Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра _____Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
__Методи оптимізації генерації зображень за допомогою нейронних мереж__
_____
(тема)

Виконав: здобувач _другого_ року навчання,
групи ___СКСм-23-1_____
_____Фесенко А.В._____
(прізвище, ініціали)

Спеціальність _____
_____123 – Комп'ютерна інженерія_____
(код і повна назва спеціальності)

Тип програми ____освітньо-професійна____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____
___Спеціалізовані комп'ютерні системи___
(повна назва освітньої програми)

Керівник _____доц. Рожнова Т.Г._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ ___Чумаченко С.В.___
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_02_» 09 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Фесенко Антону Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи оптимізації генерації зображень за допомогою нейронних мереж

затверджена наказом університету від _08_ 11 2024 р. № _1189 Ст_

2. Термін подання здобувачем роботи до екзаменаційної комісії _24_ 01 2025 р.

3. Вихідні дані до роботи
Мова програмування Python
Бібліотеки PyTorch та Scikit-Optimize
Датасет MNIST
Інструмент TensorBoard
Метрика FID

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз методів генерації зображень за допомогою нейронних мереж
Огляд методів оптимізації генеративного процесу
Розробка системи з використанням оптимізаційних методів
Експериментальне тестування системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
18 слайдів (формату .pptx)

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	02.09.2024-05.09.2024	
2	Аналіз предметної області	06.09.2024-30.09.2024	
3	Аналіз джерел з проблемної галузі	01.10.2024-07.10.2024	
4	Розробка архітектури нейронної мережі для генерації зображень	08.10.2024-15.10.2024	
5	Підбір навчальної вибірки для проведення дослідження	16.10.2024-20.10.2024	
6	Написання програмної реалізації розробленої моделі нейронних мереж	21.10.2024-25.11.2024	
7	Проведення дослідження ефективності розробленої моделі	26.11.2024-05.12.2024	
8	Оформлення пояснювальної записки	06.12.2024-29.12.2024	
9	Оформлення графічного матеріалу	30.12.2024-07.01.2025	
10	Перевірка виконаного проекту керівником	08.01.2024-15.01.2025	
11	Подання роботи до ЕК для захисту	21.01.2025-24.01.2025	

Дата видачі завдання 02.09.2024

Здобувач _____
(підпис)

Керівник роботи _____ доц. Рожнова Т.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 54 сторінки, 20 рисунків, 19 джерел за переліком посилань.

ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ, НЕЙРОННІ МЕРЕЖІ, TEXT-TO-IMAGE, DIFFUSION, GAN, DCGAN, FID, ЗАПИТ, ОПТИМІЗАЦІЯ, ЯКІСТЬ ЗОБРАЖЕННЯ, PYTHON, PYTORCH, MNIST, TENSORBOARD, SKOPT, РОЗРОБКА І ТЕСТУВАННЯ.

Метою роботи є дослідження методів оптимізації генерації зображень за допомогою нейронних мереж.

Об’єкт дослідження – процеси генерації зображень з використанням генеративних нейронних мереж та методів оптимізації.

Предмет дослідження – методи вдосконалення генеративних нейронних мереж для підвищення ефективності навчання, та якості й варіативності генерації зображень.

Розглянуто відомі архітектури генеративних нейронних мереж, такі як GAN, Diffusion й DCGAN, та їх вплив на точність і швидкодію генерації зображень. Проведено дослідження методів покращення процесу генерації, таких як додавання нарисів до зображень, додаткове налаштування запитів та оптимізація архітектури нейронної мережі.

В середовищі виконання коду розроблено скрипт мовою Python з використанням різноманітних методів оптимізації генерації зображень. Проведено експериментальний аналіз графічних та чисельних результатів розробленої програми.

ABSTRACT

The explanatory note contains 54 pages, 20 figures, 19 sources according to the list of links.

IMAGE GENERATION, NEURAL NETWORKS, TEXT-TO-IMAGE, DIFFUSION, GAN, DCGAN, FID, PROMPT, OPTIMIZATION, IMAGE QUALITY, PYTHON, PYTORCH, MNIST, TENSORBOARD, SKOPT, DEVELOPMENT AND TESTING.

The purpose of the work is to study methods for optimizing image generation using neural networks.

The object of the study is image generation processes using generative neural networks and optimization methods.

The subject of the study is methods of improving generative neural networks to increase the efficiency of training, as well as the quality and variability of image generation.

A study of the well-known architectures of generative neural networks, such as GAN, Diffusion and DCGAN, and their impact on the accuracy and speed of image generation was performed. Methods for improving the generation process were examined, such as adding sketches to images, additional query tuning, and optimizing the neural network architecture.

A Python script was developed in the code execution environment using various methods of optimizing image generation. An experimental analysis of the graphical and numerical results of the developed program was performed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TTI – Text-To-Image – текст у зображення

GAN – Generative Adversarial Network – генеративно-змагальні мережі

DM – Diffusion Model – диффузійні моделі

ELBO – Evidence Lower Bound – нижня варіаційна межа

DF-GAN – Deep Fusion Generative Adversarial Networks – глибокий синтез генеративно-змагальної мережі

DF-Block – Deep text-image Fusion Block – блок глибокого синтезу

AttnGAN – Attentional Generative Adversarial Network – уважна генеративно-змагальна мережа

LSTM – Long Short-Term Memory – довготривала короткочасна пам’ять

MA-GP – Matching-Aware Gradient Penalty – градієнтний штраф з урахуванням відповідності

PAE – Prompt Auto-Editing – автоматичне редагування запиту

DF-Prompt – Dynamic Fine-control Prompt – динамічне налаштування запиту

DiffMorph – Diffusion Morph – диффузійне перетворення

KL – розходження Кульбака-Лейблера

DCGAN – Deep Convolutional Generative Adversarial Network – глибокі конволюційні генеративно-змагальні мережі

FML – Feature Matching Loss – втрата відповідності ознак

SAGAN – Self-Attention Generative Adversarial Network – генеративно-змагальні мережі самоуваги

FID – Fréchet Inception Distance – початкова відстань Фреше

ВСТУП

В останні роки спостерігається стрімкий розвиток технологій штучного інтелекту, зокрема, в галузі генеративних нейронних мереж. Ці технології знаходять своє застосування у багатьох сферах, таких як створення зображень, відео, аудіо, та навіть тексту. Особливу роль вони відіграють у задачах автоматизації творчих процесів, де традиційні підходи можуть виявитися менш ефективними або занадто трудомісткими. Генеративні моделі стають інструментами, що сприяють розширенню можливостей людини в різноманітних креативних сферах.

Сучасні генеративні нейронні мережі можуть навчатися на величезних масивах даних, що дозволяє створювати візуально привабливі та реалістичні зображення. Це є основою для інноваційних рішень, які можуть змінити підходи до проектування та візуалізації. Наприклад, такі моделі здатні адаптуватися під конкретні вимоги користувача, надаючи можливість створення персоналізованого контенту з високим рівнем відповідності вимогам. Крім того, вони активно використовуються в наукових дослідженнях для моделювання складних систем, аналізу медичних зображень та симуляції природних процесів. Широкий спектр застосувань робить генеративні моделі одним із ключових інструментів сучасної науки і техніки.

Використання цих технологій у повсякденному житті також стає все більш реальним завдяки зростаючій доступності обчислювальних ресурсів. Сьогодні високопродуктивне обладнання, яке раніше було доступне лише великим корпораціям або дослідницьким установам, стає доступним для звичайних користувачів. Це відкриває можливості для інтеграції генеративних моделей у програми масового використання, такі як фоторедактори, системи створення цифрового контенту та мобільні додатки. Однак, разом із доступністю виникають і нові виклики, пов’язані з оптимізацією стабільності навчання та якістю результатів генерації.

Проте, незважаючи на значні досягнення у цій галузі, існує низка проблем, які потребують подальшого дослідження та оптимізації. Однією з таких проблем є підвищення точності та стабільності генерації зображень, що може бути досягнуто через оптимізацію архітектури нейронних мереж, а також методів навчання та обробки даних.

Метою кваліфікаційної роботи є дослідження сучасних методів оптимізації генерації зображень за допомогою нейронних мереж, огляд ефективних алгоритмів та експериментальне оцінювання їх впливу на якість і стабільність процесу генерації. Особливу увагу приділено впровадженню таких методів, як Minibatch Discrimination, Feature Matching Loss, Self-Attention, Pixel Normalization, та автоматизованому підбору гіперпараметрів із застосуванням бібліотеки Scikit-Optimize.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення та важливість проблеми

Генерація зображень за допомогою нейронних мереж, зокрема методів типу «Text-To-Image» (текст у зображення, TTI), є однією з найбільш перспективних і активно досліджуваних областей у сфері штучного інтелекту та комп’ютерного зору. Ця технологія дозволяє створювати нові, високоякісні зображення на основі текстових описів, що має велике значення для широкого спектра застосувань, від мистецтва і розваг до медицини та наукових досліджень.

Застосування генерації зображень за допомогою нейронних мереж охоплює численні галузі. Наприклад, в індустрії розваг і медіа, ці технології можуть використовуватись для створення візуальних ефектів, анімацій та навіть нових персонажів для фільмів і відеоігор. У сфері дизайну та моди генеративні моделі можуть допомогти в створенні нових концепцій одягу та інтер’єрів, що відповідають певним стилістичним запитам. В медичній візуалізації ці технології можуть сприяти створенню синтетичних зображень для тренування моделей діагностики, що дозволяє покращити точність і надійність систем штучного інтелекту в медицині.

Однак, генерація зображень є складною задачею, яка вимагає значних обчислювальних ресурсів і ретельного підходу до навчання моделей. Основні виклики включають досягнення високої реалістичності та деталізації створених зображень, що передбачає коректне відтворення текстур, форм, кольорів та інших візуальних характеристик. Додатково, важливою задачею є забезпечення відповідності зображень текстовим описам, що включає адекватне відображення всіх суттєвих деталей, зазначених у тексті.

Важливість проблеми генерації зображень за допомогою нейронних мереж полягає у великому потенціалі цієї технології для різних галузей. Вона

не тільки відкриває нові можливості для творчості та інновацій, але й сприяє розвитку нових методів навчання моделей, які можуть бути застосовані у багатьох інших задачах комп'ютерного зору та штучного інтелекту. Подальші дослідження та вдосконалення моделей генерації зображень сприятимуть створенню ще більш реалістичних та відповідних запитам користувачів зображень, що значно розширить можливості їх застосування у практичних сценаріях.

1.2 Огляд генеративних нейронних мереж

Одними з найпопулярніших генеративних моделей, які використовуються для створення високоякісних зображень, є генеративно-змагальні мережі (Generative Adversarial Network, GAN) та дифузійна модель (Diffusion Model, DM). Ці моделі базуються на різних підходах до генерації зображень, кожна з яких має свої унікальні особливості та переваги.

1.2.1 Generative Adversarial Network

Генеративно-змагальна мережа складається з двох нейронних мереж, що змагаються між собою: генератора та дискримінатора. Генератор створює синтетичні дані, а дискримінатор намагається відрізнити ці дані від справжніх [11]. Це змагання змушує обидві мережі поступово вдосконалювати свої навички у генерації та розпізнаванні даних відповідно (рис. 1.1).

Навчання GAN можна описати за допомогою такого рівняння, де *D* означає дискримінатор, а *G* – генератор:

$$\min_{G} \max_{D} \mathrm{O}(\mathrm{D}, \mathrm{G}) = \mathrm{Ex}\,[\log(\mathrm{D}(\mathrm{x}))] + \mathrm{Ey}\,[\log(1 - \mathrm{D}(\mathrm{G}(\mathrm{y})))], \qquad (1.1)$$

де *O(D, G)* – це цільова функція навчання GAN,

x – зразки зображень,

y – випадкові вектори шуму з певного розподілу.

E_x та E_y позначають розподіли справжніх даних і шуму відповідно.

