Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження генеративно-змагальних моделей в задачах синтезу тексту в зображення
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-2
Воронюк К. Л.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

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

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Рябова Н. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Воронюк Кристині Леонідівні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження генеративно-змагальних моделей в задачах синтезу тексту в зображення

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 травня 20 23 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових проектів щодо розробки та дослідження різних моделей зіставлення зображень з їх текстовою анотацією, пошук та аналіз існуючих аналогів, реалізація алгоритмів генеративно-змагальної моделі для синтезу тексту в зображення, проведення тестування.

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної області

2) Огляд існуючих моделей GAN для синтезу тексту в зображення

3) Опис складеної генеративно-змагальної моделі для вирішення задачі синтезу

4) Програмна реалізація та аналіз отриманих результатів

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

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	3.04.2023-7.04.2023	виконано
2	Аналіз предметної області і постановка задачі	7.04.2023-11.04.2023	виконано
3	Дослідження моделей для вирішення задач синтезу	11.04.2023-14.04.2023	виконано
4	Вибір та експериментальне дослідження однієї моделі	14.04.2023- 18.04.2023	виконано
5	Розробка вимог до моделі	18.04.2023	виконано
6	Програмна реалізація	18.04.2023-23.04.2023	виконано
7	Аналіз результатів	23.04.2023-25.04.2023	виконано
8	Оформлення пояснювальної записки	25.04.2023-05.05.2023	виконано
9	Попередній захист	17.05.2023	виконано
10	Захист перед ЕК	19.05.2023	виконано
			виконано
			виконано

Дата видачі завдання 3 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Рябова Н. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 81 с., 24 рис., 4 дод., 26 джерел.

ГЕНЕРАТИВНЕ ГЛИБОКЕ НАВЧАННЯ, GNN, T2I, ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МОДЕЛІ, KERAS, TENSORFLOW, МЕРЕЖА STACKGAN, PYTHON, СИНТЕЗ ТЕКСТУ У ЗОБРАЖЕННЯ.

Об’єкт дослідження – процес створення зображень за допомогою генеративно-змагальних моделей, використовуючи текстові описи як вхідні дані.

Предмет дослідження – види генеративно-змагальних моделей та їх архітектур для вирішення задач трансформації тексту у зображення з високою якістю, що являє собою основу даних, на яких потім навчатимуться інші системи.

Мета роботи – ознайомлення, вибір та дослідження особливостей генеративного моделювання, що визначає найбільш ефективні моделі в навчанні генерації реалістичних зображень з текстового опису, а також перевірка якості та швидкості процесу генерації за допомогою реалізації алгоритму для демонстрації отриманих результатів.

Методи дослідження – аналіз існуючих алгоритмів видів генеративно-змагальних моделей. Виділення ключових понять і алгоритмів їх роботи, а також аналіз літератури та електронних ресурсів існуючих змагальних моделей для вирішення задач синтезу тексту у зображення.

На основі результатів виконаних досліджень вирішено задачу трансформації тексту в зображення за допомогою генеративно-змагальної нейронної мережі StackGAN. Розроблено програмний модуль з реалізацією імітаційного моделювання роботи змагальних моделей, який ілюструє синтез.

ABSTRACT

Explanatory note: 81 p., 24 fig., 4 ann., 26 sources.

GENERATIVE DEEP LEARNING, GNN, T2I, GENERATIVE ADVERSIONAL MODELS, KERAS, TENSORFLOW, STACKGAN NETWORK, PYTHON, TEXT TO IMAGE SYNTHESIS.

The object of study is the process of creating images using generative-competitive models, using textual descriptions as input.

The subject of the study is the types of generative-adversarial models and their architectures for solving the problems of transforming text into a high-quality image, which is the basis of data on which other systems will then learn.

The purpose of the work is to familiarize, select and study the features of generative modeling, which determines the most effective models in training the generation of realistic images from a text description, as well as checking the quality and speed of the generation process by implementing an algorithm to demonstrate the results obtained.

Research methods analysis of existing algorithms for the types of generative-adversarial models. Identification of key concepts and algorithms for their work, as well as analysis of literature and electronic resources of existing adversarial models for solving problems of text-to-image synthesis.

Based on the results of the research, the problem of transforming text into an image was solved using the generative adversarial neural network StackGAN. A software module has been developed with the implementation of simulation modeling of the operation of competitive models, illustrating the synthesis.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НМ – нейронна мережа;

ШІ – штучний інтелект;

AttnGAN – Attentional Generative Adversarial network – уважна генеративна змагальна мережа;

CGAN – Conditional Generative Adversarial Nets – умовно-генеративні змагальні мережі;

FusedGAN – Fused Generative Adversarial Network – зрощена генеративна змагальна мережа;

GAN – Generative Adversarial Network – генеративно-змагальна мережа;

GLAM – Global-local collaborative Attentive Module – глобально-локальний спільний уважний модуль для каскадування генерація зображення;

MirrorGAN – Generative Adversarial Network by redescription – генеративна змагальна мережа шляхом переопису;

Obj-GAN – Object-Driven Attentive Generative Adversarial Network – Об'єктно-керована генеративна змагальна мережа з увагою;

ProGAN – Progressive Generative Adversarial Network – прогресивна генеративна змагальна мережа;

StackGAN – Stacked Generative Adversarial Network – стекована генеративна змагальна мережа;

STEM – Semantic Text Embedding Module – модуль семантичного вбудовування тексту;

STREAM – Semantic Text Regeneration and Alignment Module – модуль семантичної регенерації та вирівнювання тексту;

StyleGAN – Style Generative Adversarial Network – стильна генеративна змагальна мережа.

ВСТУП

В останні роки обробка природної мови та комп'ютерний зір стали популярними напрямками досліджень з появою штучного інтелекту та глибокого навчання. Перетворення тексту в зображення, ключова проблема в цій галузі, також привернула увагу і дослідження багатьох вчених, тому що коли люди чують або читають будь-що, вони відразу ж малюють уявні картини, візуалізуючи вміст у своїй голові.

Здатність візуалізувати та зрозуміти складний зв’язок між візуальним світом і мовою настільки природнє, що ми рідко замислюємося про неї. Візуальні уявні образи або «бачення розумовим оком» також відіграють важливу роль у багатьох когнітивних процесах, таких як пам’ять, просторова навігація та міркування. Із-за натхнення тим, як люди візуалізують сцени, була побудована система, яка розуміє взаємозв’язок між баченням і мовою, і яка може створювати зображення, що відображають значення текстових описів, і це є важливою віхою на шляху до людського інтелекту.

За останні кілька років програми комп’ютерного бачення та методи обробки зображень значно виграли в прогресі, який стало можливим завдяки прориву глибокого навчання. Одним із них є область синтезу зображень, яка є процесом створення нових зображень і маніпулювання існуючими. Синтез зображень є цікавим і важливим завданням через велику кількість практичних застосувань, таких як створення мистецтва, редагування зображень, віртуальна реальність, відеоігри та автоматизований дизайн .

Синтез тексту зображення (Text-to-Image Synthesis) – це процес генерації зображення з урахуванням текстового опису. Тобто система отримує на вхід текстовий рядок, який описує об'єкт, та повертає відповідне зображення цього об'єкта [3].

Поява генеративних змагальних мереж (GAN) зробила можливим

навчання генеративних моделей для зображень повністю без нагляду. GAN викликали великий інтерес і передові дослідницькі зусилля щодо синтезу зображень. Вони представили задачу синтезу зображення як гру двох гравців двох конкуруючих штучних нейронних мереж. Мережа генератора навчена виробляти реалістичні зразки, тоді як мережа дискримінатора навчена розрізняти реальні та згенеровані зображення. Навчальна мета генератора – обдурити дискримінатор [4]. Цей підхід був успішно адаптований для багатьох застосувань, таких як синтез людських обличь з високою роздільною здатністю, малювання зображення, доповнення даних, передача стилю.

Подальший розвиток у цій галузі дозволив розширити ці підходи для вивчення умовних генеративних моделей. Інтуїтивно зрозумілий інтерфейс для умовного синтезу зображень може бути створений за допомогою текстових описів, мотивований тим, як люди малюють уявні картини. У порівнянні з мітками, текстові описи можуть нести щільну семантичну інформацію про наявні об’єкти, їх атрибути, просторові аранжування, зв’язки та дозволяють представити різноманітні і детальні сцени.

Синтез тексту у зображення може мати безліч застосувань, включаючи створення зображень для ігор, віртуальної реальності, автоматичного створення ілюстрацій і багато іншого. Однак, як і будь-яка інша технологія, вона також може мати свої обмеження та потенційні ризики, які необхідно враховувати.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Предпроектне обстеження

1.1.1 Загальне визначення генеративно-змагальних мереж

Завдяки останнім досягненням у науці і техніці ми можемо створювати машини, здатні малювати оригінальні картини в певному стилі, писати абзаци зв'язного тексту з структурою, що добре простежується, складати музику, яку приємно слухати, і розробляти виграшні стратегії для складних ігор, генеруючи сцени. І це лише початок генеративної революції.

Автоматичне створення реалістичних високоякісних зображень з текстових описів це дуже цікава тема і досить корисна можливість, тому що цей напрямок має безліч практичних застосувань, але сучасні системи штучного інтелекту все ще далекі від цієї мети, оскільки це досить складне завдання в галузі комп'ютерного зору [5]. Проте в останні роки було розроблено універсальні та потужні рекурентні архітектури нейронних мереж для вивчення різних уявлень текстових ознак. Тим часом, глибокі згорткові генеративні змагальні мережі (GANs) почали генерувати досить переконливі зображення певних категорій, таких як обличчя, обкладинки альбомів та інтер'єри кімнат. Зразки, що генеруються існуючими підходами «текст-зображення», можуть приблизно відображати зміст даних описів, але вони не містять необхідних деталей та яскравих частин об'єкта.

Генеративно-змагальні мережі, або скорочено GAN, є підходом до генеративного моделювання з використанням методів глибокого навчання, таких як згорткові нейронні мережі.

Генеративне моделювання – це неконтрольована навчальна задача в галузі машинного навчання, яка включає автоматичне виявлення та вивчення закономірностей або закономірностей у вхідних даних таким чином, щоб модель можна було використовувати для створення або

виведення нових прикладів, які, ймовірно, могли бути взяті з вихідного набору даних.

GAN – це метод генеративного моделювання, який автоматично вивчає та виявляє закономірності у вхідних даних, створюючи правдоподібні результати на основі вихідного набору даних [6]. Тобто можна сказати, що це розумний спосіб навчання генеративної моделі шляхом формулювання проблеми як завдання навчання з учителем із двома підмоделями: моделлю генератора, яку ми навчаємо генерувати нові приклади, та моделлю дискримінатора, яка намагається класифікувати приклади як реальні (з домен) чи підроблені (згенеровані). Дві моделі навчаються разом у грі з нульовою сумою, змагальною, доки модель дискримінатора не буде обдурена приблизно в половині випадків, що означає, що модель генератора генерує правдоподібні приклади.

1.1.2 Історія розвитку GAN

Клас GAN нейронних мереж, був представлений у 2014 році у роботі «Generative Adversarial Nets» Іена Гудфеллоу та його колег з університету Лавласа в Монреалі. Ця стаття стала відправною точкою для розвитку GAN та відкрила нову еру в галузі генеративних моделей.

У перших роботах Гудфеллоу та його колег мережі GAN були використані для генерації зображень, але пізніше вони знайшли широке застосування в інших областях, таких як обробка природної мови, створення музики, генерація тривимірних моделей тощо.

З того часу було запропоновано безліч модифікацій та покращень GAN, таких як DCGAN (Deep Convolutional GAN), WGAN (Wasserstein GAN), StyleGAN та багато інших. Сьогодні GAN вважаються одними з найбільш перспективних та ефективних методів генерації контенту та мають широке застосування у різних галузях, від науки та технологій до мистецтва та розваг.

1.1.3 Загальна архітектура GAN

Генеративно-змагальні мережі (GAN) є зрушенням в архітектурі глибоких нейронних мереж. Використання цієї архітектури має кілька переваг: вона узагальнює обмежені дані, створює нові сцени з невеликих наборів даних та робить змодельовані дані більш реалістичними. Це важливі теми глибокого навчання, тому що сьогодні багато методів потребують великих обсягів даних. Використовуючи цю нову архітектуру, можна значно скоротити обсяг даних, необхідні виконання цих завдань. У крайніх випадках ці типи архітектур можуть використовувати 10% даних, необхідні інших типів завдань глибокого навчання.

Високорівневий опис потоку генеративно-змагальної мережі, що показує основні функції у блочному форматі зображений на рисунку 1.1.

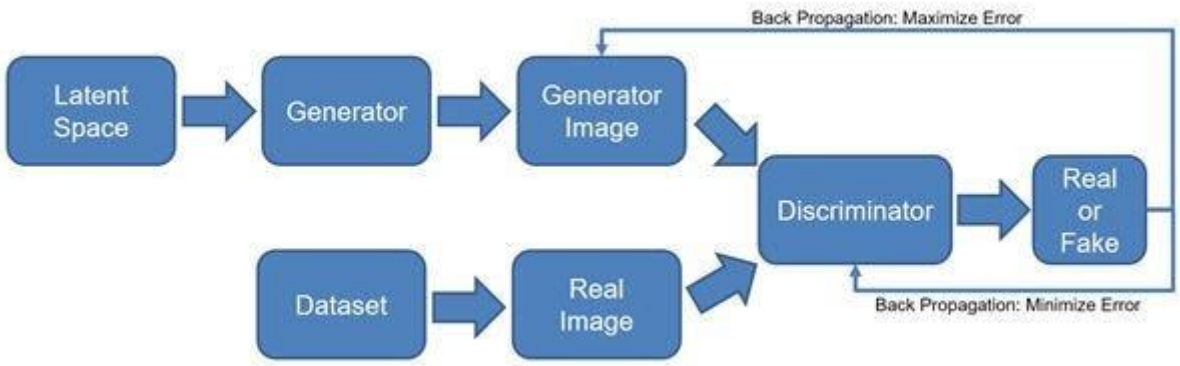

Рисунок 1.1 – Загальна архітектура потоку генеративно-змагальної мережі

У рамках GAN генератор почне навчатися разом із дискримінатором; дискримінатор повинен тренуватися протягом кількох епох до початку змагального навчання, оскільки дискримінатор повинен мати можливість фактично класифікувати зображення. Є ще одна остання частина цієї структури, яка називається функцією втрат. Функція втрат забезпечує критерії зупинення процесів навчання генератора і дискримінатора.

