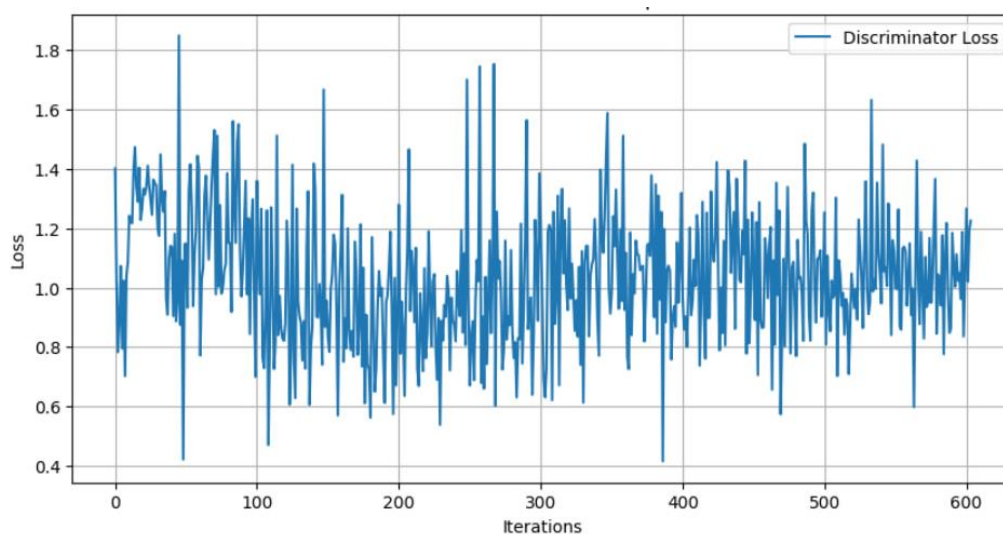


Рисунок В.1 – Графік оцінки LOSS-метрики моделі GAN протягом навчання

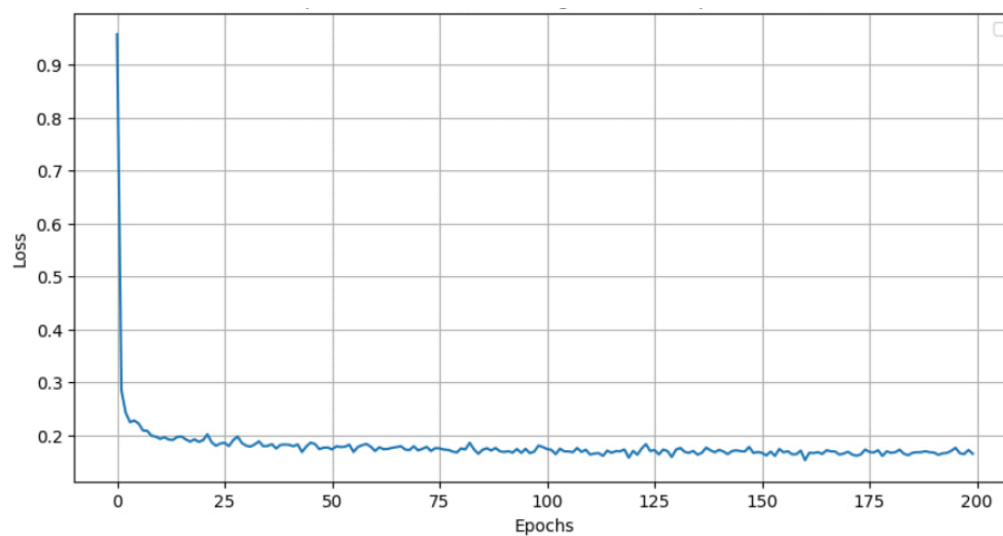


Рисунок В.2 – Графік оцінки LOSS-метрики моделі DDIM протягом навчання

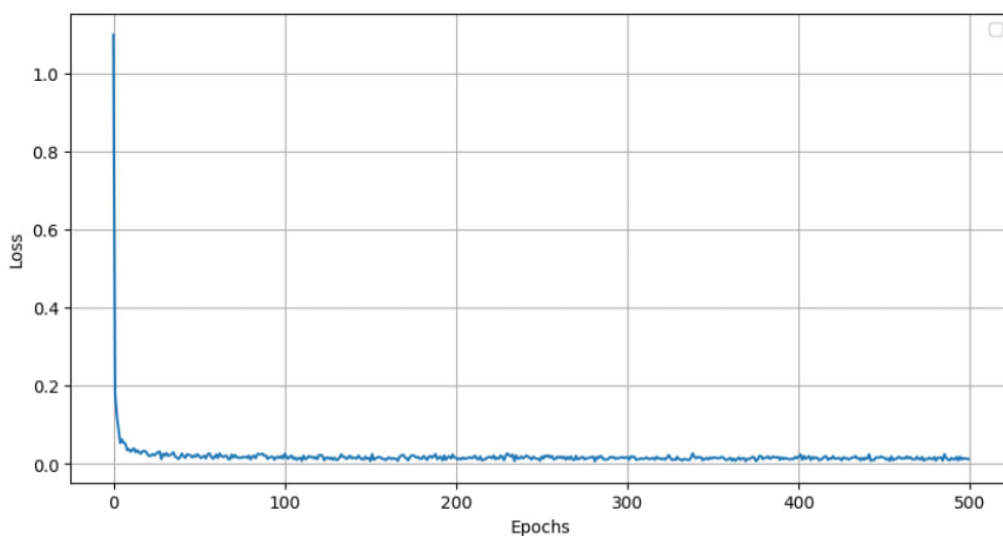


Рисунок В.3 – Графік оцінки LOSS-метрики моделі VQ-GAN протягом навчання

ДОДАТОК Г. ВИХІДНИЙ ПРОГРАМНИЙ КОД МОДЕЛІ ГЕНЕРАТИВНО-ЗМАГАЛЬНОЇ МЕРЕЖІ (GAN)

Додаток Г присвячено лістингам програмних методів і класів, які були розроблені для реалізації генеративної моделі GAN.

Лістинг Г.1 – Побудова генератора

```
# Вхідний шар -> передаємо на вхід мережі випадкові дані
inputs = Input(shape=(128,))

x = Dense(8*8*256, use_bias=False)(inputs)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Reshape((8, 8, 256))(x)
x = Conv2DTranspose(256, 3, strides=(1, 1), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2DTranspose(128, 3, strides=(2, 2), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2DTranspose(128, 3, strides=(1, 1), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2DTranspose(64, 3, strides=(2, 2), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2DTranspose(64, 3, strides=(1, 1), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2DTranspose(32, 3, strides=(2, 2), padding='same',
use_bias=False)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