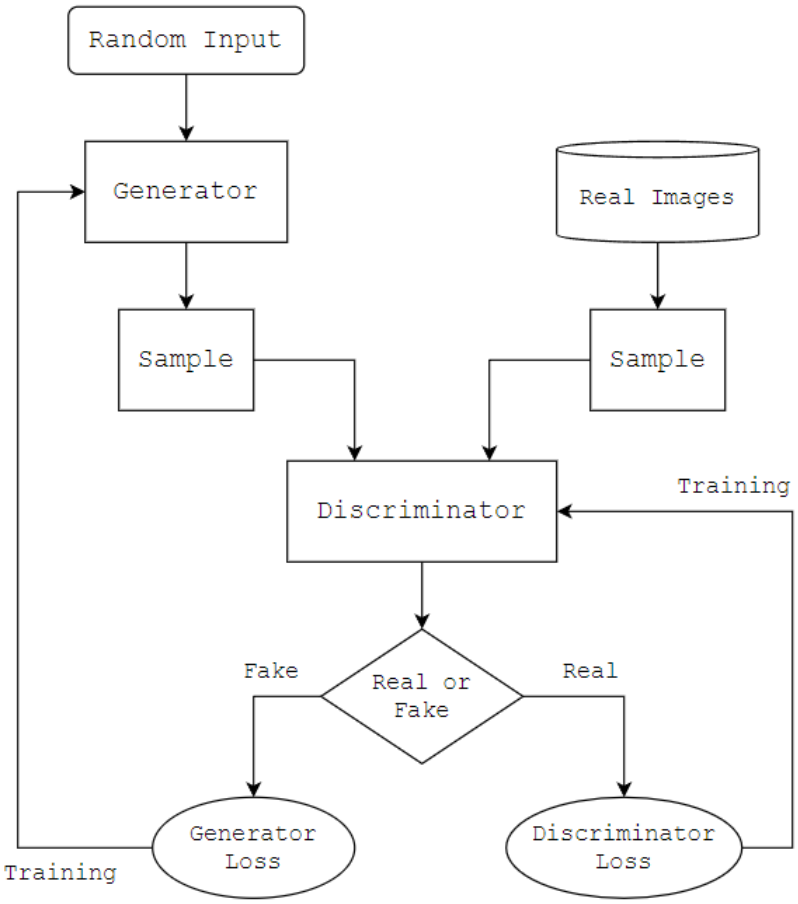


Рисунок 1.1 – Архітектура GAN

Основне завдання GAN – максимізувати функцію $O(D, G)$ для D і мінімізувати її для G. Отже, якщо на вхід D подається зображення x, дискримінатор повинен видати одиницю, тобто $D(x)=1$; якщо ж подається синтетичне зображення, результатом має бути нуль, тобто $D(G(y))=0$. Генератор, навпаки, прагне досягти $D(G(y))=1$.

Це означає, що генератор намагається обманути дискримінатор, створюючи зображення, що виглядають все реалістичніше і які дискримінатор не може відрізнити від справжніх. Такий змагальний процес навчання сприяє вдосконаленню обох мереж, що дозволяє генератору створювати високоякісні

синтетичні дані, які дуже схожі на справжні. GAN досягли значних успіхів у різних сферах, таких як синтез тексту у зображення, генерація зображень людей, синтез фото-портретів, відновлення зображень та усунення дощу на зображеннях, оскільки здатні створювати фотореалістичні зображення.

1.2.2 Diffusion Model

Дифузійні моделі – це генеративні моделі, які створюють високоякісні зображення різної якості за допомогою параметризованих ланцюгів Маркова [1]. Вони працюють шляхом поступового додавання гаусівського шуму до початкових даних під час прямого дифузійного процесу, а потім навчаються видаляти цей шум у зворотному дифузійному процесі. Це латентні змінні моделі, що використовують прихований безперервний простір ознак і базуються на принципах нерівноважної термодинаміки (рис. 1.2) [14].

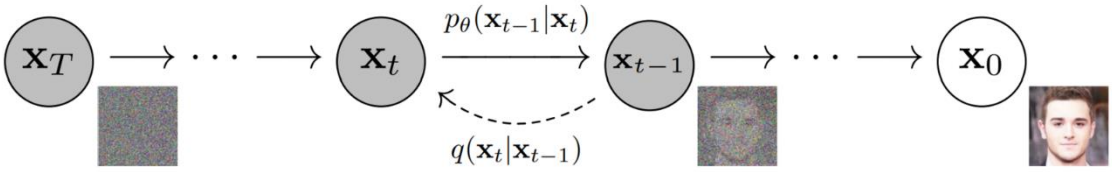


Рисунок 1.2 – Типовий процес Diffusion моделі

Нехай $q(x_0)$ – розподіл реальних даних, зокрема зображень. Обирається зразок з цього розподілу, щоб отримати зображення, $x_0 \sim q(x)$. Оскільки прямий процес є ланцюгом Маркова, визначається перехід між станами $q(x_t|x_{t-1})$, що додає гаусівський шум на кожному часовому кроці t відповідно до певного відомого розподілу дисперсій $\{\beta_t \in (0,1)\}_{t=1}^{T}$, $0 < \beta_1 < \beta_2 < \ldots < \beta_t < 1$.

$$\mathrm{q}(x_t|x_{t-1}) = N\left(x_t;\ \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)\ \mathrm{q}(x_{1:\mathrm{T}}|x_0) = \prod_{t=1}^{T} \mathrm{q}(x_t|x_{t-1}). \quad (1.2)$$

З огляду на Марківську природу, спільний розподіл латентних змінних

є добутком гауссівських умовних перехідних ймовірностей. Важливо зазначити,, що β_t не є постійним на кожному часовому кроці *t*. Порядок дисперсій може бути лінійним, квадратичним, косинусоїдальним і т.д. Таким чином, переходячи через стани $x_{1:T}$, у кінцевому стані x_T можна отримати чистий гаусівський шум.

Якби можна було обернути цей прямий процес і вибирати з переходу $q(x_{t-1}|x_t)$, то можна було б відновити справжню вибірку з початкового гаусівського шуму, $x_0 \sim N(0,I)$. Однак, безпосередньо оцінити перехід складно, оскільки для цього потрібні всі дані, а отже, потрібно навчати модель p_θ для апроксимації цих умовних ймовірностей і реалізації зворотного процесу. В цій апроксимації середнє та дисперсія також залежать від часової позначки *t*. Таким чином, нейронна мережа повинна визначати ці параметри.

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1};\ \mu_0(x_t,t),\ \sum_0(x_t,t)). \tag{1.3}$$

Під час навчання можна використовувати нижню варіаційну межу (Evidence lower bound, ELBO) для мінімізації від'ємної log-правдоподібності стосовно зразка, x_0. ELBO для цього процесу є сумою втрат на кожному часовому кроці

$$L = L_0 + L_1 + \ldots + L_T. \tag{1.4}$$

Використовуючи прямий процес q та зворотний процес p_θ, можна виразити функції втрат, окрім L_0, за допомогою розходження Кульбака-Лейблера (KL). KL розходження – асиметрична міра відстані, що показує, наскільки один розподіл ймовірностей P відрізняється від еталонного розподілу Q. Таке формулювання підходить для процесу, оскільки перехідні розподіли в марковському ланцюгу є нормальними, а KL розходження між нормальними розподілами має замкнену форму:

$$L_0 = -\log p_\theta(x_0|x_1),$$
$$L_t = D_{KL}\,(q(x_t|x_{t+1}, x_0) \,||\, p_\theta(x_t|x_{t+1})),$$
$$L_T = D_{KL}\,(q(x_T|x_0) \,||\, p_\theta(x_T)), \qquad (1.5)$$

де L_T – константа, тому його можна проігнорувати під час навчання моделі, оскільки q не має параметрів для навчання,

x_T – гаусівський шум.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ

2.1 Deep Fusion Generative Adversarial Networks

Метод глибокого синтезу генеративно-змагальної мережі (DF-GAN) спрямований на покращення синтезу високоякісних зображень за текстовими описами. Основна ідея полягає у вирішенні трьох основних проблем попередніх методів:

– складність архітектури;

– обмежена здатність додаткових мереж контролювати семантичну відповідність;

– недостатнье використання текстової інформації [2].

Модель складається з трьох ключових компонентів: генератора, дискримінатора, та попередньо навченого текстового кодувальника (рис. 2.1).

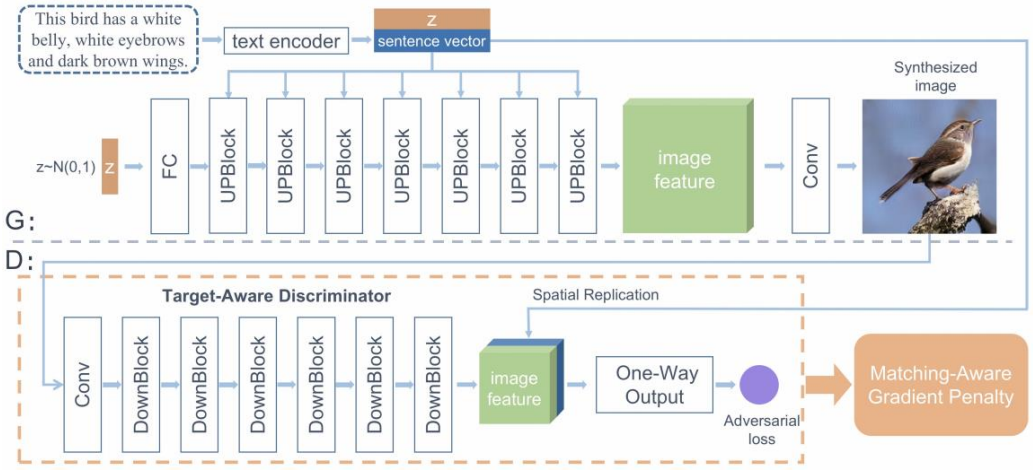


Рисунок 2.1 – Архітектура DF-GAN

Генератор приймає два вхідні вектори: закодований текстовий вектор та вектор шуму, взятий з гауссового розподілу для забезпечення різноманітності створених зображень. Спочатку вектор шуму проходить через повноз’єднаний

шар для перетворення у відповідну форму. Далі йде серія UP-блоків, які збільшують розмір зображення. Ці блоки складаються з шару підвищення дискретизації, залишкового блоку та DF-блоків (Deep text-image Fusion Block), що поєднують текстову та візуальну інформацію під час процесу генерації зображення. Нарешті, згортковий шар перетворює ознаки у фінальне зображення.

Дискримінатор перетворює зображення на ознаки за допомогою серії Down-блоків. Потім текстовий вектор дублюється та об’єднується з ознаками зображення. Обчислюється змагальна втрата для оцінки візуальної реалістичності та семантичної відповідності вхідних даних. Розрізняючи згенеровані та реальні зображення, дискримінатор спонукає генератор створювати більш якісні та текстово-відповідні зображення.

Текстовий кодувальник – це двонаправлена модель довготривалої короткочасної пам’яті (Long Short-Term Memory, LSTM), яка витягує семантичні вектори з текстового опису. Він базується на попередньо натренованій моделі від методу AttnGAN (Attentional Generative Adversarial Network).

Через нестабільність моделей GAN, деякі TTI версії GAN зазвичай використовували багатоступеневу архітектуру для генерації високоякісних зображень з низькоякісних. Проте це призводило до накладок між генераторами, що робило зображення нечіткими.

DF-GAN пропонує одноступеневу архітектуру, яка дозволяє безпосередньо синтезувати високоякісні зображення за допомогою однієї пари генератора та дискримінатора. Використовується шарнірна функція втрат для стабілізації процесу навчання. Єдиний генератор має більше шарів, щоб синтезувати високоякісні зображення з шуму. Для ефективного навчання цих шарів вводяться залишкові мережі, які дозволяють стабілізувати навчання глибоких мереж.

Для покращення семантичної відповідності між текстом і зображенням,

застосовується Target-Aware Discriminator, що складається з Matching-Aware Gradient Penalty (MA-GP) та One-Way Output. Дискримінатор оцінює чотири види входів:

– синтетичні зображення з відповідним текстом;
– синтетичні зображення з невідповідним текстом;
– реальні зображення з відповідним текстом;
– реальні зображення з невідповідним текстом.