Ця архітектура є змагальною, тому що генератор та дискримінатор працюють один проти одного з протилежними цілями: одна модель намагається імітувати реальність, а інша намагається ідентифікувати підробки. Ці два компоненти тренуються одночасно, покращуючи свої можливості з часом. Вони можуть навчитися ідентифікувати та відтворювати складні навчальні дані, такі як зображення, аудіо та відео.

GAN використовує цей базовий робочий процес:

– генератор приймає вхідні дані, що містять довільні числа;

– генератор обробляє введення для створення зображення;

– дискримінатор приймає зображення, згенероване генератором, та додаткові, реальні зображення;

– дискримінатор порівнює весь набір зображень і намагається визначити, які зображення є справжніми, а які підроблені;

– дискримінатор повертає прогноз для кожного зображення, використовуючи число від 0 до 1, щоб виразити вірогідність. Оцінка 0 показує підроблене зображення, а 1 показує реальне зображення.

Цей робочий процес створює безперервний цикл зворотний зв'язок. Дискримінатор визначає основну істину (емпіричну істину) для вхідних зображень, а генератор передає дискримінатору нові та покращені згенеровані зображення.

З цією архітектурою потрібно розбити кожну частину на складові технології: генератор, дискримінатор та функцію втрат.

1.1.4 Архітектура генератора

Генератор – з самої назви можна зрозуміти, що це генеративний алгоритм. Генератор – це зворотна нейронна мережа, він робить прямо протилежне тому, що робить CNN, тому що в CNN фактичне зображення дається як вхідні дані, а класифікована мітка очікується як висновок, але в

Generator випадковий шум (вектор, що має деяке значення, якщо бути точними) дається як вхідні дані для цієї зворотної CNN, а фактичне зображення очікується як вихідні дані. Простіше кажучи, він генерує дані частини даних, використовуючи власну уяву.

На наступній схемі представлені частини генератора (рисунок 1.2).

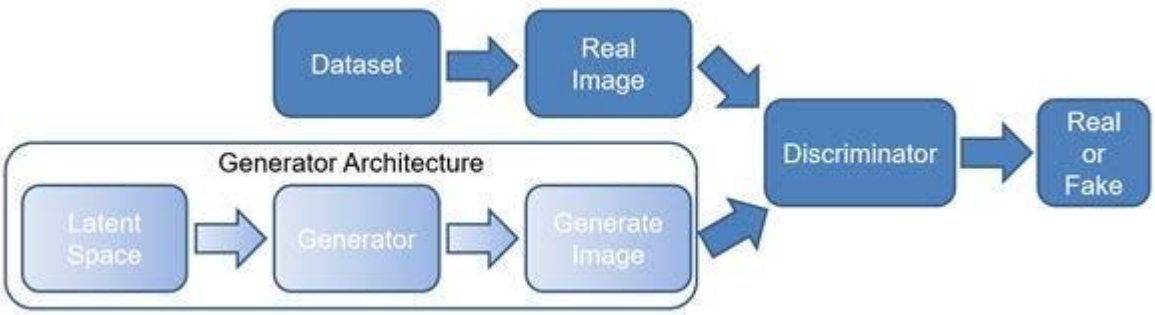

Рисунок 1.2 – Архітектура генератора

Компоненти генератора на діаграмі архітектури: прихований простір, генератор та генерація зображення генератором.

Щоб описати, створення генератора концептуально, треба кілька кроків:

– генератор робить вибірку з прихованого простору і створює зв'язок між прихованим простором та вихідними даними;

– створюється нейронна мережа, яка переходить від входу (прихованого простору) до виходу (зображення для більшості прикладів);

– навчаємо генератор у змагальному режимі, де ми з'єднуємо генератор та дискримінатор разом у моделі;

– останній крок – використання генератора для виведення після навчання.

Модель генератора приймає випадковий вектор фіксованої довжини як вхідні дані і генерує вибірку в області.

Вектор береться випадково з розподілу Гауса, і вектор використовується для засіву генеративного процесу. Після навчання точки у цьому багатовимірному векторному просторі відповідатимуть точкам у проблемній галузі, формуючи стисле уявлення розподілу даних.

Цей векторний простір називається прихованим або векторним простором, що складається з прихованих змінних . Приховані змінні або приховані змінні – це змінні, важливі для предметної області, але безпосередньо не спостерігаються [7].

Тобто прихований простір забезпечує стиснення або високо-рівневі концепції необроблених даних, що спостерігаються, таких як розподіл вхідних даних. У випадку GAN модель генератора застосовує значення до точок у вибраному прихованому просторі, так що нові точки, взяті з прихованого простору, можуть бути надані моделі генератора як вхідні дані та використані для створення нових та різних вихідних прикладів.

Тобто моделі машинного навчання можуть вивчати статистичний прихований простір зображень, музики та історій, а потім можуть робити вибірки з цього простору, створюючи нові витвори мистецтва з характеристиками, аналогічними тим, які модель бачила у своїх навчальних даних.

Після навчання модель генератора зберігається та використовується для створення нових вибірок .

1.1.5 Архітектура дискримінатора

Цю частину GAN вважають схожою на те що, що робить CNN. Дискримінатор – це згорткова нейронна мережа, що складається з безлічі прихованих шарів і одного вихідного шару, основна відмінність тут полягає в тому, що вихідний шар GAN може мати тільки два виходи, на відміну від CNN, які можуть мати виходи в залежності від кількості міток, яких він навчений. Вихід дискримінатора може бути або 1, або 0 через спеціально

вибрану функцію активації для цього завдання, якщо вихід дорівнює 1, то надані дані є реальними, а якщо вихід дорівнює 0, то він відноситься до них як до підроблених даних. Дискримінатор навчається на реальних даних, тому він навчається розпізнавати, як виглядають реальні дані та які функції мають класифікувати дані як реальні [8].

Архітектура дискримінатора визначає, чи є зображення реальним чи підробленим. На рисунку 1.3 представлена схема основних компонентів архітектури дискримінатора.

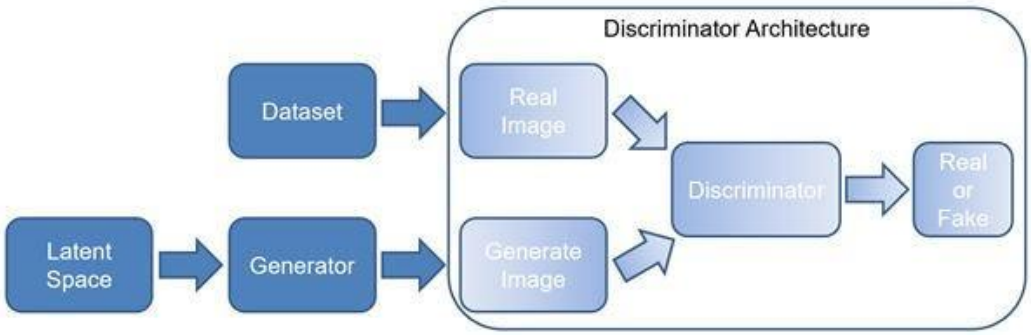

Рисунок 1.3 – Архітектура дискримінатора

Щоб проілюструвати побудову дискримінатора потрібно декілька кроків, як ми будуватимемо:

– спочатку створюється згорткова нейронна мережа для класифікації справжнього чи підробленого (бінарна класифікація);

– потім створюється набір реальних даних та використовується генератор для створення підробленого набору даних;

– навчаємо модель дискримінатора на реальних та фейкових даних;

– останнім кроком треба поєднати навчання дискримінатора з навчанням генератора – якщо дискримінатор показує добрий результат, то генератор буде розходитись.

Дискримінатор може взяти усі корисні функції, які ми маємо з дискримінаційними моделями, та діяти як адаптивна функція втрат для

GAN загалом. Це означає, що дискримінатор може адаптуватися до основного розподілу даних. Це одна з причин, чому сьогоднішні дискримінаційні моделі глибокого навчання настільки успішні – в минулому методи надто сильно покладалися на пряме обчислення деякої евристики базового розподілу даних. Глибокі нейронні мережі сьогодні здатні адаптуватися та навчатися на основі розподілу даних, і метод GAN використовує цю перевагу.

Зрештою дискримінатор оцінюватиме виведення реального зображення та згенерованого зображення на справжність. Реальні зображення спочатку матимуть високі бали за шкалою, тоді як згенеровані зображення матимуть нижчі бали. Зрештою, у дискримінатора виникнуть проблеми з розрізненням згенерованих та реальних зображень. Дискримінатор покладатиметься на побудову моделі і, можливо, початкову функцію втрат.

Іноді генератор можна перепрофілювати, оскільки він навчився ефективно отримувати ознаки з прикладів предметної області.

1.1.6 Практичне застосування моделей GAN

GAN можуть використовуватися для вирішення завдань генерації контенту в різних галузях, де потрібне створення нових даних на основі деяких вхідних даних. Вони дозволяють отримувати реалістичні та якісні результати, які можуть бути використані у різних додатках. Вони використовуються в основному для наступних завдань [9]:

– генерація зображень: GAN можуть генерувати нові зображення на основі деяких вхідних даних, таких, як випадковий шум або текстовий опис об'єкта. Це може бути корисним у різних галузях, таких як комп'ютерні ігри, віртуальна реальність, медичні та рекламні матеріали і т. д;

– обробка природної мови: GAN можуть використовуватися для створення тексту на основі деяких вхідних даних, таких як ключові слова

або речення. Це може бути корисним для створення автоматичних відповідей у чат-ботах або для генерації описів товарів на сайтах електронної комерції;

– створення музики: GAN можуть генерувати нові музичні треки на основі деяких вхідних даних, таких як жанр чи настрій. Це може бути корисним для композиторів та музичних продюсерів, які шукають нові ідеї для своїх проектів;

– генерація тривимірних моделей: GAN можуть генерувати нові тривимірні моделі на основі деяких вхідних даних, таких як опис об'єкта чи зображення. Це може бути корисним у різних областях, таких як архітектура, промисловий дизайн, медичні та наукові дослідження і т. д.

Розповсюдженою і актуальною із цих задач є саме область синтезу зображень, яка є процесом створення нових зображень на основі текстових описів.

1.2 Задача синтезу тексту в зображення

Еволюція генеративно-змагальних мереж (GAN) продемонструвала виняткову продуктивність у синтезі зображень, а саме у надвисока роздільна здатність зображень, доповнення даних та перетворення тексту на зображення.

Термін «текст-в-зображення» (T2I) – це генерація візуально реалістичних зображень із введеного тексту. Генерація T2I – це зворотній процес субтитрів до зображень, також відомий як генерація зображення текст (I2T), тобто генерація текстового опису з вхідного зображення. При генерації T2I модель приймає вхідні дані у вигляді написаного людиною опису та створює зображення RGB, що відповідає опису [10].

Синтез тексту в зображення є складним завданням, яке вимагає поєднання різних методів машинного навчання, таких як обробка природної мови (NLP), комп'ютерний зір та глибоке навчання. Це викликає інтерес у

дослідників у галузі машинного навчання і може призвести до розвитку нових методів та алгоритмів для вирішення задач синтезу тексту у зображення.

Проблема синтезу тексту у зображення є сполучною ланкою між областю обробки природної мови та комп'ютерного зору, що є важливим кроком у розвитку загального розуміння машинного навчання та його можливостей у різних галузях.

1.3 Проблеми при дослідженні використання мережі GAN для синтезу

Проблеми GAN при синтезі тексту у зображення пов'язані з тим, що завдання генерації зображень з текстового опису є досить складним і вимагає врахування багатьох факторів, таких як стиль, кольори, композиція та інші візуальні аспекти.

Однією з основних проблем є необхідність навчання моделей на великій кількості даних, щоб вони могли коректно відтворювати різні стилі та елементи зображення.

Іншою проблемою є складність визначення метрик для оцінки якості зображень, що згенерували. Наприклад, традиційні метрики, такі як PSNR (Peak Signal to Noise Ratio) і SSIM (Structural Similarity Index), можуть бути неефективними для вимірювання якості зображень, що генеруються, оскільки вони не враховують семантичні і контекстуальні аспекти [11].

Також виникають проблеми з урахуванням контексту, особливо коли текстові описи містять складні зв'язки та відносини між об'єктами. Для вирішення цієї проблеми потрібно використовувати складніші та інноваційні підходи, такі як графові нейронні мережі або архітектури Transformer.

Нарешті, однією з останньою проблемою є те, що згенеровані зображення можуть містити неточності та артефакти, що може призвести до погіршення якості їхнього сприйняття. Для вирішення цієї проблеми можна

використовувати методи, такі як додавання регуляризації або використання зворотних зв'язків для покращення якості зображень, що згенерували.