# Вихідний шар
x = Conv2DTranspose(3, 3, strides=(1, 1), padding='same',
use_bias=False, activation='sigmoid')(x)
```

Закінчення Лістингу Г.1

```
# sigmoid -> вихідне значення знаходиться в діапазоні [0; 1]

# Збірка мережі
outputs = x
generator = keras.Model(inputs, outputs)
```

Лістинг Г.2 – Побудова дискримінатора

```
# Вхідний шар
inputs = Input(shape=(64, 64, 3))

x = Conv2D(64, 3, strides=(2, 2), padding='same')(inputs)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

x = Conv2D(64, 3, strides=(1, 1), padding='same')(inputs)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

x = Conv2D(128, 3, strides=(2, 2), padding='same')(x)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

x = Conv2D(128, 3, strides=(2, 2), padding='same')(x)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

x = Conv2D(128, 3, strides=(2, 2), padding='same')(x)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

# Перетворення багатовимірної тензора на одновимірний вектор
x = Flatten()(x)
x = Dense(64)(x)
x = LeakyReLU()(x)
x = Dropout(0.2)(x)

x = Dense(1, activation = 'sigmoid')(x)
# sigmoid -> вихідне значення знаходиться в діапазоні [0; 1]

# Збірка мережі
outputs = x
discriminator= keras.Model(inputs, outputs)
```

Лістинг Г.3 – Збірка моделі мережі (клас Model)

```
class Model(tf.keras.Model):
    # функція ініціалізації
    def __init__(self, gen , disc):
        super(Model, self).__init__()
```