MA-GP накладає градієнтне штрафування саме на реальні зображення з відповідним текстом, що сприяє кращій конвергенції моделі та створенню текстово-відповідних зображень. One-Way Output спрощує структуру дискримінатора, видаючи лише одне значення для кожного зображення незалежно від тексту. Це забезпечує кращу збіжність і стабільність навчання, зосереджуючи дискримінатор на покращенні якості зображень.

Для кращої інтеграції текстової інформації в ознаки зображення використовується DF-Block. Він складається з декількох афінних перетворень, що маніпулюють візуальними ознаками через операції масштабування та зсуву каналів, що дозволяє моделям краще захоплювати текстові ознаки на різних рівнях абстракції. Використання кількох DF-Blocks на різних масштабах зображення поглиблює процес інтеграції тексту та зображення, забезпечуючи повне злиття текстових і візуальних ознак.

2.2 Prompt Auto-Editing

Метод Prompt Auto-Editing (PAE) було розроблено для автоматичного редагування текстових запитів з метою покращення генерації зображень текстом. Зазвичай користувачі додають модифікатори та коригують ваги або часові проміжки для окремих слів у запитах. Це є трудомістким процесом, що вимагає багато часу та зусиль [4]. PAE автоматизує цей процес за допомогою підходу, який включає в себе два етапи: контрольоване налаштування та онлайн навчання з підкріпленням (рис. 2.2). Останні дослідження

підтверджують значимість автоматизації редагування запитів, підкреслюючи, що такий підхід дозволяє суттєво покращити якість генерації зображень та підвищити ефективність TTI моделей [16].

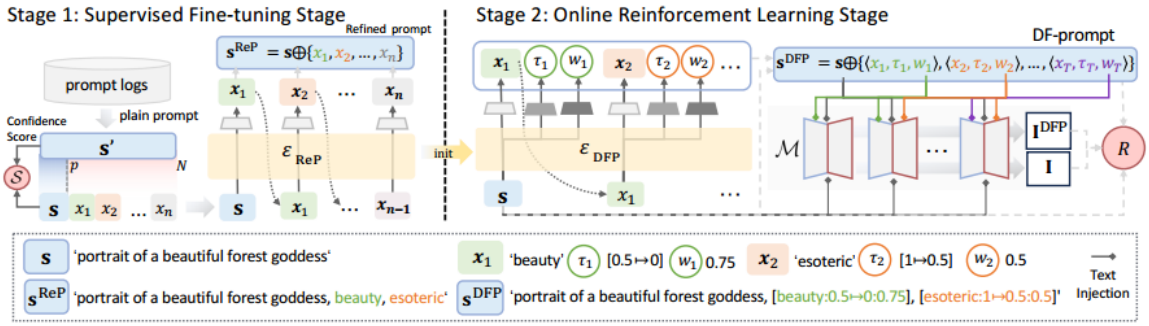


Рисунок 2.2 – Процес навчання PAE

Під час контрольованого налаштування метод навчається на існуючих парах запит-зображення, що дозволяє моделі краще розуміти та інтерпретувати початкові запити користувачів. Це досягається завдяки додаванню модифікаторів до запитів під час навчання. Модифікатори враховують додаткові фактори, такі як стиль, кольори та інші візуальні елементи, що покращують якість згенерованих зображень.

Онлайн навчання з підкріпленням дозволяє моделі динамічно оновлювати свої параметри на основі якості згенерованих зображень. Це досягається завдяки використанню методу зворотного поширення помилки та технік підкріплення, що дозволяє моделі адаптуватися до нових даних у реальному часі. Модель отримує винагороду за кожне згенероване зображення на основі його якості, яка використовується для оновлення параметрів, покращуючи якість майбутніх зображень. Такий підхід дозволяє моделі генерувати більш точні та якісні зображення, що відповідають початковим запитам користувачів.

Метод PAE впроваджує новий формат запитів під назвою динамічний запит для точного контролю (Dynamic Fine-control Prompt, DF-Prompt), який складається з кількох трійок: токен, діапазон впливу, та важливість. Цей

формат забезпечує більш точний контроль над процесом генерації зображень.

Застосовуючи попередньо натреновану генеративну модель тексту в зображення M і вхідний запит користувача s, метою є створення модифікованого запиту s_m з точним керуванням, щоб згенероване зображення $I_m \sim M(s_m)$ мало покращені візуальні ефекти, залишаючись вірним семантиці початкової запиту. Модифікований запит містить початковий запит і набір передбачених модифікаторів A:

$$A=\{x_1, \cdots, x_i, \cdots, x_n\},$$
$$s_m=s \oplus A, \tag{2.1}$$

де символ $\oplus$ означає операцію додавання.

Новий DF-Prompt формат запиту збагачує інформацію початкового запиту. У цьому форматі кожен токен x_i з набору модифікаторів, поєднаний з діапазоном ефекту τ_i та певною вагою w_i, утворюючи трійку $ai=\langle x_i, \tau_i, w_i \rangle$. Вага є числом з плаваючою комою, що визначає вплив токена під час генерації зображення. Діапазон $\tau_i = [b_i \rightarrow e_i]$ $(1 \geq b_i \geq e_i \geq 0)$ є нормалізованим діапазоном, що визначає початковий і кінцевий етапи під час ітеративного процесу знешумлення текстово-зображувальної моделі.

DF-Prompt та його набір токенів визначаються як:

$$A_{DFP}=\{\langle x_1, \tau_1, w_1 \rangle, \cdots, \langle x_n, \tau_n, w_n \rangle\},$$
$$s_{DFP}=s \oplus A_{DFP}. \tag{2.2}$$

Головна мета DF-Prompt полягає у забезпеченні точного і контрольованого генерування, створюючи оптимальну структуру підказок для обробки моделлю M. Для спрощення демонстрації та реалізації коду трійки записуються у форматі звичайного тексту в квадратних дужках: [токен:діапазон:вага].

2.3 DiffMorph

Метод спрямований на створення зміненого 2D-зображення на основі базового зображення з послідовним додаванням нарисів для умовної генерації. Це дозволяє досягти більшого художнього контролю через взаємодію з нарисами під час процесу генерації [3].

ControlNet є новаторським підходом до генерації зображень за допомогою дифузійної техніки. Він створює нові зображення, використовуючи зовнішні умови, такі як контури, глибина, α-пози або ескізи. Для цього умови обробляються через кодерний блок, а вихід з кожного шару поєднується з відповідним шаром знешумлювальної моделі U-Net (рис. 2.3).

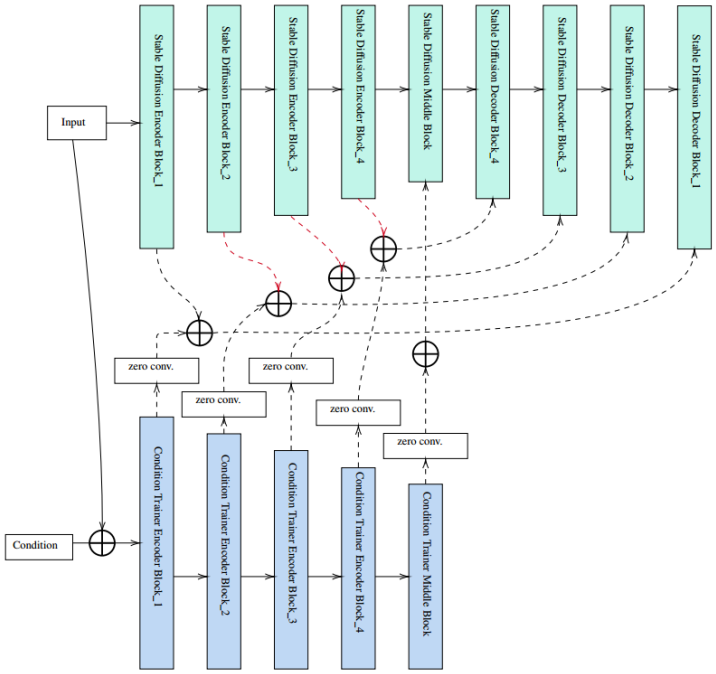


Рисунок 2.3 – Архітектура ControlNet

Модель застосовує блоки Stable Diffusion, які містять кілька шарів кодера та декодера, для поступового вдосконалення зображення. Кодерний блок Stable Diffusion обробляє вхідні умови, які потім проходять через Середній блок Stable Diffusion для подальшого вдосконалення, а потім

декодуються Декодерним блоком Stable Diffusion. Така ієрархічна обробка дозволяє ефективно інтегрувати умови на основі ескізів у процес генерації зображень.

ControlNet підвищує точність згенерованих зображень, додаючи додаткові контрольні умови до дифузійної моделі. Це дозволяє точно налаштовувати згенероване зображення, щоб воно відповідало наданому ескізу. Інтеграція умов на різних рівнях дифузійної моделі забезпечує детальне та точне перетворення ескізу у високоякісне зображення.

Традиційні методи налаштування зазвичай зосереджуються на вдосконаленні дифузійної моделі для генерації конкретних об’єктів, пов’язаних з певними класами або концепціями, зазвичай на основі одного або декількох зображень. Ці методи часто страждають від перенавчання, що зменшує вплив нових умов на згенерований результат.

Пропонований DiffMorph метод вирішує цю проблему, потребуючи лише одне зображення на концепцію та працюючи з кількома классами. Після налаштування вдосконалена модель може генерувати зображення, що включають всі класи.

Спочатку ідентифікуються класи наданих зображень і ескізів за допомогою класифікації CLIP, після чого оптимізується зв’язок між цими класами за допомогою ConceptNet. Спільний зв’язок між класами обирається для умов генерації зображення (рис. 2.4).

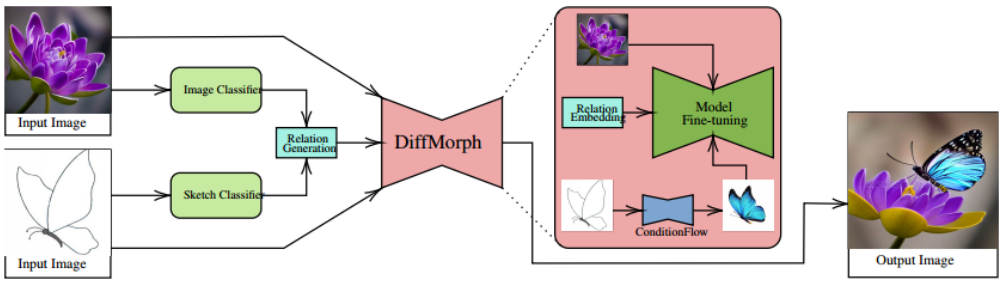


Рисунок 2.4 – Архітектура DiffMorph

Згенероване зображення з ескізу створюється за допомогою моделі

ConditionFlow, а рівень налаштування коригується для забезпечення кастомізації без перенавчання. Під час процесу налаштування мінімізується втрата реконструкції на кожній ітерації. Функція втрат для налаштування двох зображень визначається наступним чином:

$$L(x, s, c_x, c_s, \theta) = E_{t,t',\epsilon,\epsilon'}\left[w_t * \left|\left|\epsilon_\theta(x_t, t, c_x) - \epsilon\right|\right|_2^2 + w_{t'} * \lambda\left|\left|\epsilon_\theta(x_{t'}, t', c_s) - \epsilon'\right|\right|_2^2\right], \quad (2.3)$$

де x – надане зображення,

s – ескіз,

x_t – зображення, згенероване з ескізу,

cx і cs – класи зображення та ескізу відповідно.

Значення λ і w_t, $w_{t'}$ визначаються на основі площ, які займають зображення та ескіз, а також експериментальних гіперпараметрів. Ці параметри використовуються в мережі ϵ_θ для прогнозування шумів x_t і $x_{t'}$.

Цей підхід точно налаштовує вагу модуля генерації зображень у моделі Stable Diffusion та мінімізує функцію втрат для реконструкції вхідних зображень. Цей процес є швидшим за традиційний метод, займаючи лише 1-1,5 хвилини на кожну концепцію. Крім того, гіперпараметри, отримані за допомогою розробленого підходу до покриття площі, обмежують тривалість налаштування моделі, що знижує ризик перенавчання. Це дозволяє генерувати модифіковані зображення, які охоплюють кілька класів.