1.4 Пропоноване вирішення проблем

Для вирішення основних проблем у моделях GAN при синтезі тексту в зображення пропонуються наступні рішення:

– збільшення кількості даних: зменшення ризику перенавчання моделі на невеликій кількості даних, можна використовувати різні техніки збільшення обсягу навчальної вибірки, такі як аугментація даних, синтез даних тощо;

– використання складних метрик: для оцінки якості зображень, що генеруються, можна використовувати більш складні та інноваційні метрики, які враховують семантичні та контекстуальні аспекти, такі як FID (Fréchet Inception Distance) і LPIPS (Learned Perceptual Image Patch Similarity);

– використання більш складних моделей: для врахування складних зв'язків та відносин між об'єктами в текстових описах можна використовувати складніші моделі, такі як графові нейронні мережі або архітектури Transformer;

– використання методів покращення якості: для зменшення кількості неточностей та артефактів у згенерованих зображеннях можна використовувати методи покращення якості, такі як додавання регуляризації або використання зворотних зв'язків;

– навчання на великій кількості обчислювальних ресурсів: для прискорення процесу навчання та поліпшення якості зображень, що генеруються, можна використовувати потужні обчислювальні ресурси, такі як графічні процесори (GPU) або тензорні процесори (TPU);

– використання алгоритмів оптимізації: для покращення якості зображень, що генеруються, можна використовувати більш ефективні алгоритми оптимізації, такі як Adam або RMSprop.

В цілому, для вирішення проблем у моделях GAN при синтезі тексту в зображення необхідно використовувати комплексний підхід, який включає використання різних технік і методів, а також облік специфічних вимог завдання генерації зображень з текстового опису.

1.5 Варіанти використання синтезу тексту в зображення

Синтез тексту в зображення за допомогою моделей GAN є цікавим і перспективним напрямом штучного інтелекту. Тому нижче наведено деякі варіанти використання моделей GAN при синтезі тексту у зображення:

– створення аватарів; використовуватися для створення зображень людей на основі їх опису в текстовому форматі. Це може бути корисно для створення аватарів для онлайн-ігор, соціальних мереж та інших програм;

– реконструкція зображень: моделі можна використовувати для відновлення зображень на основі текстових описів. Це може бути корисним, наприклад, для відновлення втрачених картин або фотографій;

– створення ілюстрацій: створення ілюстрацій на основі текстових описів. Це може бути корисним для створення ілюстрацій до книг, коміксів та інших видів літератури;

– створення архітектурних проектів: створення візуалізацій архітектурних проектів на основі текстових описів. Це може бути корисним для архітекторів і дизайнерів інтер'єру;

– створення макетів сайтів: моделі GAN можуть використовуватися для створення візуалізації макетів сайтів на основі текстових описів. Це може бути корисним для веб-дизайнерів;

– створення комп'ютерної графіки: створення комп'ютерної графіки на основі текстових описів. Це може бути корисним для створення спецефектів у кіноіндустрії та ігровій індустрії.

1.6 Постановка завдання дослідження

Виходячи з мети кваліфікаційної роботи, можна поставити такі завдання, що розкривають її тему:

– провести аналіз предметної галузі;

– провести теоретичні дослідження роботи мережі GAN;

– провести теоретичні дослідження існуючих моделей GAN для синтезу тексту в зображення;

– обрати декілька моделей, ознайомитися з принципом роботи алгоритмів;

– обрати датасет для проводження дослідження – вхідні зображення;

– реалізація обраної моделі у Keras та Tensorflow;

– навчання обраних мереж;

– проаналізувати та порівняти результати, отримані після роботи;

– зробити висновки на основі отриманих результатів.

2 ОГЛЯД ІСНУЮЧИХ МОДЕЛЕЙ GAN ДЛЯ СИНТЕЗУ ТЕКСТУ В ЗОБРАЖЕННЯ

Останнім часом активно досліджується синтез зображень із текстових описів за допомогою генеративно-змагальних мереж. Цей підхід дозволяє створювати умовні зображення, що мають високий рівень візуального реалізму, різноманітності та семантичної точності. Незважаючи на значний прогрес у цій галузі за останні роки, він все ще стикається з кількома проблемами, такими як створення зображень з високою роздільною здатністю, що містять кілька об'єктів, та розробка надійних показників оцінки, що відповідають людському сприйняттю.

Генеративні моделі на основі ШІ, які перетворюють текст на зображення, поступово переходять від створення фантастичних зображень до створення реалістичних портретів. Деякі вчені навіть припускають, що мистецтво, створене за допомогою ШІ, може перевершити людську творчість. Більшість сучасних систем перетворення тексту зображення фокусуються на навчанні ітеративної генерації зображень на основі послідовного лінгвістичного введення, як це робить людина-художник.

Цей процес називається генеративною нейронною візуалізацією, який є ключовим процесом для трансформаторів, натхненних процесом поступового перетворення порожнього полотна на сцену. Системи, навчені виконувати це завдання, можуть використовувати переваги генерації одиничних зображень за допомогою текстового опису.

Спільнота, що займається глибоким навчанням, активно розвиває генеративні моделі, серед яких можна виділити три перспективні типи: авторегресійні моделі (AR-model), варіаційні автокодувальники (VAE) та генеративні змагальні мережі. На сьогоднішній день найбільш якісні зображення створюються за допомогою мереж GAN, які створюють фотореалістичні, високодозволені зображення з переконливими деталями та високим ступенем різноманітності [12].

В останні кілька місяців було випущено кілька помітних релізів, деякі з яких були визнані феноменальними вже одразу після їхнього виходу.

Існує кілька моделей GAN для синтезу тексту зображення. Кожна модель має свої переваги та недоліки, і вибір конкретної моделі залежатиме від конкретного завдання та доступних ресурсів.

В наступних розділах проведено дослідження деяких з найбільш відомих моделей в цій галузі.

2.1 Модель AttnGAN

2.1.1 Опис роботи та архітектура

Загальноприйнятий підхід полягає в кодуванні всього текстового опису до глобального векторного простору речень. Такий підхід демонструє ряд вражаючих результатів, але він має головні недоліки: відсутність чіткої деталізації лише на рівні слів і неможливість генерації зображень високого дозволу.

Як розв'язання цієї проблеми було запропоновано нова генеративно-змагальна нейромережа з увагою (AttnGAN), яка належить до уваги як до фактору навчання, що дозволяє виділяти слова для генерації фрагментів зображення. Архітектура мережі зображена на рисунку 2.1.

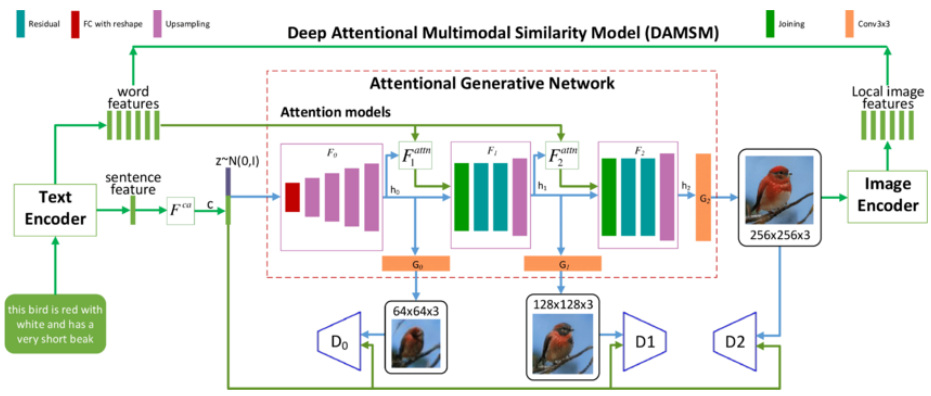

Рисунок 2.1 – Архітектура AttnGAN

Можна побачити, що модель складається з кількох взаємодіючих нейромереж:

– кодувальники тексту (англ. Text Encoder) та зображення (англ. Image Encoder) векторизують вихідний текстовий опис та реальні зображення. У разі текст у вигляді послідовності окремих слів, уявлення яких обробляється разом із поданням зображення, що дозволяє зіставити окремі слова окремим частинам зображення;

– F^{ca} – створює стисле уявлення про загальну сцену на зображенні, з усього текстового описи. Значення C на виході конкатенується з вектором із нормального розподілу Z який задає варіативність сцени. Ця інформація є основою для роботи генератора;

– Attentional Generative Network – найбільша мережа, що складається з трьох рівнів. Кожен рівень породжує зображення все більшої роздільної здатності, від 64×64 до 256×256 пікселів, і результат роботи на кожному рівні коригується за допомогою мереж уваги F^{attn}, які несуть у собі інформацію про правильне розташування окремих об'єктів сцени. Крім того, результати на кожному рівні перевіряються трьома дискримінаторами, що окремо працюють [13]. Приклад результату роботи AttnGAN зображений на рисунку 2.2.

Рисунок 2.2 – Приклад результату роботи AttnGAN

Завдяки модифікаціям нейромережа AttnGAN показує значно кращі результати, ніж традиційні системи GAN. Зокрема, максимальний з відомих показників Inception Score для існуючих нейромереж покращено на 170,25% (з 3,82 до 4,36) на складному наборі даних COCO.

2.1.2 Переваги та недоліки

Переваги AttnGAN:

– більш висока якість зображень, що генеруються, ніж у багатьох інших моделей. Модель здатна генерувати зображення з великою кількістю деталей, тонкою текстурою та вищою роздільною здатністю;

– механізм уваги дозволяє моделі генерувати зображення більш точно та реалістично, враховуючи контекст та співвідношення об'єктів на зображенні;

– модель може працювати з більш складними текстовими описами об'єктів, ніж багато інших моделей.

Недоліки AttnGAN:

– потрібна велика кількість навчальних даних та обчислювальних ресурсів для навчання цієї моделі;

– процес навчання AttnGAN може бути нестабільним та тривалим;

– через використання механізму уваги процес генерації може бути повільним, особливо при роботі з великими зображеннями.

2.2 Модель MirrorGAN

2.2.1 Опис роботи та архітектура

Генерація зображення із заданого текстового опису має дві мети: візуальний реалізм і семантична узгодженість. Ці цілі можна вирішити

пропонуючи нову глобально-локальну уважну структуру перетворення тексту, що зберігає семантику, в зображення – MirrorGAN. Архітектура MirrorGAN зображена на рисунку 2.3.

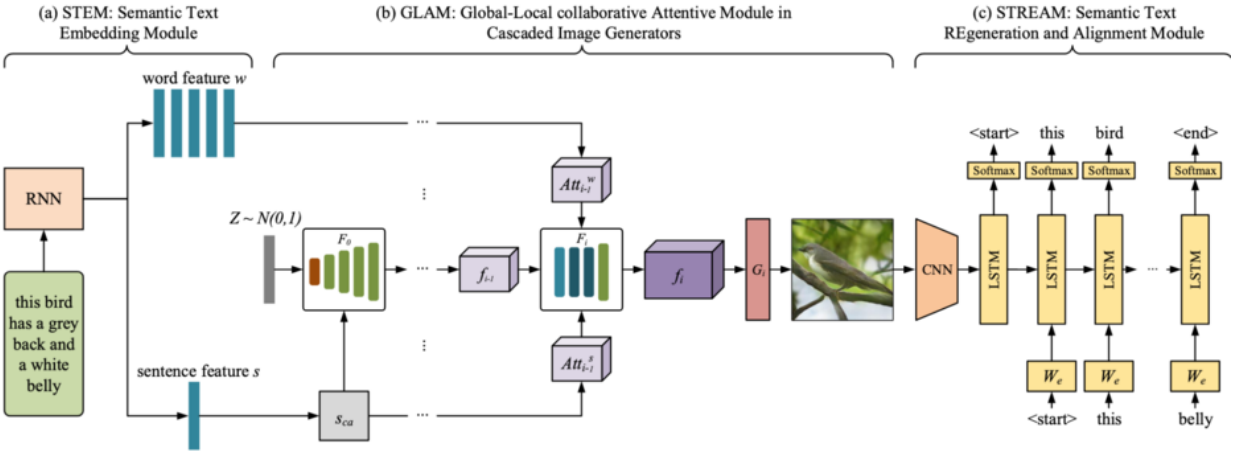

Рисунок 2.3 – Архітектура MirrorGAN

MirrorGAN використовує ідею навчання перетворення тексту на зображення шляхом переопису і складається з трьох модулів: модуля семантичного вбудовування тексту (STEM), локально-глобального спільного уважного модуля для каскадної генерації зображень (GLAM) та семантичної регенерації тексту та модуль вирівнювання (STREAM). STEM генерує вкладення на рівні слів та речень. GLAM має каскадну архітектуру для створення цільових зображень від грубого до дрібного масштабу, використовуючи як локальну увагу до слів, так і глобальну увагу до речень, щоб поступово збільшувати різноманітність та семантичну узгодженість згенерованих зображень. STREAM прагне регенерувати текстовий опис із згенерованого зображення, яке семантично узгоджується з цим текстовим описом [14].

MirrorGAN є дзеркальною структурою, поєднуючи T2I (text-to-image) і I2T (image-to-text). Щоб створити багатоетапний каскадний генератор, всі три мережі генерації зображень (STEM, GLAM і STREAM) необхідно

об'єднати. В якості архітектури STREAM використовують досить поширений фреймворк створення текстового опису зображення (англ. image captioning framework), що базується на кодуванні та декодуванні. Кодувальник зображень – це згорткова нейронна мережа, попередньо навчена на ImageNet, а декодувальник – це рекурентна нейронна мережа. Попереднє навчання STREAM допомогло MirrorGAN досягти більш стабільного процесу навчання та більш швидкої збіжності, в той час, як їхня спільна оптимізація досить нестабільна, займає багато місця і довго працює. Структура кодувальник-декодувальник та відповідні параметри фіксовані під час навчання інших модулів MirrorGAN. На рисунку 2.4 проілюстровано результат порівняння роботи різних моделей.