Продовження Лістингу Г.3

```

self.gen = gen      # генератор
self.disc = disc    # дискримінатор

self.gen_optimizer = tf.keras.optimizers.Adam(5e-5)  #
оптимізатор для генератора
self.disc_optimizer = tf.keras.optimizers.Adam(5e-5)  #
оптимізатор для дискримінатора

# функція реалізації генеративної помилки
def gan_loss(self, logits, is_real):
    if is_real:
        labels = tf.ones_like(logits)
    else:
        labels = tf.zeros_like(logits)
    loss =
tf.keras.losses.BinaryCrossentropy(from_logits=True)(labels, logits)

    return loss # помилка

# функція навчання моделі
@tf.function
def training_step(self, x ):

    # генеруємо випадкові цифрові дані (шум) для генератора
    noise = tf.random.normal([bs, 128], 0, 1, tf.float32)
    # релізуємо дві градієнтні стрічки для генератора та
дискримінатора
    with tf.GradientTape() as disc_tape , tf.GradientTape() as
gen_tape:

        # генеруємо зображення завдяки генератору
        fake_images = self.gen(noise, training=True)
        real_images = x

        # розраховуємо передбачення завдяки дескримінатору для
фейкових і реальних зображень
        fake_result = self.disc(fake_images, training=True)
        real_result = self.disc(real_images, training=True)

        # розраховуємо генеративну помилку
        disc_fake_loss = self.gan_loss(fake_result, False) # ->
реальне зображення
        disc_real_loss = self.gan_loss(real_result, True)  # ->
згенероване зображення

        # розраховуємо помилку для дискримінатора
        disc_loss = disc_fake_loss + disc_real_loss

        # розраховуємо помилку для генератора
        gen_loss = self.gan_loss(fake_result, True)

```

Закінчення Лістингу Г.3

```
# розраховуємо градієнти для генератора та дескрімінатора
disc_gradients = disc_tape.gradient(disc_loss,
self.disc.trainable_variables)
self.disc_optimizer.apply_gradients(zip(disc_gradients,
self.disc.trainable_variables))

gen_gradients = gen_tape.gradient(gen_loss,
self.gen.trainable_variables)
self.gen_optimizer.apply_gradients(zip(gen_gradients,
self.gen.trainable_variables))

return tf.reduce_mean(disc_loss) # повертаємо помилку
```

ДОДАТОК Д. ВИХІДНИЙ ПРОГРАМНИЙ КОД НЕЯВНОЇ ДИФУЗІЙНОЇ МОДЕЛІ З УСУНЕННЯМ ШУМУ (DDIM)

Додаток Д присвячено лістингам програмних методів і класів, які були розроблені для реалізації генеративної моделі DDIM.

Лістинг Д.1 – Метод ResidualBlock

```
def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = layers.Conv2D(width, kernel_size=1)(x)
            x = layers.BatchNormalization(center=False, scale=False)(x)
            x = layers.Conv2D(width, kernel_size=3, padding="same",
activation=keras.activations.swish)(x)
            x = layers.Conv2D(width, kernel_size=3, padding="same")(x)
            x = layers.Add()([x, residual])
        return x
    return apply
```

Лістинг Д.2 – Метод DownBlock

```
def DownBlock(width, block_depth):
    def apply(x):
        x, skips, emb = x
        height = x.shape[1]
        # використовуємо часовий "embedding" разом із конкантинацією
зображення
        e = layers.UpSampling2D(size=height,
interpolation="nearest")(emb)
        x = Concatenate()([x, e])
        for _ in range(block_depth):
            x = ResidualBlock(width)(x)
            skips.append(x)
        x = layers.AveragePooling2D(pool_size=2)(x)
        return x
    return apply
```

Лістинг Д.3 – Метод UpBlock

```
def UpBlock(width, block_depth):
    def apply(x):
        x, skips = x
        x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)
```

```

        for _ in range(block_depth):
            x = layers.Concatenate()([x, skips.pop()])
            x = ResidualBlock(width)(x)
        return x
    return apply