3 РОЗРОБКА АРХІТЕКТУРИ

У цьому розділі представлено детальний огляд архітектури Deep Convolutional GAN, а також аналізуються додаткові модулі оптимізації, серед яких Minibatch Standard Deviation, Pixel Normalization, Feature Matching Loss та Self-Attention Module. Використання цих підходів дозволяє вирішувати проблеми стабільності, деталізації та відповідності результатів очікуванням.

3.1 Deep Convolutional Generative Adversarial Network

Архітектура глибокої конволюційної генеративно-змагальної мережі (Deep Convolutional Generative Adversarial Network, DCGAN) є вдосконаленням традиційних GAN, що використовує конволюційні нейронні мережі для покращення якості згенерованих зображень. DCGAN складається з двох основних компонентів: генератора та дискримінатора, які навчаються одночасно в процесі змагання між собою (рис. 3.1) [5,9].

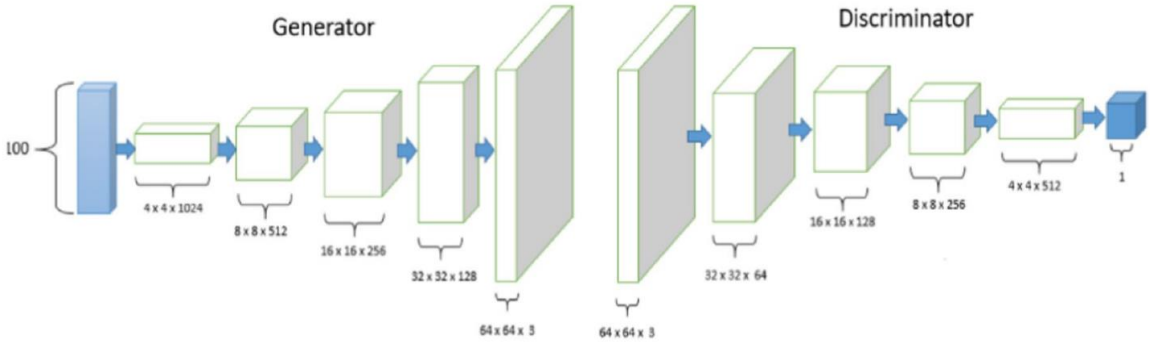


Рисунок 3.1 – Типова архітектура DCGAN

Протягом процесу генератор приймає на вхід випадковий шумовий вектор і поступово перетворює його на реалістичне зображення через послідовність шарів, що включають транспоновані конволюційні операції (ConvTranspose), пакетну нормалізацію (Batch Normalization) та функції

активації ReLU. Останній шар генератора використовує функцію активації Tanh для отримання вихідного зображення з пікселями в діапазоні [-1, 1]. Використання транспонованих конволюцій дозволяє поступово збільшувати роздільну здатність зображень, додаючи їм реалістичні текстури та деталі.

Дискримінатор у свою чергу отримує на вхід зображення і класифікує його як реальне або згенероване. Він складається з послідовності конволюційних шарів із застосуванням страйдінгу (strided convolutions) для зменшення розмірності, пакетної нормалізації та функції активації. Цей підхід допомагає дискримінатору швидше навчатися та запобігає проблемі «вибуху градієнтів». На вихідному шарі використовується сигмоїдна функція активації для отримання ймовірності, яка відповідає класу зображення (справжнє чи підробка).

Під час тренування генератор і дискримінатор змагаються: генератор намагається створити зображення, які неможливо відрізнити від реальних, тоді як дискримінатор прагне виявити підроблені дані. Цей процес приводить до поліпшення обох компонентів упродовж ітерацій. Для обчислення втрат генератора та дискримінатора використовується функція крос-ентропії, яка допомагає мінімізувати різницю між реальними та синтетичними зображеннями.

Однією з ключових переваг DCGAN є його здатність ефективно використовувати просторову інформацію, завдяки чому забезпечується висока якість і деталізація згенерованих зображень. Застосування пакетної нормалізації на кожному шарі генератора й дискримінатора сприяє стабільності навчання, а використання транспонованих конволюцій дозволяє уникнути артефактів, що виникають при простому масштабуванні зображень.

3.2 Minibatch Standard Deviation

Одним із методів підвищення варіативності синтезованих зображень є мініпакетна стандартна девіація (Minibatch Standard Deviation, або Minibatch

Discrimination). Цей підхід базується на аналізі статистичних характеристик даних у межах кожного мініпакету, що подається до дискримінатора [6].

Мініпакетна девіація дозволяє дискримінатору враховувати середню варіацію між зображеннями в пакеті, додаючи до кожного вихідного шару спеціальний «узагальнюючий» вектор. Для цього спершу обчислюється стандартне відхилення піксельних значень по всьому пакету, після чого отриманий показник додається як додаткова ознака [8]. Формула розрахунку девіації має наступний вигляд:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}, \tag{3.1}$$

де m – кількість елементів у мініпакеті,

x_i – значення окремого елемента,

μ – середнє значення ознаки у мініпакеті [13].

Розширення мініпакетної девіації передбачає інтеграцію отриманих статистичних значень у внутрішні шари дискримінатора. Це забезпечує покращення спроможності мережі виявляти повторювані чи недостатньо різноманітні структури, що є важливим для підтримки генерації нових унікальних зразків.

3.3 Pixelwise Feature Vector Normalization

Іншим методом оптимізації є нормалізація піксельних ознак (Pixelwise Feature Vector Normalization, або Pixel Normalization), що використовується в генераторі після кожного шару згортування. Ця техніка допомагає уникнути ескалації сигналів у процесі тренування, що є поширеною проблемою для генеративних мереж [8].

Піксельна нормалізація працює працює таким чином, що після обчислення значень ознак для кожного пікселя їхній вектор нормалізується до

одиничної довжини. Завдяки цьому зменшується вплив абсолютного масштабу сигналів і забезпечується збалансована передача інформації між шарами [12]. Формула нормалізації вектора:

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{n}\sum_{j=1}^{n}(a_{x,y}^{j})^2+\epsilon}}, \quad (3.2)$$

де $a_{x,y}$ – оригінальне значення вектора ознак пікселів x і y,

n – кількість ознак,

$\epsilon = 10^{-8}$ – мале додаткове значення для запобігання діленню на нуль.

Додатковою перевагою цього підходу є те, що він забезпечує стабільність у генерації навіть при нарощуванні шарів, коли роздільна здатність мережі досягає максимуму. Це дозволяє уникати перенасичення деталей і підтримувати реалістичність зображень на фінальних етапах тренування.

3.4 Feature Matching Loss

Втрата відповідності ознак (Feature Matching Loss, FML) є одним з підходів для стабілізації процесу навчання GAN і покращення якості згенерованих зображень. Його основна ідея полягає у порівнянні проміжних векторів ознак (feature maps), отриманих у прихованих шарах дискримінатора, для реальних та згенерованих зображень. На відміну від традиційної функції втрат GAN, яка спрямована на максимізацію ймовірності обману дискримінатора, FML допомагає генератору навчитися відтворювати статистичні властивості реальних зображень у просторі ознак [6,10].

FML визначається як середньоквадратичне відхилення між середніми представленнями реальних і синтезованих зображень у прихованому просторі ознак дискримінатора. Це обчислюється за формулою:

$$L_{FM} = ||E_{x\sim P_{data}}[f_D(x)] - E_{z\sim P_z}[f_D(G(z))]||_2^2, \quad (3.3)$$

де f_D – проміжний вектор ознак на прихованому шарі дискримінатора,

x – реальні дані,

G – згенеровані дані,

z – випадковий шум,

E – математичне очікування.

Особливістю використання цієї функції втрат є зниження ймовірності виникнення колапсу режиму (mode collapse), коли генератор створює обмежений набір варіативних зображень. Завдяки аналізу векторів ознак на проміжному рівні, дискримінатор забезпечує генератор більш детальною зворотною інформацією, яка стимулює створення нових, якісно різноманітних даних.

Практичне застосування FML охоплює задачі, де потрібне підвищення реалістичності синтезованих зображень, зокрема у генерації людських облич, текстур для віртуальних середовищ або створенні деталей для ігрових проєктів. Важливим є те, що ця функція втрат дозволяє досягати значних результатів навіть у складних задачах із обмеженими даними, забезпечуючи стабільність навчання генератора.

Використання FML також є актуальним у випадках, коли дискримінатор суттєво переважає генератор за точністю. Завдяки цьому методу генератору вдається ефективніше адаптуватися до складної оцінки дискримінатора, що у свою чергу зменшує ризик дестабілізації навчання.

3.5 Self-Attention Module

Модуль самоуваги (Self-Attention Module) є одним із ключових удосконалень у GAN, яке дозволяє краще розуміти довгострокові залежності

між різними частинами зображення. Це забезпечує можливість враховувати глобальний контекст під час генерації, що особливо важливо для створення зображень із високою деталізацією та узгодженістю. Підхід вперше був запропонований у контексті генеративно-змагальних мереж самоуваги (Self-Attention Generative Adversarial Network, SAGAN) [7] і згодом став основою для інших архітектур. Основна ідея методу полягає у застосуванні механізму уваги, який дозволяє кожному пікселю «враховувати» інформацію від усіх інших пікселів, що забезпечує глибше розуміння просторових залежностей.

Процес роботи модуля самоуваги включає три основні кроки: обчислення ключів (keys), запитів (queries), та значень (values), а також їхній подальший агрегований аналіз. Кожен піксель, представлений у вигляді вектора ознак, взаємодіє з іншими пікселями через вагові коефіцієнти, що визначаються матрицею уваги (рис. 3.2). Формула для обчислення уваги β між пікселями i та j має вигляд:

$$\beta_{j,i} = \frac{\exp(s_{i,j})}{\sum_{i=1}^{n} \exp(s_{i,j})},$$
$$s_{i,j} = f(x_i)^T g(x_j), \tag{3.4}$$

де $s_{i,j}$ – скалярний добуток між key і querie представленнями пікселів, які отримуються через вагові матриці f і g,

n – загальна кількість пікселів у зображенні.

Після цього обчислюється нове представлення для кожного положення j, що є взваженою сумою всіх values, масштабованих вагами уваги:

$$o_j = v\left(\sum_{i=1}^{n} \beta_{j,i}\, h(x_i)\right),$$
$$h(x_i) = W_h x_i,$$
$$v(x_i) = W_v x_i, \tag{3.5}$$

де W_h, W_v – вагова матриця, яка перетворює значення x_i.

Цей процес дозволяє модулю уваги вибірково фокусуватися на ключових частинах зображення, таких як краї, симетрії чи текстурні особливості, ігноруючи несуттєві або зашумлені області.

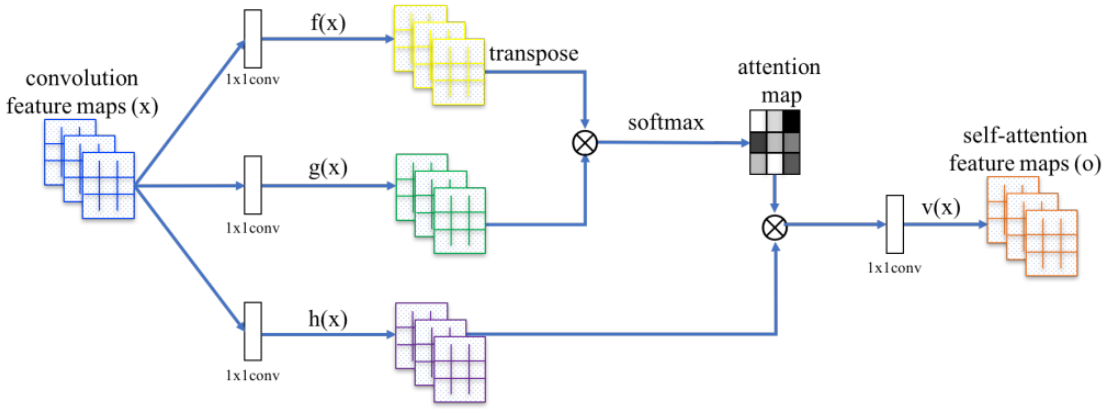


Рисунок 3.2 – Архітектура модулю Self-Attention

Практичне значення модулю полягає у його здатності покращувати якість генерації, особливо у задачах, які потребують складного просторового аналізу. Наприклад, у випадку генерації людських облич, модуль уваги дозволяє ефективніше враховувати симетрію рис обличчя, а у випадку природних пейзажів – забезпечувати когерентність між далекими частинами зображення, такими як небо і горизонт.