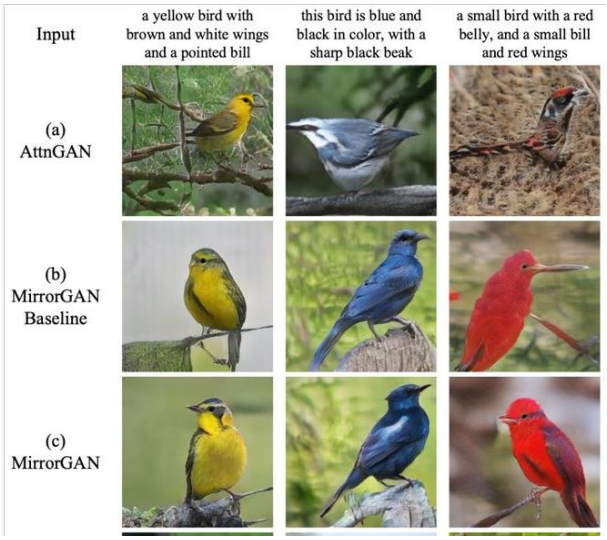

Рисунок 2.4 – Порівняння роботи MirrorGAN, AttnGAN та інших генеративних змагальних мереж

2.2.2 Переваги та недоліки

Переваги MirrorGAN:

– поліпшена якість зображень, що генеруються. Дзеркальна симетрія дозволяє генерувати більш реалістичні та симетричні об'єкти;

– більше ефективне використання ресурсів. MirrorGAN використовує менше параметрів ніж інші архітектури, що робить її більш ефективною.

Недоліки MirrorGAN:

– не підходить всім типів завдань. MirrorGAN націлена на генерацію об'єктів із дзеркальною симетрією, що може бути непридатним для деяких завдань, де об'єкти не є симетричними;

– обмеження у генерації текстових описів. MirrorGAN здатна генерувати зображення лише з урахуванням текстових описів, які мають певну структуру.

2.3 Модель FusedGAN

2.3.1 Опис роботи та архітектура

Для покращення генерації зображень з опису та отримання контрольованої вибірки деякі моделі поділяють процес генерації на кілька етапів. Тому модель FusedGAN може виконувати контрольовану вибірку різних зображень з дуже високою точністю, що також досягається шляхом розбиття процесу генерації зображень на етапи. У цій моделі на відміну від StackGAN, де кілька етапів GAN навчаються окремо з повним контролем помічених проміжних зображень [15]. Модель складається з двох взаємозалежних етапів (рисунок 2.5).

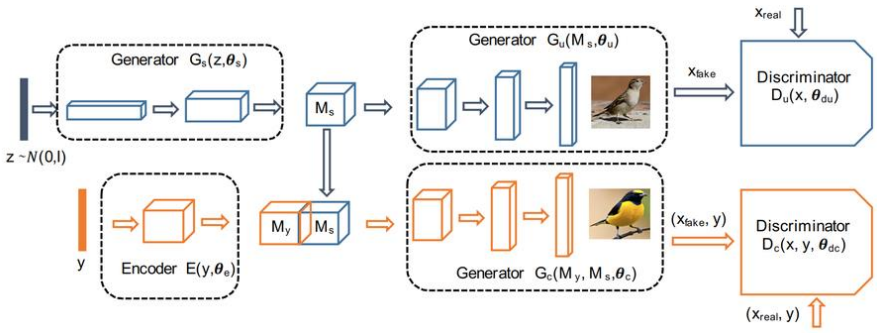

Рисунок 2.5 – Архітектура FusedGAN

На першому етапі за допомогою GAN виконується генерація зображень випадкового вектора, а також створюються ознаки для стилю, в якому буде оформлено згенероване зображення на другому кроці;

На другому етапі CGAN (англ. Conditional Generative Adversarial Nets) генерує остаточне зображення (тобто зображення, що відповідає опису та стилю заданому на першому кроці), використовуючи як вхідні дані текстовий опис та дані отримані з першого кроку.

M_S виступає у ролі шаблону подаючи додаткові ознаки на другий крок генерації. Внаслідок чого зображення згенерованих зображень не тільки відповідають опису, але також зберігають інформацію про стиль. Тому замість того, щоб навчати з нуля, G_c будується поверх M_S, додаючи до нього стилі за допомогою текстового опису. Слід зазначити, що у моделі відсутня явна ієрархія, тому обидва етапи можуть навчатися одночасно, використовуючи альтернативний метод оптимізації. Різницю між роботою FusedGAN та StackGAN зображено на рисунку 2.6.

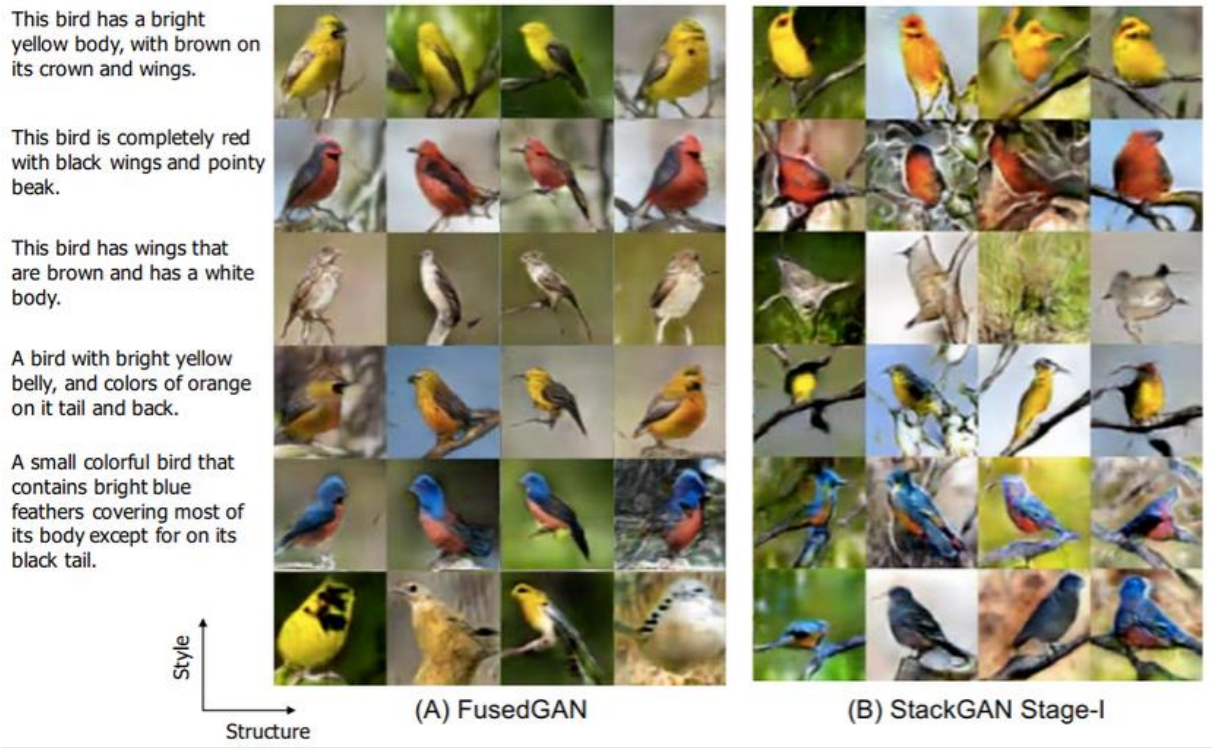

Рисунок 2.6 – Порівняння FusedGAN з StackGAN

2.3.2 Переваги та недоліки

Переваги:

– модель використовує унікальний підхід до навчання, який називається модифікованим підходом адверсарної генерації. Цей підхід дозволяє навчатися швидше та робити більш якісні зображення;

– FusedGAN використовує спеціальний шар згортки, званий «шар FusedConv», який поєднує згортки різних фільтрів для покращення якості зображення.

Недоліки:

– FusedGAN вимагає велику кількість обчислювальних ресурсів для навчання та генерації зображень;

– FusedGAN як і багато інших генеративних моделей, FusedGAN може зіткнутися з проблемою «mode collapse», коли модель генерує лише певні типи зображень, ігноруючи інші.

2.4 Модель StackGAN

2.4.1 Опис роботи та архітетура

Складова генеративна змагальна мережа (англ. Stacked Generative Adversarial Networks, StackGAN) названа так тому, що вона має дві мережі GAN, які об'єднані, щоб сформувати мережу, здатну генерувати зображення з високою роздільною здатністю. Це здійснюється у два етапи: етап I (StageI) та етап II (StageII) [16].

Ці мережі служать для створення фотореалістичних зображень розміру 256x256, заданих текстовими описами. У цій моделі важке завдання генерації зображення розкладається на більш дрібні задачі за допомогою процесу ескіз-уточнення (англ. sketch-refinement process). Таким чином, Stage-I GAN малює примітивну форму та кольори об'єкта на основі даного

текстового опису, отримуючи зображення Stage-I з низькою роздільною здатністю (рисунок 2.7). Stage-II GAN приймає результати Stage-I та текстові описи як вхідні дані та генерує зображення високої роздільної здатності з фотореалістичними деталями. Він здатний виправляти дефекти в результатах етапу I і додавати дрібніші деталі у процесі уточнення (англ. refinement process). Щоб покращити різноманітність синтезованих зображень і стабілізувати навчання CGAN, вводиться техніка умовно-когнітивної регуляції (англ. Conditioning Augmentation), яка сприяє плавності в різноманітті.

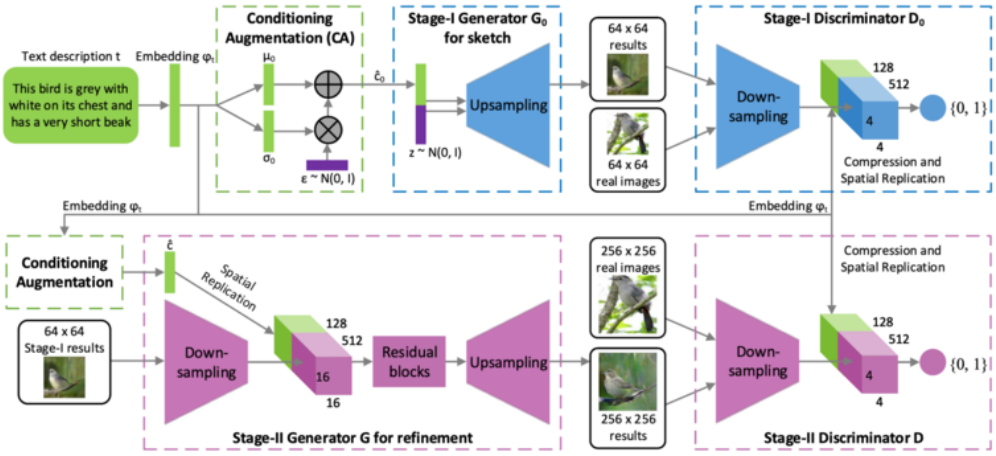

Рисунок 2.7 – Архітектура StackGAN

Вклад та перевага запропонованого методу полягає в наступному [17]:

– пропонується нова складова генеративної змагальної мережі для синтезу фотореалістичних зображень текстових описів. Він розбиває складне завдання генерації зображень з високою роздільною здатністю на дрібніші задачі і значно покращує стан справ. StackGAN вперше генерує зображення з роздільною здатністю 256x256 пікселів з фотореалістичними деталями з текстових описів;

– пропонується техніка Condition Augmentation для стабілізації навчання CGAN, а також для поліпшення різноманітності вибірок, що генеруються;

– великі якісні та кількісні експерименти демонструють ефективність дизайну моделі загалом, а також вплив окремих компонентів, що надають корисну інформацію для розробки майбутніх умовних моделей GAN. Для перевірки методу було проведено великі кількісні та якісні оцінки. Результати роботи моделі порівнюються з двома сучасними методами синтезу тексту на зображення – GAN-INT-CLS та GAWWN (рисунок 2.8).

Рисунок 2.8 – Порівняння методів (згенеровані зображення птахів)

2.4.2 Переваги та недоліки

Переваги StackGAN:

– генерація зображень у кілька етапів: StackGAN використовує двоетапний підхід генерації зображень, що дозволяє більш точно контролювати зображення, що генерується. У першому етапі генерується грубий малюнок, який потім уточнюється другою етапі [18];

– контроль за зображенням, що створюється: StackGAN може бути навчений з керованою умовою вхідних даних, що дозволяє контролювати, які об'єкти повинні з'явитися на зображенні.

Недоліки StackGAN:

– вимагає великої кількості обчислювальних ресурсів: Використання двоетапного підходу генерації зображень потребує значних обчислювальних ресурсів, що робить StackGAN не найшвидшим та найефективнішим підходом;

– складність реалізації: Реалізація StackGAN може бути складним завданням для користувачів-початківців, що вимагає хорошого знання глибокого навчання та відповідних інструментів;

– проблеми з різноманітністю зображень, що генеруються: StackGAN має проблеми з генерацією досить різноманітних зображень, що може бути проблемою для деяких додатків, що вимагають безлічі різних зображень.

2.5 Multi-Scale Gradient GAN

2.5.1 Опис роботи та архітектура

MSG-GAN – це мультимасштабна генеративна змагальна мережа, яка вирішує проблему нестабільності градієнтів, що передаються від дискримінатора до генератора, стаючи неінформативними через дисбаланс у процесі навчання. MSG-GAN використовує ефективну техніку, що дозволяє передавати градієнти від дискримінатора до генератора на кількох масштабах, що допомагає генерувати багатомасштабні зображення, синхронізовані за різними масштабами [19].

У MSG-GAN дискримінатор аналізує не тільки вихідний шар генератора (висока роздільна здатність), але також вихідні шари проміжних рівнів, як показано на рисунку 2.9. Таким чином, дискримінатор стає

функцією декількох масштабних виходів генератора (за допомогою операції конкатенації) і передає градієнти одночасно на всі масштаби.