```

Лістинг Д.4 – Побудова мережі U-Net

```

def get_network(image_size, widths, block_depth):
    noisy_images = keras.Input(shape=(image_size, image_size, 3))
    noise_variances = keras.Input(shape=(1, 1, 1))

    x = noisy_images
    emb = layers.Lambda(sinusoidal_embedding)(noise_variances)

    skips = []

    for width in widths[:-1]:
        x = DownBlock(width, block_depth)([x, skips, emb])

    for _ in range(block_depth):
        x = ResidualBlock(widths[-1])(x)

    for width in reversed(widths[:-1]):
        x = UpBlock(width, block_depth)([x, skips])

    # додаємо шари для кращої чіткості моделі
    e = layers.UpSampling2D(size=image_size,
interpolation="nearest")(emb)
    x = layers.Concatenate()([x, noisy_images, e])
    x = layers.Conv2D(32, kernel_size=1, padding = 'same',
activation=keras.activations.swish )(x)

    x = layers.Conv2D(3, kernel_size=1,
kernel_initializer="zeros")(x)

    # збираємо модель
    return keras.Model([noisy_images, noise_variances], x,
name="residual_unet")

```

Лістинг Д.5 – Збірка моделі мережі (клас DiffusionModel)

```

class DiffusionModel(keras.Model):
    def __init__(self, image_size, widths, block_depth):
        super().__init__()

        self.normalizer = layers.Normalization() # шар нормалізатора
        self.network = get_network(image_size, widths, block_depth)
        self.optimizer = optimizer = AdamW(learning_rate=1e-4,
weight_decay=1e-4)

```

Продовження Лістингу Д.5

```

def denormalize(self, images):
    images = self.normalizer.mean + images *
self.normalizer.variance**0.5
    return tf.clip_by_value(images, 0.0, 1.0)

def diffusion_schedule(self, diffusion_times):
    start_angle = tf.acos(max_signal_rate)
    end_angle = tf.acos(min_signal_rate)

    diffusion_angles = start_angle + diffusion_times *
(end_angle - start_angle)

    signal_rates = tf.cos(diffusion_angles)
    noise_rates = tf.sin(diffusion_angles)
    # для нормальної роботи моделі:  $\sin^2(x) + \cos^2(x) = 1$ 

    return noise_rates, signal_rates

def denoise(self, noisy_images, noise_rates, signal_rates):
    pred_noises = self.network([noisy_images, noise_rates**2])
    pred_images = (noisy_images - noise_rates * pred_noises) /
signal_rates

    return pred_noises, pred_images

def reverse_diffusion(self, initial_noise, diffusion_steps):
    num_images = initial_noise.shape[0]
    step_size = 1.0 / diffusion_steps

    next_noisy_images = initial_noise

    for step in range(diffusion_steps):
        noisy_images = next_noisy_images

        diffusion_times = tf.ones((num_images, 1, 1, 1)) - step
* step_size

        noise_rates, signal_rates =
self.diffusion_schedule(diffusion_times)

        pred_noises, pred_images = self.denoise(noisy_images,
noise_rates, signal_rates)

        next_diffusion_times = diffusion_times - step_size

        next_noise_rates, next_signal_rates =
self.diffusion_schedule(next_diffusion_times)

        next_noisy_images = (next_signal_rates * pred_images +
next_noise_rates * pred_noises)

```

Закінчення Лістингу Д.5

```

    return pred_images

    def generate(self, num_images, diffusion_steps):
        initial_noise = tf.random.normal(shape=(num_images,
image_size, image_size, 3))
        generated_images = self.reverse_diffusion(initial_noise,
diffusion_steps)
        generated_images = self.denormalize(generated_images)
        return generated_images

    @tf.function
    def train_step(self, images):
        images = self.normalizer(images, training=True)
        noises = tf.random.normal(shape=(batch_size, image_size,
image_size, 3))

        diffusion_times = tf.random.uniform(shape=(batch_size, 1, 1,
1), minval=0.0, maxval=1.0)
        noise_rates, signal_rates =
self.diffusion_schedule(diffusion_times)

        noisy_images = signal_rates * images + noise_rates * noises

        with tf.GradientTape() as tape:

            pred_noises, pred_images = self.denoise(noisy_images,
noise_rates, signal_rates)

            noise_loss = tf.reduce_mean(tf.abs(noises -
pred_noises), axis = [1,2,3])

            gradients = tape.gradient(noise_loss,
self.network.trainable_weights)
            self.optimizer.apply_gradients(zip(gradients,
self.network.trainable_weights))

        return noise_loss

# Збірка мережі для генерації зображень
model = DiffusionModel(image_size, widths, block_depth)

```

ДОДАТОК Е. ВИХІДНИЙ ПРОГРАМНИЙ КОД МОДЕЛІ ВЕКТОРНО-КВАНТОВОЇ МЕРЕЖІ (VQ-GAN)

Додаток Е присвячено лістингам програмних методів і класів, які були розроблені для реалізації генеративної моделі VQ-GAN.