Його впровадження також сприяє підвищенню варіативності згенерованих зображень. Завдяки здатності враховувати глобальні залежності, генератор може краще моделювати складні розподіли даних, що зменшує ймовірність mode collapse. Крім того, ця техніка є особливо корисною у великих мережах, де стандартні згорткові операції не здатні ефективно охопити всі взаємозв’язки через обмежений розмір рецептивного поля.

3.6 Frechet Inception Distance

Початкова відстань Фреше (Fréchet Inception Distance, FID) – це метрика,

яка використовується для оцінки якості зображень, створених генеративними моделями, такими як GAN. Вона дозволяє оцінювати схожість між реальними та згенерованими зображеннями шляхом порівняння їх розподілів у просторі ознак. Ці ознаки витягуються за допомогою попередньо навченої нейронної мережі Inception v3 [15], яка була створена для задач класифікації зображень.

На відміну від простих підходів, таких як оцінка якості лише в піксельному просторі, FID використовує більш високорівневі характеристики зображень. Це дозволяє враховувати не тільки візуальну схожість, але й статистичну подібність між наборами реальних і згенерованих зображень. Низьке значення FID свідчить про те, що згенеровані зображення мають розподіл ознак, близький до реальних, і, отже, є більш якісними.

Нижче на рисунку 3.3 показано, як спотворення зображення впливає на значення FID. Можна побачити, що чим більші спотворення, тим вищий результат.

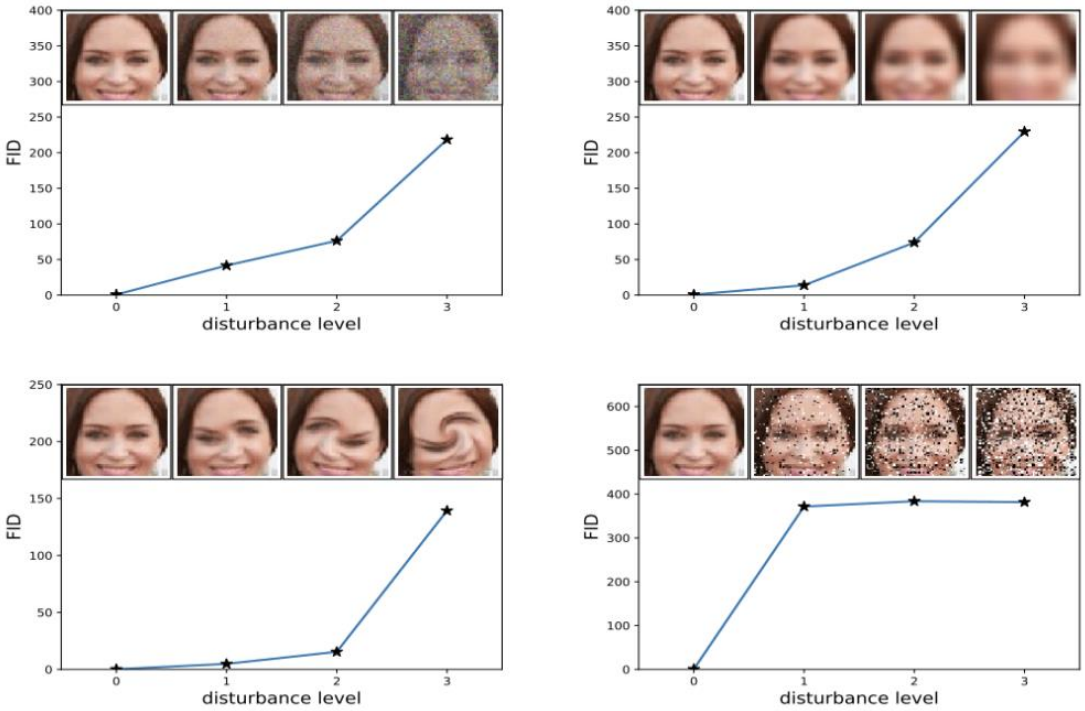


Рисунок 3.3 – Залежність значення FID від рівня спотворення

FID обчислюється на основі статистичних характеристик, отриманих із

двох наборів зображень: реальних та згенерованих. Спочатку обидва набори пропускаються через модель Inception v3, яка генерує векторні представлення (активації) для кожного зображення на одному з внутрішніх шарів. Ці активації описують високорівневі ознаки зображень, такі як форма, текстура чи структура.

Потім обчислюються середнє значення та коваріаційна матриця активацій для кожного набору. FID визначається як математична відстань між двома багатовимірними нормальними розподілами, що задаються цими статистиками. Формула виглядає так:

$$FID = ||\mu_r - \mu_g||^2 + Tr(\sum_r + \sum_g - 2\sqrt{(\sum_r \sum_g)}), \quad (3.6)$$

де Tr – слід матриці,

μ_r, μ_g – середнє значення для реальних та згенерованих зображень,

$\sum_r$, $\sum_g$ – коваріація активацій для реальних та згенерованих зображень.

Результат відображає ступінь схожості між наборами. Якщо значення FID дорівнює 0, це означає, що розподіли повністю ідентичні.

На практиці FID широко застосовується для порівняння ефективності різних архітектур генеративних моделей. Наприклад, результати можуть бути залежними від параметрів мережі, таких як розмір мініпакету, тип функції втрат або методи регуляризації. Незважаючи на це, метод залишається чутливим до артефактів у згенерованих зображеннях і забезпечує більш надійні результати порівняно з іншими метриками.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ І ЕКСПЕРЕМЕНТИ

4.1 Підготовка даних та програмного забезпечення

У рамках дослідження для реалізації та експериментів було використано мову програмування Python завдяки її популярності, наявності великої кількості бібліотек для машинного навчання, а також зручності в реалізації моделей глибинного навчання. Основною бібліотекою для створення та навчання генеративної нейронної мережі стала PyTorch, що забезпечує високу гнучкість при створенні архітектур нейронних мереж, включно з GAN.

Для навчання генеративної мережі було використано датасет MNIST. Цей набір даних містить рукописні цифри розміром 28×28 пікселів у відтінках сірого, що широко застосовується для навчання та тестування моделей машинного навчання (рис. 4.1). Із загального набору зображень було створено підмножину (subset) даних, що дозволяє зменшити час обчислень та полегшує експерименти. Для обробки даних використовувалися стандартні перетворення, такі як нормалізація значень пікселів до інтервалу [-1, 1], що забезпечує стабільність навчання моделей глибинного навчання.

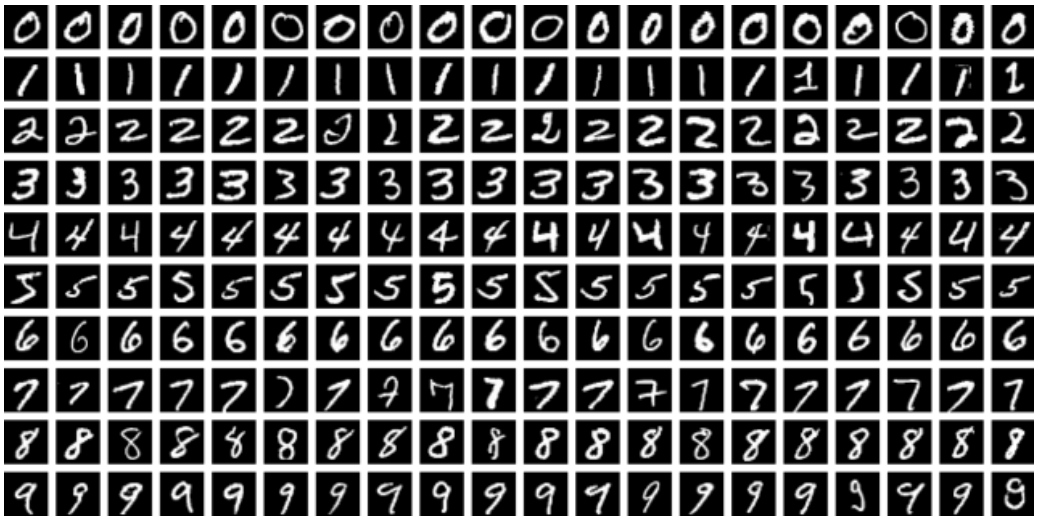


Рисунок 4.1 – Датасет MNIST

Експерименти виконувалися на ноутбуці моделі AN515-47 з відеокартою GeForce RTX 3050Ti на 4 ГБ пам’яті, під управлінням операційної системи Windows 11. Для підвищення швидкості навчання моделі було використано графічний процесор через інтеграцію PyTorch з CUDA, коли це було можливо.

Окрім PyTorch, у реалізації використовувалася бібліотека Scikit-Optimize (Skopt). Цей інструмент дозволяє автоматизувати підбір оптимальних значень гіперпараметрів, таких як швидкість навчання, бета-коефіцієнти оптимізаторів та розмір латентного простору. Автоматизація цього процесу сприяє підвищенню ефективності та точності навчання нейронної мережі.

Серед інших компонентів реалізації можна виділити:

– обчислення показника FID для оцінки якості згенерованих зображень;

– логування даних до інструменту візуалізації TensorBoard для моніторингу процесу навчання [19], включаючи втрати, показник FID та згенеровані зображення;

– збереження результатів роботи в локальну папку для подальшого аналізу.

4.2 Особливості реалізації

Реалізація генеративної моделі містить кілька ключових фрагментів коду, що забезпечують ефективність та автоматизацію навчання нейронної мережі.

У генераторі для покращення деталізації зображень реалізовано механізм Self-Attention, який дозволяє враховувати глобальні залежності між пікселями. Основною метою цього підходу є фокусування на ключових ділянках зображення, що сприяє створенню більш якісних та реалістичних результатів.

Функціонал класу складається з кількох етапів:

1) вхідний тензор перетворюється у три окремі представлення – query, key і value – за допомогою згорткових шарів;

2) обчислюється матриця уваги через скалярний добуток запиту і ключа, після чого вона нормалізується за допомогою нормованої експоненційної функції Softmax для отримання ваг кожного пікселя;

3) зважені значення агрегуються, формуючи вихідний тензор, який додається до початкових даних через залишкове з'єднання.

Цей механізм допомагає моделі обирати значущі регіони зображення, що покращує передачу текстур і структурних елементів. Реалізація класу наведена в лістингу 4.1 нижче.

Лістинг 4.1 – Модуль Self-Attention

```
class SelfAttention(nn.Module):
    def __init__(self, in_dim):
        super(SelfAttention, self).__init__()
        self.query_conv = nn.Conv2d(in_dim, in_dim // 8,
kernel_size=1)
        self.key_conv = nn.Conv2d(in_dim, in_dim // 8,
kernel_size=1)
        self.value_conv = nn.Conv2d(in_dim, in_dim,
kernel_size=1)
        self.gamma = nn.Parameter(torch.zeros(1))
    def forward(self, x):
        batch_size, channels, height, width = x.size()
        query = self.query_conv(x).view(batch_size, -1, height *
width).permute(0, 2, 1)
        key = self.key_conv(x).view(batch_size, -1, height *
width)
        attention = torch.bmm(query, key)
        attention = F.softmax(attention, dim=-1)
        value = self.value_conv(x).view(batch_size, -1, height *
width)
        out = torch.bmm(value, attention.permute(0, 2, 1))
        out = out.view(batch_size, channels, height, width)
        out = self.gamma * out + x
        return out
```

Щоб уникнути проблеми колапсу режиму, коли генератор створює лише

обмежений набір однакових зображень, у дискримінаторі реалізовано механізм Minibatch Discrimination. Цей підхід дозволяє враховувати взаємозв’язки між прикладами всередині одного мінібатчу, що сприяє створенню більш різноманітних зображень.