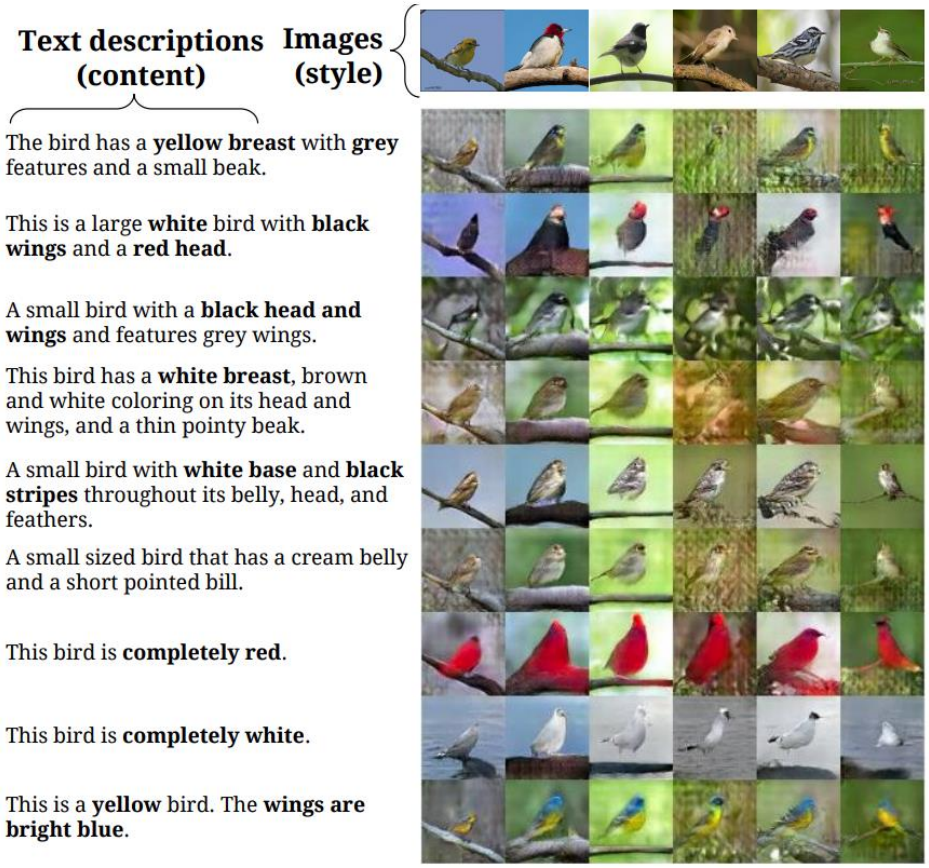

Рисунок 2.9 – Результат роботи MSG-GAN

Архітектура, запропонована в цьому дослідженні, включає сполучення між проміжними рівнями генератора та дискримінатора (рисунок 2.10). Мультимасштабні зображення, що вводяться в дискримінатор, перетворюються на об'єми і поєднуються з відповідними активаційними обсягами з основного шляху згорткових шарів. MSG-GAN виявляє більшу стійкість до змін швидкості навчання та більш стійке підвищення якості зображення порівняно з Pro-GAN.

Ця модель забезпечує однакову швидкість збіжності та узгодженість для всіх дозволів, а зображення, згенеровані на більш високих роздільних здатності, мають симетричні властивості, такі як однаковий колір для обох

очей або сережки в обох вухах. Крім того, фаза навчання допомагає краще розуміти властивості зображень, такі як якість та різноманітність.

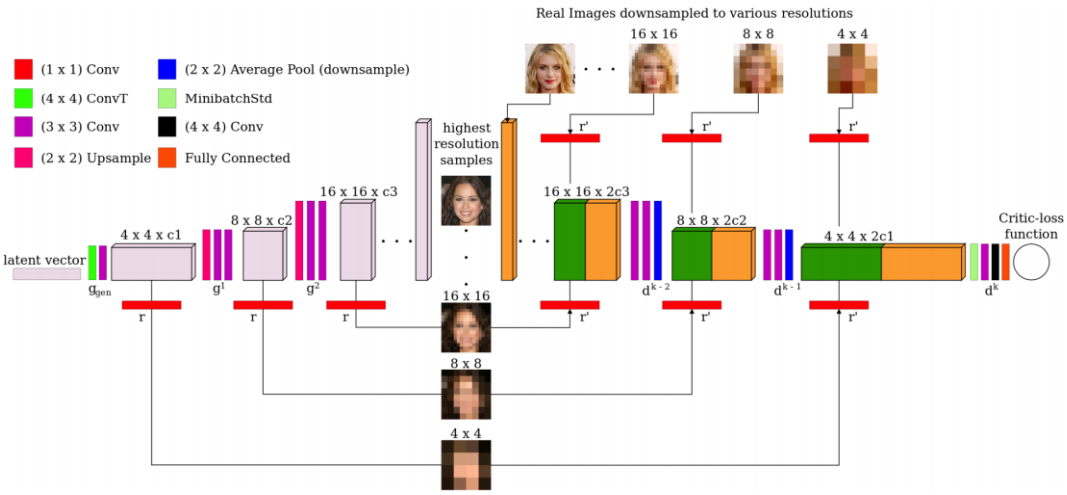

Рисунок 2.10 – Архітектура MSG-GAN для створення синхронізованих багатомасштабних зображень

На початку навчання, шари з нижчою роздільною здатністю генерують блоки однотонних кольорів, але в міру просування навчання воно поширюється на всі шари, і вони починають синхронізуватися, щоб створювати якісніші зразки. У перші кілька секунд навчання зображення осіб починають з'являтися поступово, починаючи з низької роздільної здатності і закінчуючи високою роздільною здатністю. Форма цих осіб спочатку може бути невизначеною та нагадувати краплі.

2.5.2 Переваги та недоліки

Переваги:

– багатомасштабний підхід: модель MSG-GAN використовує багатомасштабний підхід для генерації зображень, що дозволяє моделі генерувати зображення різних розмірів та масштабів з високою якістю;

– простота використання: користувач може вводити випадковий шум у систему та отримувати відповідне зображення без необхідності мати навички роботи з графічними редакторами чи комп'ютерним зором.

Недоліки:

– високі вимоги до обчислювальних ресурсів: MSG-GAN вимагає великої кількості обчислювальних ресурсів для навчання та генерації зображень, що може бути проблематичним для невеликих компаній чи дослідницьких груп;

– необхідність великої кількості даних для навчання: MSG-GAN також вимагає великої кількості даних для навчання, що може бути проблематичним для деяких додатків, особливо якщо доступних даних обмежений.

2.6 Модель DALL-E2

2.6.1 Опис роботи та архітектури

DALL-E2 використовує модель дифузії, яка може маніпулювати різними елементами, такими як тіні, відображення та текстури, і здатна миттєво додавати або видаляти їх. Ця модель вважається перспективною генеративною структурою, що розсуває межі створення зображень і відео. Щоб досягти високоякісних результатів, модель дифузії в DALL-E2 використовує метод керування для оптимізації точності вибірки для фотореалізму за рахунок різноманітності вибірки [20].

DALL-E2 вивчає кореляцію між зображеннями та текстом за допомогою процесу, який називається «дифузія», де він починається з випадкового візерунка точок і поступово переходить до зображення, яке розпізнає особливі особливості зображення. З 3,5 мільярдами параметрів DALL-E2 не такий великий, як GPT-3, і навіть менший за свого попередника DALL-E (який мав 12 мільярдів параметрів). Незважаючи на свій розмір,

модель генерує зображення з роздільною здатністю в чотири рази кращою, ніж DALL-E, і оцінювачі віддають перевагу йому більше ніж у 70% випадків, як для відповідності підписів, так і для фотореалізму. На рисунку 2.11 зображена архітектура моделі DALL-E2.

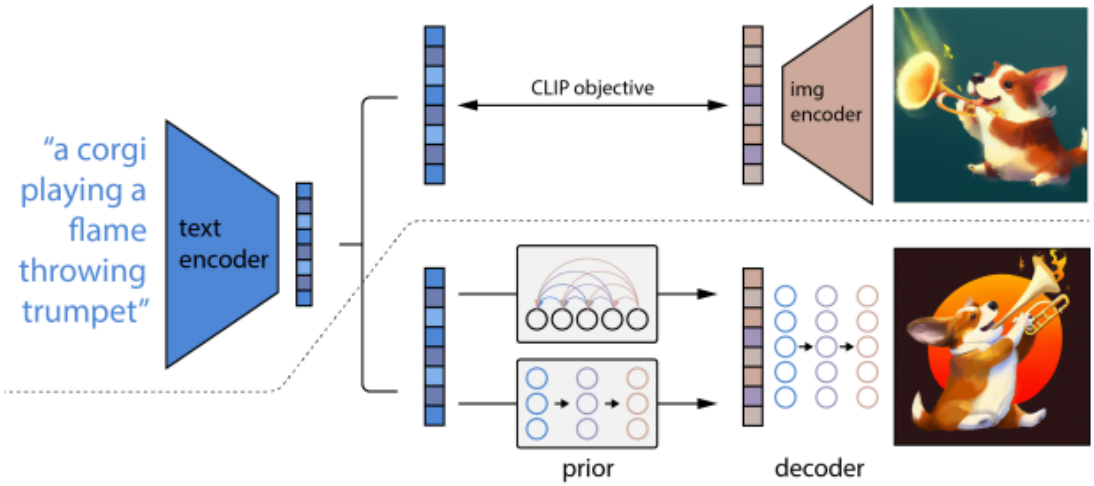

Рисунок 2.11 – Архітектура DALL-E2

Ця універсальна модель, використовуючи вкладення з CLIP, може розширити свої можливості, перевищити рамки генерації запропонованих зображень і створити кілька варіантів вихідних даних, зберігаючи при цьому семантичну інформацію та стилістику. CLIP – це система комп'ютерного перегляду від OpenAI, яка зв'язує текст із зображенням і вбудовує їх в одноприкритий простір, що дозволяє керувати зображеннями за допомогою мови. Порівняно з іншими моделями представлення зображень, CLIP забезпечує більш надійні вкладання, що розширює можливості керування зображеннями за допомогою мови.

Хоча використання вбудовування CLIP для генерації зображень підвищує їхню різноманітність, цей підхід має деякі обмеження. Наприклад, модель unCLIP, яка генерує зображення шляхом інвертування декодера зображень CLIP менш ефективно пов'язує атрибути з об'єктами, ніж аналогічна модель GLIDE. Це тому, що саме вбудовування CLIP не явно

пов'язує характеристики з об'єктами, що призводить до змішування атрибутів і об'єктів при реконструкції зображень з декодера. Однак на більш високих масштабах, які використовуються для створення фотореалістичних зображень, unCLIP забезпечує більшу різноманітність зі збереженням схожості візуального стилю та текстової інформації.

2.6.2 Переваги та недоліки

Переваги:

– якість зображень: DALL-E2 здатний генерувати високоякісні зображення, які можуть бути сприйняті як справжні, що робить його корисним для створення реалістичних віртуальних світів чи допомоги у створенні нових дизайнів;

– автоматичний процес: користувач може ввести текстовий опис у систему та отримати відповідне зображення без необхідності мати навички роботи з графічними редакторами або комп'ютерним зором.

Недоліки:

– необхідність великої кількості даних для навчання: DALL-E2 також вимагає великої кількості даних для навчання, що може бути проблематичним для деяких додатків, особливо якщо доступних даних обмежений;

– обмеження за словниковим запасом: DALL-E2 використовує певний словниковий запас, тому може бути обмежений, які зображення він може створювати. Деякі зображення можуть бути складними для опису тексту, тому DALL-E2 може не впоратися з генерацією цих зображень.

2.7 Модель Obj-GAN

2.7.1 Опис роботи та архітектура

Модель Obj-GAN це система генерації зображень на основі опису з урахуванням об'єктного компонування. Цей об'єктно-керований генератор зображень працює у два етапи: спочатку створюється макет на основі найзначніших слів у текстовому описі, а потім генерується зображення з використанням цієї компонування об'єктів [21]. Щоб точніше співвіднести згенеровані об'єкти з текстовим описом та макетом, у системі використовується новий об'єктний дискримінатор, заснований на Fast RCNN.

Основна мета – створення якісних зображень з урахуванням семантично значущого макета та реалістичних об'єктів. Вона складається з двох основних компонентів: генератора зображень з увагою, керованого об'єктами, та пооб'єктного дискримінатора. Генератор приймає на вхід текстовий опис та семантичний макет та поступово покращує якість зображення. У ході процесу генератор зосереджується на найбільш значущих для об'єкта словах і синтезує фрагменти зображення в межах, що обмежують.

Архітектура Obj-GAN складається з кількох етапів. Спочатку генеративна змагальна мережа приймає текстову пропозицію та створює семантичний макет, який складається з об'єктів з відповідними рамками, що обмежують, і мітками класів. Генератор рамок, що обмежують, і генератор фігур працюють разом, створюючи послідовність рамок, а потім для кожної рамки генерується відповідна фігура. Оскільки кожній рамці зіставлено слово з текстової речення, модель seq2seq використовується для зв'язку між рамками і словами. Потім створюється G_{shape}, який базується на двонаправленій згортковій LSTM. Навчання G_{shape}, ґрунтується на генеративній змагальній мережі, в якій використовується втрата сприйняття

для стабілізації навчання та обмеження генерації фігур. Архітектура зображена на рисунку 2.12.