Лістинг Е.1 – Клас VectorQuantizer

```
class VectorQuantizer(layers.Layer):
    def __init__(self, num_embeddings, embedding_dim, beta=0.25,
**kwargs):
        super().__init__(**kwargs)
        self.embedding_dim = embedding_dim
        self.num_embeddings = num_embeddings
        self.beta = beta

        w_init = tf.random_uniform_initializer()
        self.embeddings = tf.Variable(
            initial_value=w_init(
                shape=(self.embedding_dim, self.num_embeddings),
dtype="float32"
            ),
            trainable=True,
            name="embeddings_vqvae",
        )

    def call(self, x):
        input_shape = tf.shape(x)
        flattened = tf.reshape(x, [-1, self.embedding_dim])

        # виконуємо процес квантування
        encoding_indices = self.get_code_indices(flattened)
        encodings = tf.one_hot(encoding_indices,
self.num_embeddings)
        quantized = tf.matmul(encodings, self.embeddings,
transpose_b=True)

        quantized = tf.reshape(quantized, input_shape)
        commitment_loss =
tf.reduce_mean((tf.stop_gradient(quantized) - x) ** 2)
        codebook_loss = tf.reduce_mean((quantized -
tf.stop_gradient(x)) ** 2)
        self.add_loss(self.beta * commitment_loss + codebook_loss)

        quantized = x + tf.stop_gradient(quantized - x)
        return quantized

    def get_code_indices(self, flattened_inputs):
        similarity = tf.matmul(flattened_inputs, self.embeddings)
```

Закінчення Лістингу E.1

```

        distances = (
            tf.reduce_sum(flattened_inputs ** 2, axis=1,
keepdims=True)
            + tf.reduce_sum(self.embeddings ** 2, axis=0)
            - 2 * similarity
        )
        encoding_indices = tf.argmin(distances, axis=1)
        return encoding_indices

# фрагмент коду для збереження моделі у форматі .h5
def get_config(self):
    config = super().get_config()
    config.update({
        "arg1": self.num_embeddings,
        "arg2": self.embedding_dim,
        "arg3": self.beta,
    })
    return config

```

Лістинг E.2 – Метод ResidualBlock

```

def ResidualBlock(width):
    def apply(x):
        input_width = x.shape[3]
        if input_width == width:
            residual = x
        else:
            residual = layers.Conv2D(width, kernel_size=1,
activation=keras.activations.swish)(x)
            x = layers.Conv2D(width // 2, kernel_size=1,
activation=keras.activations.swish)(x)
            x = layers.Conv2D(width // 2, kernel_size=3, padding="same",
activation=keras.activations.swish)(x)
            x = layers.Conv2D(width, kernel_size=1,
activation=keras.activations.swish)(x)
            x = layers.Add()([x, residual])
        return x
    return apply

```

Лістинг E.3 – Побудова кодувальника

```

def get_encoder(image_size):
    image = keras.Input(shape=(image_size, image_size, 3))

    x = image

    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = layers.AveragePooling2D(pool_size=2)(x)

```

Закінчення Лістингу Е.3

```

x = ResidualBlock(128)(x)
x = ResidualBlock(128)(x)
x = ResidualBlock(128)(x)
x = ResidualBlock(128)(x)
x = layers.AveragePooling2D(pool_size=2)(x)

x = ResidualBlock(256)(x)
x = ResidualBlock(256)(x)
x = ResidualBlock(256)(x)
x = ResidualBlock(256)(x)
x = ResidualBlock(256)(x)
x = ResidualBlock(256)(x)

# додаємо шар, що "сильно" зменшує кіл-ть фільтрів
x = layers.Conv2D(4, kernel_size=1,
activation=keras.activations.swish)(x)

# додаємо шар квантування
x = vq_layer(x)

return keras.Model(image, x, name="encoder")

```

Лістинг Е.4 – Побудова декодувальника

```

def get_decoder(latent_size):
    latent = keras.Input(shape=(latent_size, latent_size, 4))

    x = latent
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = ResidualBlock(256)(x)
    x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)

    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = ResidualBlock(128)(x)
    x = layers.UpSampling2D(size=2, interpolation="bilinear")(x)

    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)
    x = ResidualBlock(64)(x)

    # перетворюємо дані знову в картинку
    x = layers.Conv2D(3, kernel_size=1, activation="sigmoid")(x)

    return keras.Model(latent, x, name="decoder")