Клас MinibatchDiscrimination працює наступним чином:

1) вхідний тензор проходить через матрицю T, яка визначається як параметр моделі. Ця матриця містить інформацію про відносини між прикладами;

2) згенерована матриця M використовується для обчислення різниць між кожною парою прикладів. Для цього застосовується функція, що обчислює різницю між елементами у просторі ознак;

3) отримані значення агрегуються, формуючи додаткові ознаки для кожного прикладу, які додаються до його початкових ознак.

Процес дозволяє дискримінатору враховувати не лише індивідуальні ознаки кожного зображення, але й інформацію про їхню унікальність у межах мінібатчу. В лістингу 4.2 наведена реалізація цього класу.

Лістинг 4.2 – Minibatch Discrimination

```
class MinibatchDiscrimination(nn.Module):
    def __init__(self, in_features, out_features,
intermediate_features):
        super(MinibatchDiscrimination, self).__init__()
        self.T = nn.Parameter(torch.randn(in_features,
out_features, intermediate_features))
    def forward(self, x):
        M = x @ self.T.view(x.size(1), -1)
        M = M.view(-1, self.T.size(1), self.T.size(2))
        out = torch.exp(-torch.abs(M.unsqueeze(0) -
M.unsqueeze(1)).sum(dim=3))
        out = out.sum(dim=0) - 1
        out = out.mean(dim=0).unsqueeze(0).expand(x.size(0), -1)
        return torch.cat([x, out], dim=1)
```

Для організації процесу навчання нейронної мережі у коді реалізовано функцію train_step, яка відповідає за навчання генератора і дискримінатора на

кожному кроці. У цьому процесі використовуються функція втрат BCEWithLogitsLoss та оптимізатор Adam.

Функція втрат поєднує операцію логістичної активації (sigmoid) з бінарною крос-ентропією, що дозволяє уникнути чисельної нестабільності, а також є зручним для задач класифікації реальних і згенерованих даних у дискримінаторі:

```
criterion = nn.BCEWithLogitsLoss()
```

Оптимізатор використовується для оновлення ваг генератора і дискримінатора. Завдяки використанню градієнтів першого порядку та моментів другого порядку, Adam забезпечує швидку і стабільну оптимізацію параметрів:

```
g_optimizer = optim.Adam(generator.parameters(), lr=config.lr,
betas=config.betas)
d_optimizer = optim.Adam(discriminator.parameters(),
lr=config.lr, betas=config.betas)
```

Функція train_step реалізує навчання у два етапи:

1) оновлення дискримінатора: реальні зображення передаються через дискримінатор, після чого він оцінює згенеровані зображення. На основі вихідних результатів обчислюється втрата дискримінатора як середнє значення помилок для реальних і згенерованих даних. Далі ваги дискримінатора оновлюються, щоб підвищити його здатність правильно класифікувати дані;

2) оновлення генератора: генератор створює нові зображення, які передаються через дискримінатор. Втрата генератора складається з двох компонентів, що згодом поєднуються для оновлення ваг генератора:

– Adversarial Loss, що мотивує генератор створювати зображення, які дискримінатор оцінює як реальні;

– Feature Matching Loss, яка гарантує подібність між латентними

ознаками реальних і згенерованих зображень.

Лістинг 4.3 демонструє реалізацію цього процесу.

Лістинг 4.3 – Функція train_step

```
def train_step(real_images, noise, generator, discriminator,
g_optimizer, d_optimizer):
    batch_size = real_images.size(0)
    real_labels = torch.ones(batch_size,
1).to(real_images.device)
    fake_labels = torch.zeros(batch_size,
1).to(real_images.device)
    d_optimizer.zero_grad()
    real_output = discriminator(real_images)
    fake_images = generator(noise)
    fake_output = discriminator(fake_images.detach())
    real_loss = criterion(real_output, real_labels)
    fake_loss = criterion(fake_output, fake_labels)
    d_loss = (real_loss + fake_loss) / 2
    d_loss.backward()
    d_optimizer.step()
    g_optimizer.zero_grad()
    real_features = discriminator.forward(real_images).detach()
    fake_features = discriminator.forward(fake_images)
    feature_matching_loss = F.mse_loss(fake_features,
real_features)
    fake_output = discriminator(fake_images)
    adversarial_loss = criterion(fake_output, real_labels)
    g_loss = adversarial_loss + feature_matching_loss
    g_loss.backward()
    g_optimizer.step()
    return g_loss, d_loss
```

Для підвищення ефективності навчання генеративної нейронної мережі використано підхід автоматизованого підбору гіперпараметрів за допомогою бібліотеки Scikit-Optimize (Skopt). Він дозволяє систематично шукати найкращі конфігурації, уникаючи рутинного ручного налаштування [17,18].

Першим кроком є визначення простору пошуку гіперпараметрів, у якому буде проводитися оптимізація. У даному випадку це швидкість навчання (lr), бета-коефіцієнти (beta1 і beta2) для оптимізатора Adam, а також розмір латентного простору (z_dim):

```
space = [Real(1e-5, 1e-3, prior="log-uniform", name='lr'),
    Real(0.3, 0.7, name='beta1'),
    Real(0.8, 0.999, name='beta2'),
    Integer(50, 150, name='z_dim')]
```

Наступним етапом є створення функції hyper (лістинг 4.4), яка виконує налаштування гіперпараметрів моделі, запуск навчання та оцінку якості результатів. Вона приймає на вхід значення гіперпараметрів, оновлює об’єкти конфігурації (config) і створює нові екземпляри генератора та дискримінатора.

Лістинг 4.4 – Збір параметрів та ініціалізація компонентів

```
@use_named_args(space)
def hyper(lr, beta1, beta2, z_dim):
    config.lr = lr
    config.betas = (beta1, beta2)
    config.z_dim = int(z_dim)
    generator = Generator(z_dim=config.z_dim).to(device)
    discriminator = Discriminator().to(device)
    g_optimizer = optim.Adam(generator.parameters(),
lr=config.lr, betas=config.betas)
    d_optimizer     =     optim.Adam(discriminator.parameters(),
lr=config.lr, betas=config.betas)
```

Функція виконує навчання на обмеженій кількості епох, що дозволяє оцінити комбінацію параметрів за короткий час, не витрачаючи значні обчислювальні ресурси. Результати навчання оцінюються за допомогою комбінованої метрики combined_metric, яка поєднує середні значення втрат генератора та дискримінатора разом із оцінкою FID, застосовуючи до них ваги 0.25, 0.25, і 0.5 відповідно. Ознайомитися з програмною реалізацію процесу можна у лістингу 4.5.

Лістинг 4.5 – Навчання та оцінка параметрів

```
test_noise = torch.randn(16, config.z_dim).to(device)
average_fid, fid_count = 0.0, 0
```

```
average_g_loss, average_d_loss = 0.0, 0.0
for epoch in range(5):
    epoch_g_loss, epoch_d_loss = 0.0, 0.0
    for batch_idx, (real_images, _) in enumerate(train_loader):
        real_images = real_images.to(device)
        noise = torch.randn(real_images.size(0),
config.z_dim).to(device)
        g_loss, d_loss = train_step(real_images, noise,
generator, discriminator, g_optimizer, d_optimizer)
        epoch_g_loss += g_loss.item()
        epoch_d_loss += d_loss.item()
    average_g_loss += epoch_g_loss / len(train_loader)
    average_d_loss += epoch_d_loss / len(train_loader)
    with torch.no_grad():
        fake_images = generator(test_noise)
        fid_score = calculate_fid(fid, real_images, fake_images)
        average_fid += fid_score
        fid_count += 1
if fid_count > 0:
    total_fid = average_fid / fid_count
else:
    total_fid = float('inf')
total_g_loss = average_g_loss / 5
total_d_loss = average_d_loss / 5
combined_metric = 0.5 * total_fid + 0.25 * total_g_loss + 0.25 *
total_d_loss
return combined_metric
```

Оптимізація виконується за допомогою функції gp_minimize, яка здійснює визначену кількість ітерацій для пошуку найкращих значень гіперпараметрів, оцінюючи кожну конфігурацію за результатами combined_metric:

```
result = gp_minimize(hyper, space, n_calls=10, random_state=42)
config.lr = result.x[0]
config.betas = (result.x[1], result.x[2])
config.z_dim = int(result.x[3])
```

4.3 Опис експериментів

Експерименти проводилися протягом 30 епох з використанням повного та половини набору даних MNIST (60000 та 30000 зразків зображень відповідно). При навчанні, для оптимізатору Adam було застосовано

швидкість навчання 0.0002 та бета-коефіцієнти [0.5, 0.999], а розмір латентного простору дорівнював 100.

Першим кроком була перевірка якості генерації DCGAN лише з використанням Adam та бінарної крос-ентропії. Середнє значення FID для згенерованих зображень протягом навчання дорівнювало 12.86. Графіки на рисунках 4.2 і 4.3, отримані з TensorBoard, демонструють зміну похибки генератора та дискримінатора впродовж процесу.

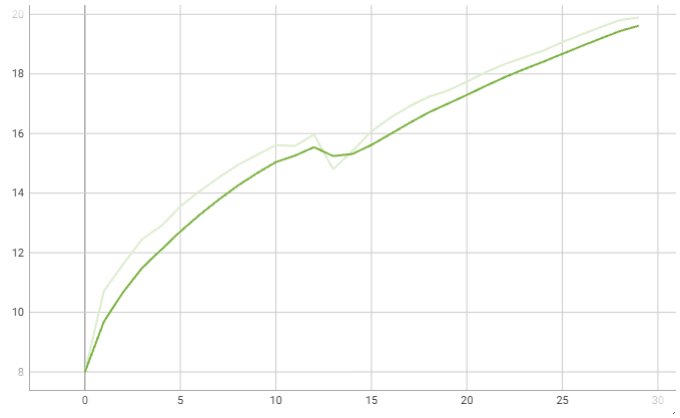


Рисунок 4.2 – Похибка G, перший тест

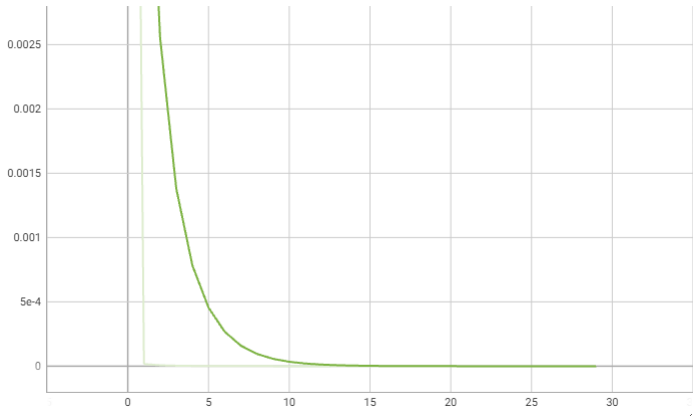


Рисунок 4.3 – Похибка D, перший тест

Як можна побачити, генератор демонструє стабільне зростання похибки впродовж усього навчання. Початкове значення близько 9.85 поступово збільшується, досягаючи максимальних значень близько 19.99 до 30 епохи. Це зростання похибки може вказувати на те, що генератор продовжує

адаптуватися до роботи дискримінатора, водночас покращуючи якість згенерованих зображень.

Початкове значення FID на 2 епосі становило 16.33, що вказує на значну відмінність між реальними та згенерованими даними. Проте до 8 епохи FID суттєво знижується, досягаючи 7.25, що свідчить про помітне покращення реалістичності згенерованих зображень. У цей період можна ідентифікувати окремі чіткі цифри (наприклад, цифру 3). Починаючи з 12 епохи, FID демонструє тимчасове покращення до мінімуму 3.80 на 12 епосі, але згодом коливається, досягаючи максимуму на 14 епосі (20.32). Далі значення FID стабілізуються ближче до кінця навчання, знижуючись до 13.49.

Втрати дискримінатора починаються з 0.0013 на осі ординат і швидко зменшуються до 0.0000 вже після кількох перших епох. Це може свідчити про те, що дискримінатор швидко адаптується до завдання, але також може свідчити про дисбаланс між генератором та дискримінатором у міру ускладнення завдання.