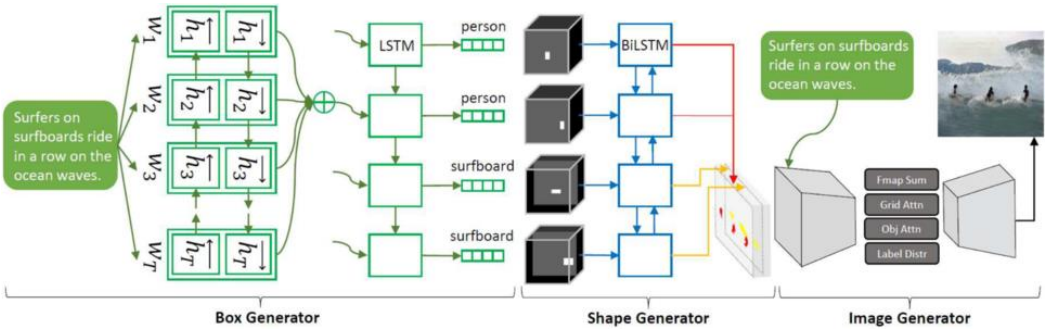

Рисунок 2.12 – Архітектура Obj-GAN

Більш детально, Obj-GAN використовує шар уваги, який керується об'єктами, для роботи з мітками класу та запитує відповідні слова у реченні, щоб сформувати вектор контексту. Порівняння результатів Obj-GAN з іншими генеративними змагальними мережами зображено на рисунку 2.13.

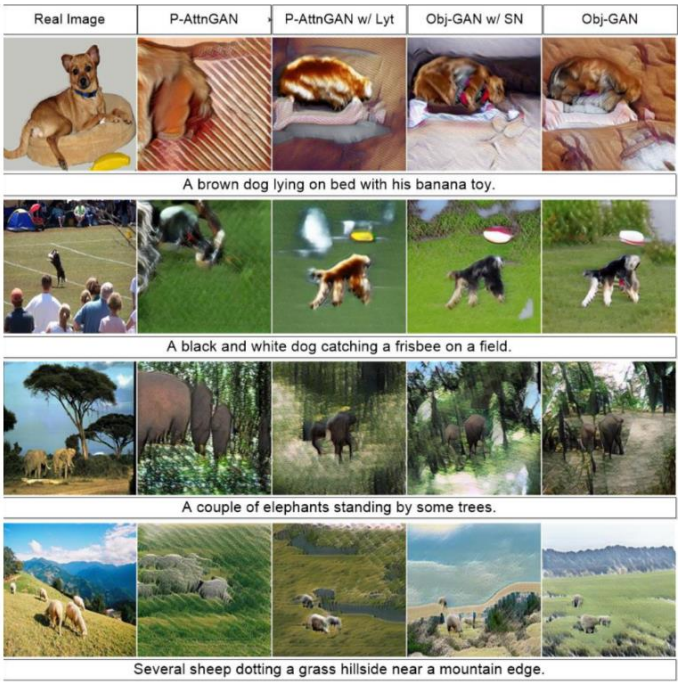

Рисунок 2.13 – Результат роботи Obj-GAN в порівнянні з іншими моделями

2.7.2 Переваги та недоліки

Obj-GAN має такі переваги:

– якість зображення. Obj-GAN дозволяє генерувати високоякісні зображення з семантично важливими макетами та реалістичними об'єктами;

– керована генерація. За допомогою текстових описів та попередньо згенерованих макетів користувач може керувати процесом генерації зображень;

– ефективність. Навчання Obj-GAN ґрунтується на генеративній змагальній мережі, що робить процес навчання відносно швидким та ефективним.

Однак у Obj-GAN також є деякі недоліки:

– складність навчання. Навчання Obj-GAN потребує значних обчислювальних ресурсів та великої кількості даних для навчання;

– обмеженість при генерації зображень. Obj-GAN може генерувати зображення лише на основі заданих макетів та об'єктів, що може обмежувати його застосування у деяких областях;

– необхідність вручну створювати макети. Для створення зображень за допомогою Obj-GAN необхідно створювати вручну макети, що може бути трудомістким процесом.

3 ОПИС СКЛАДЕНОЇ ГЕНЕРАТИВНО-ЗМАГАЛЬНОЇ МОДЕЛІ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ СИНТЕЗУ

У цьому розділі було запропоновано модель Stacked Generative Adversarial Networks (StackGAN) для створення фотореалістичних зображень розміром 256×256 на основі текстових описів. Модель StackGAN представляє собою репрезентативний метод створення зображень із текстових описів. Хоча він може генерувати зображення з високою роздільною здатністю, але як виявилось він має кілька обмежень; деякі із згенерованих зображень зазвичай нерозбірливі, і можуть виникнути колапс режиму. Тому в цьому дослідженні ми прагнули вирішити ці дві проблеми для створення зображень, які більш точно наведені в наступних розділах.

3.1 Введення в модель StackGAN

Синтез високоякісних зображень текстових описів є складним завданням комп'ютерного зору і має безліч практичних додатків. Зразки, створені за допомогою існуючих підходів перетворення тексту на зображення, можуть приблизно відображати зміст даних описів, але вони не містять необхідних деталей та яскравих частин об'єкта. Ці труднощі навчання часто виникають через велику кількість варіацій відповідності між природною мовою та зображеннями. Існує безліч виразів природної мови, що виражають один образ, і також існує безліч образів, що відповідають одному виразу природної мови.

Одним із найдивовижніших результатів у цій галузі є StackGAN – це набір GANs, який може створювати фотореалістичні зображення з високою роздільною здатністю [22]. StackGAN складається з декількох частин, кожна з яких створює зображення більш високої роздільної здатності, починаючи з грубих нарисів і закінчуючи деталізацією на рівні пікселів.

Процес розбитий на два етапи: перший етап, який також називають Stage-I, і другий етап, відомий як Stage-II. Етап-I GAN створює низькорозв'язані зображення, на яких зображені прості форми та кольори об'єктів, виходячи з текстового опису, який йому надається. На другому етапі GAN використовується результат, отриманий на першому етапі, а також текстові описи, і на їх основі створюються зображення з високою роздільною здатністю та фотореалістичними деталями. На цьому етапі модель виправляє помилки, які можуть виникнути на першому етапі та додає нові переконливі деталі, уточнюючи зображення.

Мережа StackGAN можна порівняти з роботою художника: на початку роботи художник малює прості геометричні фігури, такі як круги, лінії та прямокутники, потім заповнює їх квітами та додає все більше деталей, щоб зробити картину більш реалістичною. Аналогічно, в StackGAN на першому етапі створюються низькорозв'язані зображення з примітивними формами на основі текстового опису, а на другому етапі виправляються помилки, виявлені на першому етапі, і додаються більш реалістичні деталі, щоб зробити зображення більш деталізованим. Обидві мережі генератора StackGAN є умовно породжувальними змагальними мережами (cGAN). Перша мережа GAN використовує текстові описи, а друга мережа GAN використовує як текстові описи, і зображення, згенеровані на першому етапі.

3.2 Загальний опис архітектури

Модель StackGAN приймає текстовий опис як вхідні дані та генерує зображення, що відображають її функції, за допомогою двоетапного процесу. Кожен ступінь має два генератори і два дискримінатори. StackGAN складається з багатьох мереж [23]:

– Stack-I GAN: кодувальник тексту, мережа розширення умов, мережа генератора, мережа дискримінатора, вбудована мережа компресії;

– Stack-II GAN: кодувальник тексту, мережа розширення умов, мережа генератора, мережа дискримінатора, вбудована мережа компресії.

Огляд архітектури StackGAN показано на рисунку 3.1.

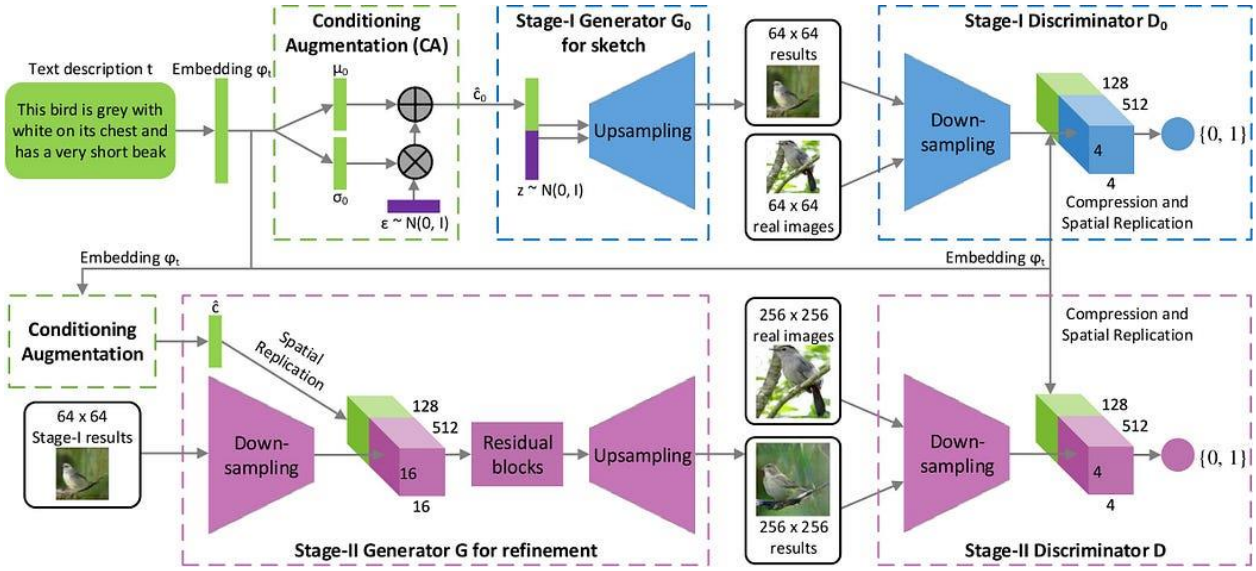

Рисунок 3.1 – Архітектура StackGAN

На цій схемі представлені обидва етапи роботи мережі StackGAN. На першому етапі створюються зображення розміром 64×64 пікселів. Потім на другому етапі ці зображення з низькою роздільною здатністю використовуються для генерації зображень з високою роздільною здатністю розміром 256x256 пікселів. У наступних розділах ми докладніше розглянемо кожен із компонентів мережі StackGAN.

3.3 Мережа кодувальника тексту

Завданням мережі кодувальника тексту є перетворення текстового опису (t) на текстове вкладення (φ_t) з метою подальшого використання в мережі. Мережа кодувальника тексту генерує 1024-мірне вкладення тексту та є спільною для обох етапів мережі StackGAN [24].

Однак у цьому дослідженні ми не навчатимемо мережу кодувальника тексту. Ми будемо використовувати попередньо навчені текстові вкладення як вхідні дані для мережі.

3.4 Позначення та їх опис

Однак щоб розуміти формули та позначення, які використовуються у цьому розділі, опишемо їх в таблиці 3.1.

Таблиця 3.1 – Позначення та їх значення

Позначення	Опис
t	Текстовий опис дійсного розподілу даних
z	Вибірка вектора шуму з гауссового розподілу
φ_t	Вкладення тексту даного текстового опису попередньо навченим кодувальником
$\hat{c}_0$	Текстова змінна умов – гауссова змінна умов, обрана із розподілу. Вона набуває різних значень
$N(\mu(\varphi_t), \sum(\varphi_t))$	Умовний гаусовий розподіл
$N(0,1)$	Нормальний розподіл
$\sum(\varphi_t)$	Діагональна коваріаційна матриця
p_{data}	Справжній розподіл даних
$p(z)$	Розподіл Гауса
D_0	Етап I дискримінатор
G_0	Етап I генератор
D	Етап II дискримінатор
G	Етап II генератор
$N2$	Розміри випадкової змінної шуму
$\hat{c}$	Гауссова прихована змінна для етапу IIGAN

3.5 Мережа розширення умов

Мережа розширення умов (CA) використовує нормальний розподіл $N(\mu(\varphi_t), \sum(\varphi_t))$, щоб вибирати випадкові приховані змінні $\hat{c}$. Блок CA надає безліч переваг:

– додає випадковість у мережу;

– забезпечує надійність генераторної мережі за рахунок захоплення різних об'єктів у різних позах та проявах;

– створює більше пар зображення-текст, що дозволяє створити надійнішу мережу, здатну справлятися з обуреннями.

3.6 Опис мережі на I етапі

На першому етапі процесу генератор створює зображення низької якості, які відображають загальні риси даного текстового опису. Генератор отримує два входи: $\hat{c}$, який є вибіркою зі скритого простору змінних доповнення кондиціонування, і z, який є вибіркою з довільного розподілу, що називається $p(z)$. На цьому етапі дискримінатор не визначає, чи є вхідне зображення автентичним чи ні. Замість цього він класифікує зображення на основі того, наскільки добре вони відповідають текстовому опису. Отже, є три категорії вхідних даних: справжні зображення, які відповідають текстовому опису, справжні зображення, які не відповідають тексту, та зображення, створені штучно.

Мережа StackGAN складається з двох основних компонентів: генератора та дискримінатора. У цьому розділі ми детальніше розглянемо обидві мережі.

3.6.1 Мережа генератора

Мережа генератора для першого етапу є глибокою нейронною мережею згортання з декількома шарами, які дозволяють збільшити дискретизацію зображення. Вона використовує умовну генеративно-змагальну мережу (CGAN), яка залежить від змінної $\hat{c}$ і випадкової змінної z. Мережа генератора приймає гаусову умовну змінну $\hat{c}_0$ і випадковий шумовий вектор z, і створює зображення розміром 64×64×3. Отримане зображення з низькою роздільною здатністю може мати прості форми та основні кольори, а також містити різні дефекти. Тут змінна z є випадковим шумом, обраним з гауссового розподілу p_z з розмірністю N_z. Зображення, створені за допомогою мережі генератора, можуть бути як $s_0 = G_0(z, \hat{c}_0)$.

Мережа генератора включає кілька шарів згортки, за кожним з яких слід шар пакетної нормалізації або шар активації. Основною метою цієї мережі є створення зображень розміром 64×64×3, тому всі шари націлені на досягнення цієї мети.