```

Лістинг Е.5 – Побудова дискримінатора

```
def get_discriminator(image_size, discriminator_widths):
    image = keras.Input(shape=(image_size, image_size, 3))

    x = image

    for width in discriminator_widths:
        x = ResidualBlock(width)(x)
        x = layers.AveragePooling2D(pool_size=2)(x)
        x = ResidualBlock(width)(x)

    x = keras.layers.GlobalAveragePooling2D()(x)
    x = Flatten()(x)
    x = Dropout(0.3)(x)
    x = Dense(1, activation='sigmoid')(x)

    return keras.Model(image, x, name="discriminator")
```

Лістинг Е.6 – Збірка моделі мережі (клас Model)

```
class Model(keras.Model):
    def __init__(self, image_size, latent_size):
        super().__init__()

        self.encoder = get_encoder(image_size)
        self.decoder = get_decoder(latent_size)
        self.discriminator = get_discriminator(image_size, [32, 64,
96, 128, 256])

        self.ema_encoder = get_encoder(image_size)
        self.ema_decoder = get_decoder(latent_size)
        self.ema_encoder.set_weights(self.encoder.get_weights())
        self.ema_decoder.set_weights(self.decoder.get_weights())
        self.ema = 0.999

        self.lr = 1e-4
        self.enc_optimizer = tf.keras.optimizers.Adam(self.lr,
clipnorm=1.0)
        self.dec_optimizer = tf.keras.optimizers.Adam(self.lr,
clipnorm=1.0)
        self.disc_optimizer = tf.keras.optimizers.Adam(self.lr,
clipnorm=1.0)

    def gan_loss(self, logits, is_real):
        if is_real:
            labels = tf.ones_like(logits)
        else:
            labels = tf.zeros_like(logits)
        loss =
tf.keras.losses.BinaryCrossentropy(from_logits=True)(labels, logits)
        return loss
```

Продовження Лістингу E.6

```

@tf.function
def train_step(self, x):
    with tf.GradientTape() as tape_e, tf.GradientTape() as
tape_d, tf.GradientTape() as tape_disc:

        latent = self.encoder(x, training=True)

        reconstructions = self.decoder(latent, training=True)

        # Додаємо loss кодової книги, щоб усі значення також
навчалися та "підстраювалися"
        reconstruction_loss = tf.reduce_mean((x -
reconstructions) ** 2, axis = (1,2,3)) + sum(self.encoder.losses)

        gan_mode = True

        if gan_mode:
            fake_result = self.discriminator(reconstructions,
training=True)
            real_results = self.discriminator(x, training=True)

            gen_fake_loss = self.gan_loss(fake_result, True)

            disc_fake_loss = self.gan_loss(fake_result, False)
            disc_real_loss = self.gan_loss(real_results, True)

            total_loss = reconstruction_loss + gen_fake_loss/100

            disc_loss = disc_real_loss + disc_fake_loss

        else:
            total_loss = reconstruction_loss

        grads_e = tape_e.gradient(total_loss,
self.encoder.trainable_variables)
        self.enc_optimizer.apply_gradients(zip(grads_e,
self.encoder.trainable_variables))

        grads_d = tape_d.gradient(total_loss,
self.decoder.trainable_variables)
        self.dec_optimizer.apply_gradients(zip(grads_d,
self.decoder.trainable_variables))

        # Оновлення ЕМА-ваг

        for weight, ema_weight in zip(self.encoder.weights,
self.ema_encoder.weights):
            ema_weight.assign(self.ema * ema_weight + (1 - self.ema)
* weight)

```

Закінчення Лістингу E.6

```
        for weight, ema_weight in zip(self.decoder.weights,
self.ema_decoder.weights):
            ema_weight.assign(self.ema * ema_weight + (1 - self.ema)
* weight)

        if gan_mode:
            grads_disc = tape_disc.gradient(disc_loss,
self.discriminator.trainable_variables)
            self.disc_optimizer.apply_gradients(zip(grads_disc,
self.discriminator.trainable_variables))

        return total_loss
```