На наступному етапі до архітектури було додано Minibatch Discrimination та Pixel Normalization. Ці зміни були спрямовані на покращення розпізнавання дискримінатором статистичних особливостей зображень та нормалізацію значень на рівні пікселів для стабільнішого навчання. Нижче наведено порівняння тестів (рис. 4.4 і 4.5).

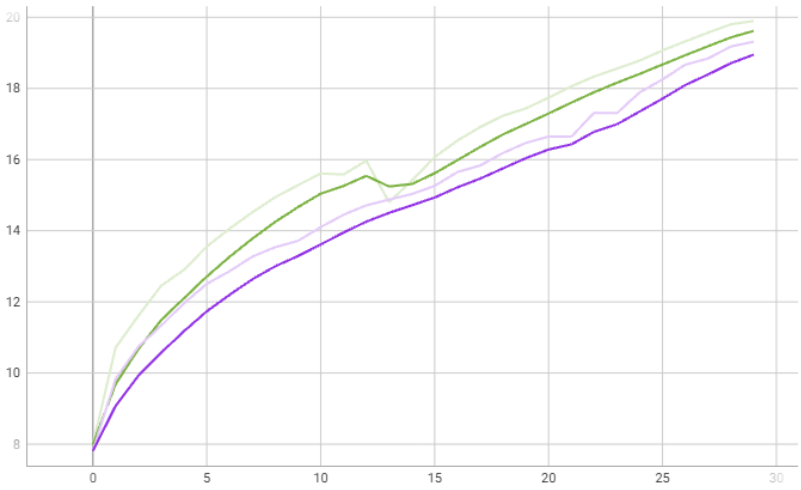


Рисунок 4.4 – Похибка G, 1 та 2 тести

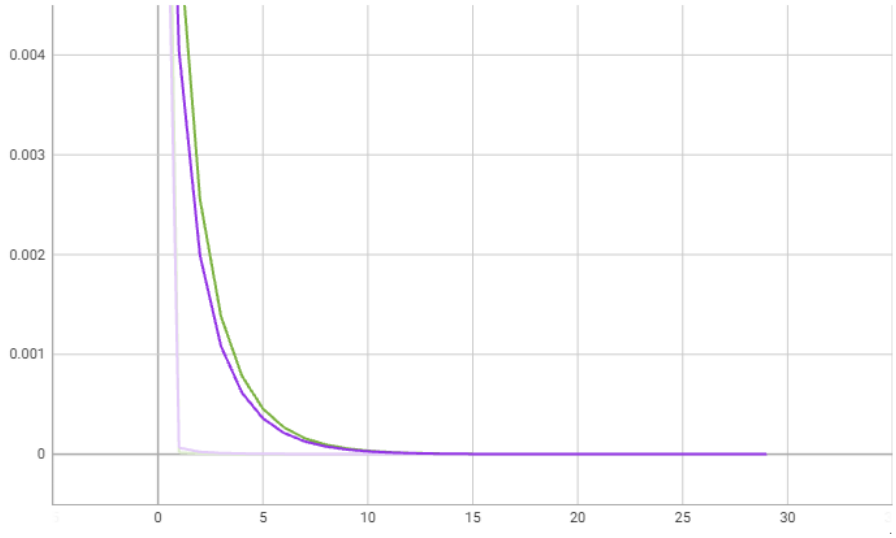


Рисунок 4.5 – Похибка D, 1 та 2 тести

Результати демонструють певні відмінності від першого експерименту, що можна пояснити зміненими умовами навчання. Зокрема, значення втрат генератора на початкових етапах навчання було нижчим, стартуючи з 7.13, тоді як у першому тесті цей показник починався з 9.85. Це свідчить про зменшення стартової невизначеності, що може бути результатом більш вдалої ініціалізації або початкової стабільності генератора. Проте, на пізніших етапах динаміка графіка показує стабільне зростання втрат, яке на 30 епосі досягає 15.22, що все ще нижче пікових значень першого тесту (19.99).

Втрати дискримінатора також є значно нижчими. Початкове значення на першій епосі становить 0.0014, що майже відповідає попередньому. Дискримінатор демонструє кращу адаптацію, знижуючи втрати до 0 вже після 3 епохи. Проте така низька похибка може також вказувати на дисбаланс у системі, де дискримінатор має перевагу, особливо на середніх етапах навчання.

FID-результати підтверджують змішані тенденції. Мінімальне значення FID у другому тесті було зафіксовано на 10-й епосі та становило 9.26, проте подальша динаміка виявилася менш стабільною. Після 10 епохи FID демонструє значний стрибок до 35.31 на 16 епосі, що значно перевищує будь-

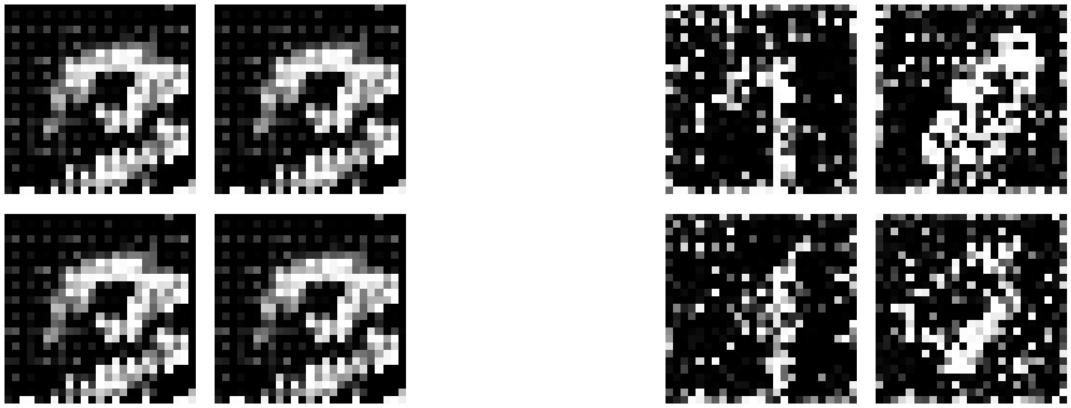
яке значення з першого тесту. Наприкінці навчання FID знижується, але середнє значення залишається високим – 21.91, що значно перевищує 12.86 у першому тесті.

На етапі 8–10 епох можна розібрати цифри, які мають більш різноманітну структуру, ніж у першому тесті, але подальша якість починає погіршуватися. Це може бути пов’язано зі складністю завдання для дискримінатора через зміни в його архітектурі, що негативно впливає на навчання генератора.

На рисунку 4.6 можна побачити приклади якісних результатів перших двох генерацій:

а) результат 8 епохи 1 тесту;

б) результат 10 епохи 2 тесту.



а) б)

Рисунок 4.6 – Приклади якісних результатів перших двох генерацій

Далі до процесу було підключено FML та Self-Attention. Ці модифікації були спрямовані на зменшення розбіжностей між реальними та згенерованими зображеннями за допомогою оптимізації внутрішніх представлень дискримінатора та врахування ширших просторових залежностей у згенерованих даних. Зміну похибок наведено на рисунках 4.7 і 4.8.

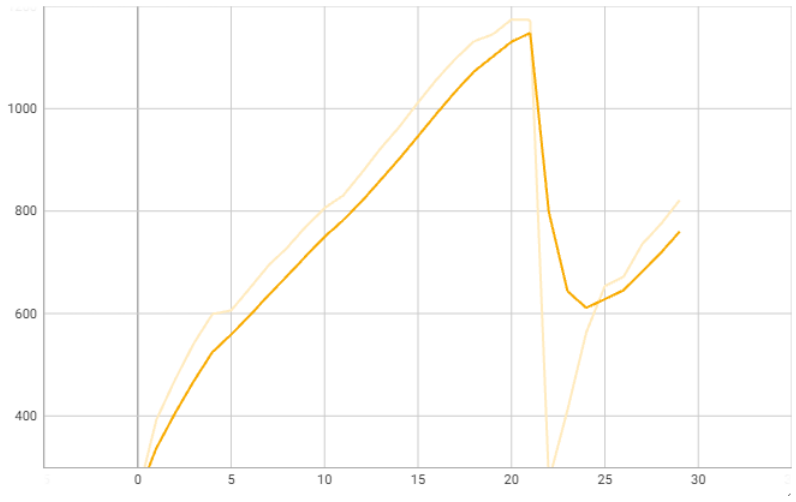


Рисунок 4.7 – Похибка G, третій тест

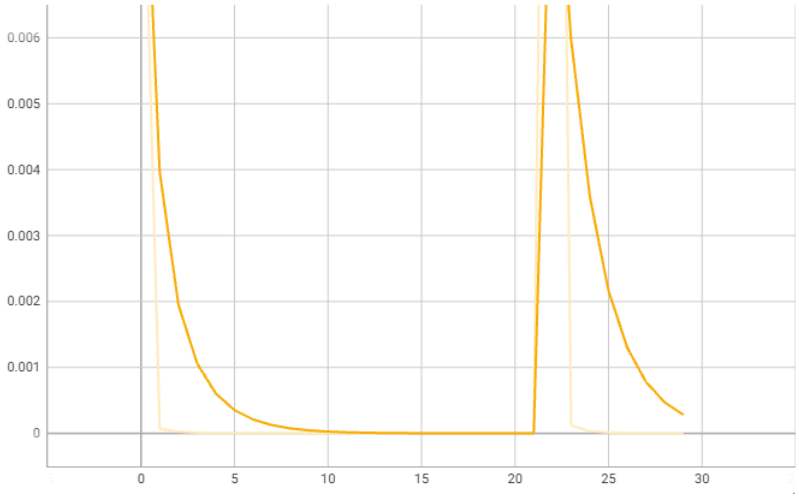


Рисунок 4.8 – Похибка D, третій тест

На графіках помітно, що на початкових етапах навчання похибка генератора демонструє помірне зростання, починаючи з 206.91 та досягаючи локального максимуму на 10 епосі (461.54). Це зростання свідчить про початкові труднощі генератора у створенні реалістичних зображень. Однак після цього похибка стабілізується і поступово знижується до 413.50 на 9 епосі, що збігається зі зменшенням значення FID до 13.83 на цьому ж етапі. Подібне падіння свідчить про те, що генератор починає створювати більш реалістичні зображення.

Проте після 9 епохи спостерігається нерівномірна динаміка. Зростання

похибки генератора досягає нового максимуму (920.06) на 29 епосі. Незважаючи на це, значення FID знову знижується наприкінці навчання (6.13 на 30 епосі), що вказує на покращення якості зображень, навіть за умов збільшення втрат генератора.

Втрати дискримінатора залишаються стабільними протягом усього навчання, коливаючись у діапазоні від 0.01 до 0.0000. Це може бути ознакою того, що дискримінатор швидко адаптувався до завдання, проте така стабільність також може вказувати на надмірну перевагу дискримінатора над генератором.

FID демонструє значну варіативність. Початкові значення (15.63 на 2 епосі) свідчать про низьку якість згенерованих зображень, однак вже до 6 епохи FID досягає мінімуму (6.67), що підтверджує ефективність генератора в цей період. В подальшому FID знову зростає, досягаючи локального максимуму 35.19 на 14 епосі, а потім поступово знижується. Середнє значення FID протягом усього процесу навчання склало 18.22, що перевищує середнє значення у першому тесті (12.86) та є нижчим за другий тест (21.91). Це свідчить про те, що хоча якість згенерованих зображень не досягла рівня першого тесту, результат третього тесту є значним покращенням порівняно з другим.

Таким чином, результати третього тесту вказують на поліпшення роботи генератора в середніх етапах навчання (6–10 епохи), коли спостерігається чітке покращення FID. Водночас нерівномірність втрат генератора на пізніх етапах свідчить про потенційні проблеми у балансуванні генератора і дискримінатора.

Останнім експериментом була оптимізація гіперпараметрів з використанням Skopt. Для цього кожному з параметрів було встановлено певний діапазон значень:

– lr = [0.00001, 0.001];
– beta1 = [0.3, 0.7];
– beta2 = [0.8, 0.999];

– z_dim = [50, 150].

Функцією було зроблено 10 перевірок різноманітних комбінацій протягом 5 епох, після чого оптимальні гіперпараметри застосовувалися до навчання. Нижче (рис. 4.9 і 4.10) можна побачити результати при lr = 0.000063, beta = [0.3187, 0.9938], та z_dim=109.