3.6.2 Мережа дискримінатора

Дискримінатор, подібно до генератора, являє собою глибоку нейронну мережу згортки, що містить послідовність шарів згортки, що знижують дискретизацію. Після застосування цих шарів ми отримуємо карти ознак зображень, які допомагають визначити, чи є зображення реальними (з реальних даних p_{data}) або згенерованими за допомогою генератора.

Щоб об'єднати вкладений текст із картами ознак, ми використовуємо стиснення та просторову реплікацію. Це досягається за рахунок повнозв'язкового шару, який стискає вкладений текст до розміру виходу N_d і перетворює його в тензор розміру $M_d \times M_d \times N_d$ за допомогою

просторової реплікації. Потім карта ознак і вкладений текст поєднуються відповідно до розміру каналу.

Зрештою, ми отримуємо повнозв'язковий шар з одним вузлом, який використовується для двійкової класифікації.

Мережа дискримінатора складається з кількох шарів згортки і її мета полягає в тому, щоб розрізняти справжні зображення реальних даних від зображень, які генерує мережу генератора.

Далі розглянемо втрати, що використовуються на першому етапі мережі StackGAN.

3.6.3 Втрати мережі StackGAN на I етапі

На першому етапі мережі StackGAN використовуються дві функції втрат:

– втрати генератора;

– втрати дискримінатора.

Функція втрат для дискримінатора $\mathcal{L}_D$ визначається так:

$$\mathcal{L}_D = E_{(I_0,t)\sim p_{data}}[logD_0(I_0,\varphi_t) + E_{z\sim p_z,t\sim p_{data}}[log(1 - D_0(G_0(z,\hat{c}_0),\varphi_t))]. \quad (3.1)$$

Цей вираз представляє функцію втрат для мережі дискримінатора, яка використовує вкладення тексту як умову обох мереж.

Функція втрат для генератора $\mathcal{L}_G$ визначається так:

$$\mathcal{L}_{G_0} = E_{z\sim p_z,t\sim p_{data}}[log(1 - D_0(G_0(z,\hat{c}_0),\varphi_t))] + \lambda D_{KL}(N(\mu_0(\varphi_t),\sum\nolimits_0(\varphi_t)) \,||\, N(0,1)). \quad (3.2)$$

Це рівняння описує функцію втрат для мережі генератора з вкладеннями тексту з членом KL-розбіжності та застосовується до обох мереж StackGAN.

3.7 Опис мережі на II етапі

На другому етапі StackGAN також присутні генератор і дискримінатор, при цьому генератор є мережею типу кодер-декодер. На відміну від першого етапу, на другому етапі випадковий шум z не використовується, оскільки на першому етапі була збережена випадковість у вигляді s_0-зображення, яке було створено генератором.

Спочатку необхідно використовувати попередньо навчений кодувальник тексту для генерації гауссівських умовних змінних $\hat{c}_0$, що призводить до генерації того ж вкладеного тексту φ_t. Для розширення умов на етапах I та II використовуються різні повнозв'язкові шари для генерації середніх та стандартних відхилень.

Це означає, що на етапі II GAN навчається отримувати корисну інформацію з текстового вкладення, що не відбувається на етапі I.

Зображення, створювані GAN на етапі I, можуть мати проблеми, такі як відсутність частин об'єкта, спотворення форми та недостатність деталей для досягнення фотореалістичності. У етапі II GAN використовується висновок етапу I GAN у поєднанні з низькодозволяючим зображенням та текстовим описом, що дозволяє створювати зображення високої роздільної здатності та усувати недоліки.

3.7.1 Мережа генератора

Мережа генератора в StackGAN є глибокою нейронною мережею згортки. На першому етапі зображення з низькою роздільною здатністю пропускається через кілька шарів знижувальної дискретизації для створення

ознак зображення, потім об'єднується зі змінними формування тексту відповідно до розміру каналу. Далі, мультимодальні уявлення функцій зображення і тексту вивчаються в блоках різниці, а потім вихідні дані останньої операції використовуються для генерації зображення високої роздільної здатності за допомогою шарів підвищення дискретизації розміром 256×256×3. Цей процес допомагає виправити дефекти, такі як відсутність яскравих частин об'єкта, спотворення форми та відсутність важливих деталей, що зазвичай спостерігається на зображеннях з низькою роздільною здатністю, створених на першому етапі GAN.

Метою мережі генератора на етапі II є генерація зображень високої роздільної здатності із зображень низької роздільної здатності, які генеруються мережею генератора на етапі I. Таким чином, мережа генератора Stage-II не займається ніяким іншим завданням, крім генерації зображень високої роздільної здатності з вхідних зображень низької роздільної здатності.

3.7.2 Мережа дискримінатора

Мережа дискримінатора також є глибокою нейронною мережею згортки і має шари дискретизації, що знижує, для обробки великих зображень. Вона працює як дискримінатор із функцією зіставлення, яка допомагає вирівняти зображення та текстову умову. Під час навчання дискримінатор отримує кілька реальних зображень і відповідних їм текстових описів як позитивні приклади, і навіть дві групи негативних прикладів. Перша група містить реальні зображення з невідповідними текстовими умовами, друга – синтетичні зображення з відповідними текстовими умовами.

3.7.3 Втрати мережі StackGAN на II етапі

Як і в інших моделях GAN, GAN на етапі II генератор G і дискримінатор D можуть бути навчені через максимізацію втрат дискримінатора і мінімізацію втрат генератора.

Формула для втрат для генератора G описується таким чином:

$$\mathcal{L}_G = E_{(I,t)\sim p_{data}}[logD_0(I_0, \varphi_t) + E_{s_t\sim p_G, t\sim p_{data}}[log(1 - D(G(s_t, \hat{c}), \varphi_t))]. \quad (3.3)$$

Це рівняння зрозуміле і є функцією втрат для мережі дискримінатора, яка враховує вкладення тексту в обидві мережі. Одна з головних відмінностей полягає в тому, що мережа генератора приймає на вхід дані s_0 і $\hat{c}$, де s_0 - це зображення, згенероване на етапі I, а $\hat{c}$ є змінною CA.

Формула для втрат для дискримінатора D описується таким чином:

$$\mathcal{L}_D = E_{s_0\sim p_{G_0}, t\sim p_{data}}[log(1 - D(G(s_0, \hat{c}_0), \varphi_t))] + \lambda D_{KL}(N(\mu(\varphi_t), \sum(\varphi_t)) \,||\, N(0,1)). \quad (3.4)$$

Це рівняння представляє функцію втрат для мережі генератора, в якій обидва компоненти залежать від текстового вкладення. Це рівняння також включає дивергенцію Кульбака-Лейблера (KL).

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Опис набору даних для дослідження моделі

Наша модель була навчена на наборі даних CUB_200_2011, який включає понад 11 000 фотографій 200 видів птахів, зафіксованих у природному середовищі. Кожен вид птиці має від 40 до 60 фотографій із різними кутами зйомки та зміною освітлення, що ускладнює завдання класифікації зображень [25].

Крім фотографій, набір даних CUB_200_2011 містить також інструкції, що описують кожен вид птахів. Ці інструкції містять інформацію про наукову назву, поширення, місце проживання та фізичні характеристики кожного виду птиці.

Для StackGAN це означає, що можна навчати генератор зображень з високою роздільною здатністю на цьому наборі даних, використовуючи текстові описи, щоб генерувати зображення птахів з різними ракурсами та освітленням.

Оскільки 80% птахів у цьому наборі даних мають відношення розміру об'єкта до зображення менше 0,5, як етап попередньої обробки ми обрізаємо всі зображення, щоб гарантувати, що рамки птахів, що обмежують, мають відношення розміру об'єкта до зображення більше 0,75.

Додатково були завантажені готові до використання векторні уявлення тексту, отримані з попередньо навченої глибокої нейронної мережі char-CNN-RNN, яка здатна вивчати багато аспектів природної мови, включаючи синтаксис та семантику.

4.2 Реалізація StackGAN в Keras

Keras – це високорівневий API для глибокого навчання, який забезпечує простий та зручний інтерфейс для створення та навчання моделей. Бібліотека Keras надає набір інструментів та утиліт для створення та навчання моделей глибокого навчання, які дозволяють легко реалізовувати складні архітектури мереж. Також ця бібліотека є надбудовою над бібліотеками глибокого навчання, такими як TensorFlow, Theano і CNTK, дозволяє швидко і легко створювати моделі з мінімальною кількістю коду.

У контексті реалізації моделі StackGAN, Keras надає зручний та простий спосіб створення генеративно-змагальної мережі (GAN). Keras має вбудований клас GAN, який спрощує реалізацію та навчання GAN-моделей. Він також пропонує широкий вибір оптимізаторів і функцій втрат, які можна використовувати для навчання моделі.

Реалізація моделі StackGAN через Keras проходить так:

– створення генератора та дискримінатора у вигляді Keras моделей;

– створення екземпляра класу GAN з Keras та додавання генератора та дискримінатора як компонент GAN;

– компіляція моделі GAN з вибраним оптимізатором та функцією втрат;

– навчання моделі GAN на наборі даних за допомогою методу fit().

Keras також надає можливість зберігати та завантажувати навчені моделі, що дозволяє використовувати модель StackGAN для генерації зображень на нових даних у майбутньому.

Реалізація мережі StackGAN з використанням Keras включає розбиття на два етапи: етап I і етап II, кожен з яких має свої унікальні характеристики і параметри. Детальний опис процесу реалізації обох елементів представлено в наступних розділах даної роботи.

4.3 Навчання моделі StackGAN етапу I

4.3.1 Завантаження набору даних

Завантаження даних це процес, що складався з декількох етапів:

– на першому етапі ми завантажуємо ідентифікатори класів (ID) з файлу pickle, потім імена файлів і відповідні їм текстові вкладення також завантажуються з файлу;

– отримуємо рамки, що обмежують, які використовуються для виділення об'єктів з вихідних зображень;

– на третьому етапі створюється метод завантаження та обрізання зображення навколо рамки, що обмежує, а також зміна розміру зображення до заданого розміру;

– поєднавши попередні методи, на четвертому етапі ми отримуємо всі зображення, їх маркування та відповідні текстові вкладення;

– зрештою, на останньому етапі ми завантажуємо дані та робимо їх доступними для навчання.

Коли дані для навчання були успішно завантажені, потрібно було створити декілька моделей.

4.3.2 Створення моделей

У процесі реалізації було застосовано чотири моделі:

– модель генератора, яка відповідає за створення високоякісних зображень на основі вкладення тексту;

– модель дискримінатора, яка класифікує зображення як реальні чи згенеровані;

– модель компресора, яка стискає вкладення тексту та допомагає подати його у вигляді більш компактного та інформативного уявлення;

– змагальна модель, яка поєднує генератор і дискримінатор, і дозволяє їм навчатися взаємодіяти один з одним.

Спочатку треба почати з визначення оптимізаторів, необхідних для навчання. Потім потрібно побудувати та скомпілювати різні мережі. Наступним кроком буде визначити функцію втрат. І нарешті ми маємо готові дані та моделі та можемо почати навчання моделі. Але спочатку був доданий також TensorBoard, щоб запам'ятовувати втрати для їхньої візуалізації.

Програмний код для створення моделей, поданий у лістингу 4.1.

Лістинг 4.1 – Програмний код створення моделей

```
     dis_optimizer=Adam(lr=stage1_discriminator_lr,beta_1=0.5,
beta_2=0.999)
     gen_optimizer=Adam(lr=stage1_generator_lr,beta_1=0.5,
beta_2=0.999)
     ca_model = build_ca_model()
     ca_model.compile(loss="binary_crossentropy", optimizer="adam")
     stage1_dis = build_stage1_discriminator()
     stage1_dis.compile(loss='binary_crossentropy',
optimizer=dis_optimizer)
     stage1_gen = build_stage1_generator()
     stage1_gen.compile(loss="mse", optimizer=gen_optimizer)
     embedding_compressor_model = build_embedding_compressor_model()
     embedding_compressor_model.compile(loss="binary_crossentropy",
optimizer="adam")
     adversarial_model =
build_adversarial_model(gen_model=stage1_gen,
     dis_model=stage1_dis)
     adversarial_model.compile(loss=['binary_crossentropy',
KL_loss],
     loss_weights=[1, 2.0],
     optimizer=gen_optimizer, metrics=None
     def KL_loss(y_true, y_pred):
     mean = y_pred[:, :128]
     logsigma = y_pred[:, :128]
     loss = -logsigma + .5 * (-1 + K.exp(2. * logsigma) +
K.square(mean))
     loss = K.mean(loss)
     return loss
     tensorboard = TensorBoard(log_dir="logs/".format(time.time()))
     tensorboard.set_model(stage1_gen)
     tensorboard.set_model(stage1_dis)
     tensorboard.set_model(ca_model)
```

4.3.3 Навчання моделі

Навчання моделі також проводилося в кілька кроків.

Для навчання генератора та дискримінатора нам знадобилися два тензори: один із реальними маркуваннями, а інший – з підробленими. Щоб покращити якість навчання, ми використовували згладжені маркування.

Далі ми створили цикл, який буде виконуватися певну кількість разів, яка дорівнювала числу епох.

Визначили кількість пакетів даних та написали цикл, який буде виконаний для кожного пакета.

Для поточної ітерації був обраний міні-пакет даних. Був створений вектор шуму, обрали пакет зображень та вкладень, а потім нормалізували зображення.

Далі ми застосовували модель генератора, використовуючи вектор шуму та вкладення, що передаються їй як вхідні дані. Отримані результати будуть згенеровані зображення. Тобто треба було згенерувати пакет підроблених зображень, обумовлених пакетом вкладень та пакетом векторів шуму.

Наступним кроком застосували модель компресора до вкладення його стиснення. Потім перетворили його на тензор шляхом повторення просторових розмірностей, щоб отримати тензор з формою (batch_size, 4, 4, 128).