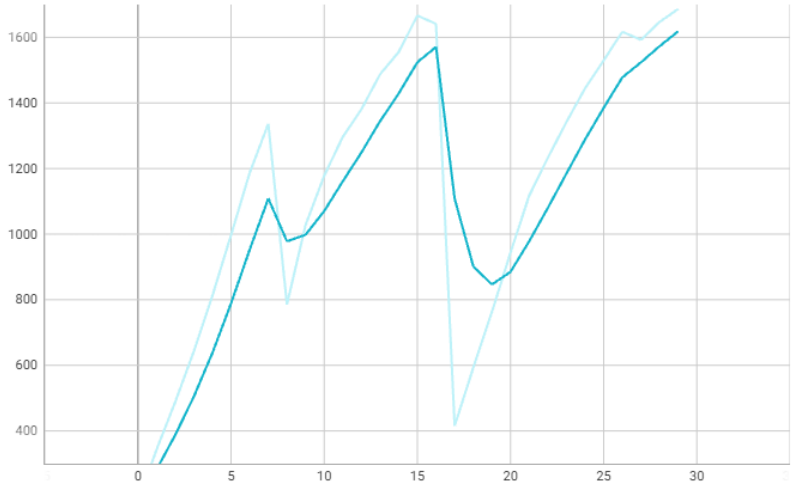


Рисунок 4.9 – Похибка G, 4 тест

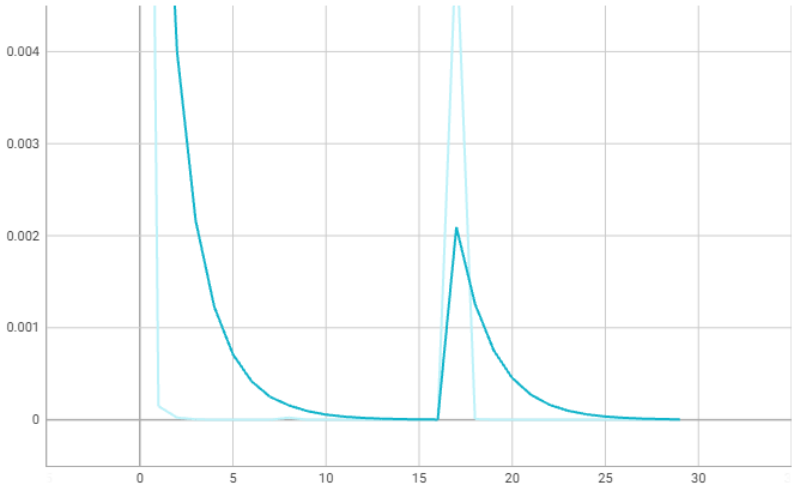


Рисунок 4.10 – Похибка D, 4 тест

На графіку втрат генератора видно, що його похибка демонструє помітні коливання. У перших епохах похибка різко зростає, досягаючи максимуму між 7 і 15 епохами (понад 1000). На цьому етапі модель активно шукає оптимальні шляхи генерації. Після 13 епохи спостерігається значний спад до 19 епохи

(приблизно 266), що вказує на поступове покращення роботи генератора. Проте після цього, аж до завершення навчання, відбувається чергове зростання похибки, що може бути пов’язано з перегином генератора у спробах обійти дискримінатор.

Втрати дискримінатора залишаються стабільно низькими протягом усього навчання. Це свідчить про те, що дискримінатор швидко адаптувався до змін і не зазнає труднощів із розпізнаванням реальних та згенерованих даних.

FID, навпаки, демонструє більш складну динаміку. У перші епохи він знижується з 11.62 на 2 епосі до мінімуму 2.33 на 10 епосі, що підтверджує покращення реалістичності згенерованих зображень. Однак після 10 епохи FID починає зростати, досягаючи локального максимуму (37.13 на 14 епосі). Ця тенденція змінюється після 18 епохи, коли FID знову спадає до значень нижче 10. Середнє значення протягом усього процесу навчання дорівнює 13.27, що практично дорівнює значенню з першого тесту.

Рисунок 4.11 демонструє приклади якісних результатів останніх двох генерацій:

а) результат 6 епохи 3 тесту;

б) результат 18 епохи 4 тесту.

а) б)

Рисунок 4.11 – Приклади якісних результатів останніх двох генерацій

У порівнянні з попередніми тестами, застосування оптимізації гіперпараметрів дозволило досягти високої стабільності втрат дискримінатора та покращення якості згенерованих зображень на ранніх етапах. Проте коливання FID та втрат генератора на пізніх епохах свідчать про необхідність подальшої додаткової уваги до узгодженості між генератором і дискримінатором протягом фінальних етапів навчання.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було проведено дослідження методів оптимізації генерації зображень за допомогою нейронних мереж.

У роботі було розглянуто принципи роботи відомих мереж GAN та DM. Встановлено, що GAN демонструють високу здатність до навчання на складних наборах даних, що дозволяє генерувати реалістичні зображення. DM, своєю чергою, забезпечують високу стабільність та контрольованість процесу генерації.

Проаналізовано існуючі методи покращення генерації зображень, що включають оптимізацію архітектури нейронних мереж, покращення методів навчання та обробки даних, додавання вагових параметрів до тексту, а також використання спеціальних технік для покращення якості зображень. Аналіз показав, що комбінація подібних методів дозволяє суттєво підвищити якість генерованих зображень та зменшити час, необхідний для їх створення.

Розроблено програмне забезпечення для навчання та тестування генеративно-змагальної мережі DCGAN на основі датасету MNIST. Реалізовано оптимізаційні методи, включаючи Self-Attention для покращення глобальної деталізації, Minibatch Discrimination для уникнення колапсу мод, Pixel Normalization для стабільності сигналів, Feature Matching Loss для покращення відповідності ознак, а також автоматизований підбір гіперпараметрів за допомогою Skopt. Інтегровано механізми збереження результатів, логування до TensorBoard і обчислення FID для оцінки якості.

Наукова новізна полягає в тому, що у роботі розроблено та реалізовано підхід до оптимізації генеративних нейронних мереж, який поєднує сучасні методи вдосконалення навчання і автоматичного налаштування гіперпараметрів. Це дозволило підвищити якість, варіативність та ефективність процесу генерації зображень.

Експерименти підтвердили, що впровадження запропонованих підходів дозволяє досягти суттєвого зниження FID, підвищити якість і різноманітність згенерованих зображень, та забезпечити стабільність процесу навчання. Застосування автоматизованого підбору гіперпараметрів значно спростило процес оптимізації моделі, дозволяючи скоротити час навчання. Отримані результати демонструють ефективність впроваджених методів у задачах генерації зображень та створюють основу для подальших удосконалень у цій галузі.

Проведене дослідження підтвердило, що оптимізація процесу генерації зображень за допомогою нейронних мереж є перспективним напрямком, який відкриває нові можливості у різних сферах, таких як мистецтво, дизайн, медицина та розваги.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jonathan Ho. Denoising Diffusion Probabilistic Models [Електронний ресурс] / Jonathan Ho, Ajay Jain, Pieter Abbeel // arXiv.org. – 2020. – Режим доступу : www/ URL: https://arxiv.org/abs/2006.11239v2

2. Ming Tao. DF-GAN: A Simple and Effective Baseline for Text-to-Image Synthesis [Електронний ресурс] / Ming Tao, Hao Tang, Fei Wu, Xiao-Yuan Jing, Bing-Kun Bao, Changsheng Xu // arXiv.org. – 2020. – Режим доступу : www/ URL: https://arxiv.org/abs/2008.05865

3. Shounak Chatterjee. DiffMorph: Text-less Image Morphing with Diffusion Models [Електронний ресурс] // arXiv.org. – 2024. – Режим доступу : www/ URL: https://arxiv.org/abs/2401.00739

4. Wenyi Mo. Dynamic Prompt Optimizing for Text-to-Image Generation [Електронний ресурс] / Wenyi Mo, Tianyu Zhang, Yalong Bai, Bing Su, Ji-Rong Wen, Qing Yang // arXiv.org. – 2024. – Режим доступу : www/ URL: https://arxiv.org/abs/2404.04095v1

5. Alec Radford. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [Електронний ресурс] / Alec Radford, Luke Metz, Soumith Chintala // arXiv.org. – 2015. – Режим доступу : www/ URL: https://arxiv.org/abs/1511.06434

6. Tim Salimans. Improved Techniques for Training GANs [Електронний ресурс] / Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen // arXiv.org. – 2016. – Режим доступу : www/ URL: https://arxiv.org/abs/1606.03498

7. Han Zhang. Self-Attention Generative Adversarial Networks [Електронний ресурс] / Han Zhang, Ian Goodfellow, Dimitris Metaxas, Augustus Odena // arXiv.org. – 2018. – Режим доступу : www/ URL: https://arxiv.org/abs/1805.08318

8. Tero Karras. Progressive Growing of GANs for Improved Quality,

Stability, and Variation [Електронний ресурс] / Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen // arXiv.org. – 2017. – Режим доступу : www/ URL: https://arxiv.org/abs/1710.10196

9. John Jenkins. Exploring deep convolutional generative adversarial networks (DCGAN) in biometric systems: a survey study [Електронний ресурс] / John Jenkins, Kaushik Roy // link.springer.com. – 2024. – Режим доступу : www/ URL: https://link.springer.com/article/10.1007/s44163-024-00138-z

10. Asumi Yamazaki. Two-View Mammogram Synthesis from Single-View Data Using Generative Adversarial Networks [Електронний ресурс] / Asumi Yamazaki, Takayuki Ishida // Applied Sciences. – 2022. – Vol. 12, No. 23, Article 12206. – Режим доступу : www/ URL: https://www.mdpi.com/2076-3417/12/23/12206

11. Hyeeun Ku. TextControlGAN: Text-to-Image Synthesis with Controllable Generative Adversarial Networks [Електронний ресурс] / Hyeeun Ku, Minhyeok Lee // Applied Sciences. – 2023. – Vol. 13, No. 8, Article 5098. – Режим доступу : www/ URL: https://www.mdpi.com/2076-3417/13/8/5098

12. Sik-Ho Tsang. Review – Progressive GAN: Progressive Growing of GANs for Improved Quality, Stability, and Variation [Електронний ресурс] // medium.com. – 2023. – Режим доступу : www/ URL: https://sh-tsang.medium.com/review-progressive-gan-progressive-growing-of-gans-for-improved-quality-stability-and-d3731ca597ea

13. Jay jiyani. Mini-Batch Standard Deviation [Електронний ресурс] // medium.com. – 2022. – Режим доступу : www/ URL: https://medium.com/@jayjiyani1999/mini-batch-standard-deviation-progan-deca475b2aa3

14. Weng, L. What are Diffusion Models? [Електронний ресурс] // Lilian Weng's Blog. – 2021. – Режим доступу : www/ URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

15. Jason Brownlee. How to Implement the Frechet Inception Distance (FID) for Evaluating GANs [Електронний ресурс] // machinelearningmastery.com.

– 2019. – Режим доступу : www/ URL: https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

16. Фесенко А. В. Методи оптимізації генерації зображень за допомогою нейронних мереж / А. В. Фесенко // Т. 6 : «Інформаційні інтелектуальні системи» : Матеріали тез 28-ого міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті» Харків, 16-18 квітня, 2024. – Харків, 2024. – С. 89-90.

17. Tushar Aggarwal. Master the Power of scikit-optimize: A Step-by-Step Guide [Електронний ресурс] // medium.com. – 2023. – Режим доступу : www/ URL: https://medium.com/python-in-plain-english/master-the-power-of-scikit-optimize-a-step-by-step-guide-4346d3a484ea

18. Jakub Czakon. Scikit Optimize: Bayesian Hyperparameter Optimization in Python [Електронний ресурс] // neptune.ai. – 2023. – Режим доступу : www/ URL: https://neptune.ai/blog/scikit-optimize

19. Kuan Hoong. How to use TensorBoard with PyTorch [Електронний ресурс] // medium.com. – 2023. – Режим доступу : www/ URL: https://kuanhoong.medium.com/how-to-use-tensorboard-with-pytorch-e2b84aa55e67