Далі виконали навчання моделі дискримінатора на кількох типах зображень, включаючи підроблені, створені генератором, справжні зображення з навчальних даних та помилкові зображення.

Навчили змагальну модель, подавши на її вхід три тензори, включаючи згенеровані зображення, реальні зображення з набору даних та згенеровані вкладення, які будуть стиснуті моделлю компресора. В результаті були обчисленні градієнти та ваги моделі будуть оновлені для

поточного пакета даних. Цей процес повторювався для кожного пакету даних у циклі навчання.

Після цього були обчислені втрати, які зберігали для оцінки. Корисно було зберігати друк різних втрат, які супроводжують процес навчання, щоб оцінити ефективність моделі.

Після кожної епохи ми зберігали всі втрати в TensorBoard на сервері для подальшої оцінки. Це дозволяє нам стежити за процесом навчання та оцінювати ефективність моделі.

Після закінчення кожної епохи ми створювали зображення для оцінки прогресу та зберігали їх у зазначеній директорії. Крім того, потрібно було визначити функцію корисності.

Треба було зберігати ваги кожної моделі на стадії I мережі StackGAN, щоб використовувати їх у майбутньому.

Програмний код для навчання моделі, поданий у лістингу 4.2.

Лістинг 4.2 – Програмний код навчання моделі

```
real_labels = np.ones((batch_size, 1), dtype=float) * 0.9
fake_labels = np.zeros((batch_size, 1), dtype=float) * 0.1
for epoch in range(epochs):
print("==========================================")
print("Epoch is:", epoch)
print("Number of batches", int(X_train.shape[0] / batch_size))
gen_losses = []
dis_losses = []
number_of_batches = int(X_train.shape[0] / batch_size)
for index in range(number_of_batches):
print("Batch:{}".format(index+1))
# Create a batch of noise vectors.
z_noise = np.random.normal(0, 1, size=(batch_size, z_dim))
image_batch = X_train[index * batch_size:(index + 1) *
batch_size]
embedding_batch = embeddings_train[index *
batch_size:(index + 1) * batch_size]
# Image normalization.
image_batch = (image_batch - 127.5) / 127.5
fake_images, _ = stage1_gen.predict([embedding_batch, z_noise],
verbose=3)
compressed_embedding =
embedding_compressor_model.predict_on_batch(embedding_batch)
```

Продовження лістингу 4.2

```
      compressed_embedding = np.reshape(compressed_embedding,
      (-1, 1, 1, condition_dim))
      compressed_embedding = np.tile(compressed_embedding, (1, 4, 4,
1))
      dis_loss_real = stage1_dis.train_on_batch([image_batch,
      compressed_embedding],
      np.reshape(real_labels, (batch_size, 1)))
      dis_loss_fake = stage1_dis.train_on_batch([fake_images,
      compressed_embedding],
      np.reshape(fake_labels, (batch_size, 1)))
      dis_loss_wrong =
      stage1_dis.train_on_batch([image_batch[:(batch_size – 1)],
      compressed_embedding[1:]], np.reshape(fake_labels[1:],
(batch_size-1, 1)))
      g_loss = adversarial_model.train_on_batch([embedding_batch,
      z_noise, compressed_embedding],[K.ones((batch_size, 1)) * 0.9,
      K.ones((batch_size, 256)) * 0.9])
      d_loss = 0.5 * np.add(dis_loss_real, 0.5 *
np.add(dis_loss_wrong, dis_loss_fake))
      print("d_loss:{}".format(d_loss))
      print("g_loss:{}".format(g_loss))
      dis_losses.append(d_loss)
      gen_losses.append(g_loss)
      write_log(tensorboard, 'discriminator_loss',
np.mean(dis_losses), epoch)
      write_log(tensorboard, 'generator_loss',
np.mean(gen_losses[0]), epoch)
      z_noise2 = np.random.normal(0, 1, size=(batch_size, z_dim))
      embedding_batch = embeddings_test[0:batch_size]
      fake_images, _ = stage1_gen.predict_on_batch([embedding_batch,
z_noise2])
      # Save images.
      for i, img in enumerate(fake_images[:10]):
      save_rgb_img(img, "results/gen_{}_{}.png". format(epoch, i))
      def save_rgb_img(img, path):
      #Saving rgb_img
      fig = plt.figure()
      ax = fig.add_subplot(1, 1, 1)
      ax.imshow(img)
      ax.axis("off")
      ax.set_title("Image")
      plt.savefig(path)
      plt.close()
      stage1_gen.save_weights("stage1_gen.h5")
      stage1_dis.save_weights("stage1_dis.h5")
```

В результаті було успішно завершено навчання моделі першого етапу StackGAN. Реалізований навчений генератор, який здатний генерувати зображення розміром 64x64x3 з базовими кольорами та простими формами.

4.4 Навчання моделі StackGAN етапу II

4.4.1 Завантаження набору даних

Були використані ті ж самі методи завантаження даних, що були визначені в розділі для етапу I створення StackGAN. Ми завантажили набори даних з високою та низькою роздільною здатністю окремо, а також – окремі набори даних для навчання та тестування.

Програмний код для завантаження набору даних на II етапі, поданий у лістингу 4.3.

Лістинг 4.3 – Програмний код завантаження набору даних

```
X_hr_train, y_hr_train, embeddings_train =
load_dataset(filenames_file_path=filenames_file_path_train,
class_info_file_path=class_info_file_path_train,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_train,
image_size=(256, 256))
X_hr_test, y_hr_test, embeddings_test =
load_dataset(filenames_file_path=filenames_file_path_test,
class_info_file_path=class_info_file_path_test,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_test,
image_size=(256, 256))
X_lr_train, y_lr_train, _ =
load_dataset(filenames_file_path=filenames_file_path_train,
class_info_file_path=class_info_file_path_train,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_train,
image_size=(64, 64))
X_lr_test, y_lr_test, _ =
load_dataset(filenames_file_path=filenames_file_path_test,
class_info_file_path=class_info_file_path_test,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_test
```

4.4.2 Створення моделей

На етапі II ми створюємо Keras-модель аналогічно тому, як це було зроблено на етапі I. Однак тут є відмінності – ми використовуємо оптимізатор Adam з коефіцієнтом навчання 0,0002 та значеннями beta_1 та beta_2 рівними відповідно 0,5 та 0,999.

Програмний код для створення моделей на II етапі, можна побачити у лістингу 4.4.

Лістинг 4.4 – Програмний код створення моделей

```
    dis_optimizer = Adam(lr=stage1_discriminator_lr, beta_1=0.5,
beta_2=0.999)
    gen_optimizer = Adam(lr=stage1_generator_lr, beta_1=0.5,
beta_2=0.999)
    stage2_dis = build_stage2_discriminator()
    stage2_dis.compile(loss='binary_crossentropy',
optimizer=dis_optimizer)
    stage1_gen = build_stage1_generator()
    stage1_gen.compile(loss="binary_crossentropy",
optimizer=gen_optimizer)
    stage1_gen.load_weights("stage1_gen.h5")
    stage2_gen = build_stage2_generator()
    stage2_gen.compile(loss="binary_crossentropy",
optimizer=gen_optimizer)
    embedding_compressor_model = build_embedding_compressor_model()
    embedding_compressor_model.compile(loss='binary_crossentropy',
optimizer='adam')
    adversarial_model = build_adversarial_model(stage2_gen,
stage2_dis, stage1_gen)
    adversarial_model.compile(loss=['binary_crossentropy',
KL_loss], loss_weights=[1.0, 2.0],
    optimizer=gen_optimizer, metrics=None)
```

Нарешті, коли були підготовлені дані та моделі для другого етапу StackGAN, можливо було розпочати навчання.

4.4.3 Навчання моделі

Навчання моделі на етапі II відбувалося так же послідовно, поетапно як і на етапі I, дотримуючись певних кроків та процедур:

Спочатку треба було налаштувати сервер TensorBoard, щоб зберігати інформацію про втрати в процесі навчання.

Для навчання генератора та дискримінатора було створено два тензори з реальними та згенерованими мітками. Ми використовували згладжені позначки, які допоможуть покращити стабільність навчання.

Далі створили цикл, як і на першому етапі, який буде виконуватися певну кількість разів, яка дорівнювала числу епох.

Також був створений вкладений цикл усередині циклу епох, який ітерується за кількістю пакетів.

Наступним кроком обрали необхідні для навчання дані.

Потім процес продовжується, використовуючи генеративну мережу для генерації фальшивих зображень розміром 256x256x2. Спочатку ми використовуємо генеративну мережу, навчену на етапі I, щоб згенерувати зображення з низькою роздільною здатністю. Потім ці зображення передаються генеративної мережі, навченої на етапі II, щоб створити зображення з високою роздільною здатністю, враховуючи низькодозволені зображення як умову.

Для стиснення вкладення ми застосовуємо модель компресора, яка перетворює вкладення у тензор з формою (batch_size, 4, 4, 128). Для цього ми повторюємо елементи вкладення у просторі, щоб отримати тензор потрібної форми.

Наступним кроком розпочинається навчання моделі дискримінатора. Для цього використовуються підроблені та реальні зображення, а також зображення, отримані шляхом генерації зображень за допомогою генераторної мережі, а потім піддані стиску за допомогою

компресорної моделі. Ми навчатимемо дискримінатор на тому, як відрізнити підроблені та реальні зображення один від одного, а також на тому, як відрізнити підроблені зображення від зображень, отриманих за допомогою генераторної та компресорної мереж, і навпаки.

Далі ми переходимо до навчання змагальної моделі, яка є комбінацією моделі генератора та моделі дискримінатора. На вхід цієї моделі ми подаємо три параметри: підроблені зображення, реальні зображення та маркування, що відповідають цим зображенням. Крім того, ми подаємо на вхід відповідні реальні значення, необхідні для навчання моделі.

Після цього розрахували значення функцій втрат для кожної з моделей – генератора, дискримінатора та змагальної моделі, та запишемо їх для подальшої оцінки якості навчання. Після кожної епохи зберігаємо втрати на сервері TensorBoard.

Після закінчення кожної епохи ми проводимо оцінку процесу навчання, генеруємо зображення за допомогою навченої моделі та зберігаємо їх у директорію для результатів.

Після завершення навчання ми зберігаємо моделі або зберігаємо значення їх ваг. Це необхідно для того, щоб ми могли використовувати їх пізніше, наприклад, для подальшого навчання моделей або для створення зображень в іншому середовищі.

Програмний код для навчання моделей на ІІ етапі, можна побачити у лістингу 4.5.

Лістинг 4.5 – Програмний код навчання моделей

```
real_labels = np.ones((batch_size, 1), dtype=float) * 0.9
fake_labels = np.zeros((batch_size, 1), dtype=float) * 0.1
real_labels = np.ones((batch_size, 1), dtype=float) * 0.9
fake_labels = np.zeros((batch_size, 1), dtype=float) * 0.1
for epoch in range(epochs):
print("Epoch is:", epoch)
gen_losses = []
```

Продовження лістингу 4.5

```
dis_losses = []
print("Number of batches:{}".format(number_of_batches))
for index in range(number_of_batches):
print("Batch:{}".format(index))
# Create mini-batch noise vectors.
z_noise = np.random.normal(0, 1, size=(batch_size, z_dim))
X_hr_train_batch = X_hr_train[index * batch_size:(index + 1) *
batch_size]
embedding_batch = embeddings_train[index *
batch_size:(index + 1) * batch_size]
# Image normalization.
X_hr_train_batch = (X_hr_train_batch - 127.5) / 127.5
lr_fake_images, _ = stage1_gen.predict([embedding_batch,
z_noise], verbose=3)
g_loss = adversarial_model.train_on_batch([embedding_batch,
z_noise, compressed_embedding],[K.ones((batch_size, 1)) * 0.9,
K.ones((batch_size, 256)) * 0.9])
d_loss = 0.5 * np.add(dis_loss_real, 0.5 *
np.add(dis_loss_wrong, dis_loss_fake))
print("d_loss:{}".format(d_loss))
print("g_loss:{}".format(g_loss))
write_log(tensorboard, 'discriminator_loss',
np.mean(dis_losses), epoch)
write_log(tensorboard, 'generator_loss',
np.mean(gen_losses)[0], epoch)
# Generate and save image after every second epoch
if epoch % 2 == 0:
# z_noise2 = np.random.uniform(-1, 1, size=(batch_size, z_dim))
z_noise2 = np.random.normal(0, 1, size=(batch_size, z_dim))
embedding_batch = embeddings_test[0:batch_size]
lr_fake_images, _ = stage1_gen.predict([embedding_batch,
z_noise2], verbose=3)
hr_fake_images, _ = stage2_gen.predict([embedding_batch,
lr_fake_images], verbose=3)
# Saving the image
for i, img in enumerate(hr_fake_images[:10]):
save_rgb_img(img, "results2/gen_{}_{}.png". format(epoch, i))
# Saving models.
stage2_gen.save_weights("stage2_gen.h5")
stage2_dis.save_weights("stage2_dis.h5")
```

У результаті успішно завершилося навчання StackGAN другого етапу, що дозволило створити генератор, здатний генерувати реалістичні зображення розміром 256×256×3. Для цього ми використовували текстове вкладення та шум як вхідні дані для генератора, який потім створив

зображення з бажаною роздільною здатністю. У наступних розділах розглядаються графіки втрат мереж та результати, згенеровані генератором.

4.5 Результати візуалізації згенерованих зображень

Для запуску проекту, необхідно виконати команду «python3 run.py» в консолі системи, щоб відкрити текстове поле «Enter a text description of the bird:», де можна ввести опис птаха, який ми хочемо згенерувати (рисунок 4.1). Після введення опису птиці система запускає алгоритм генерації зображення птиці на основі текстового опису. Наприклад, для експерименту можна вибрати опис, такий як «Small bird with gray-blue body, black head and white beak».

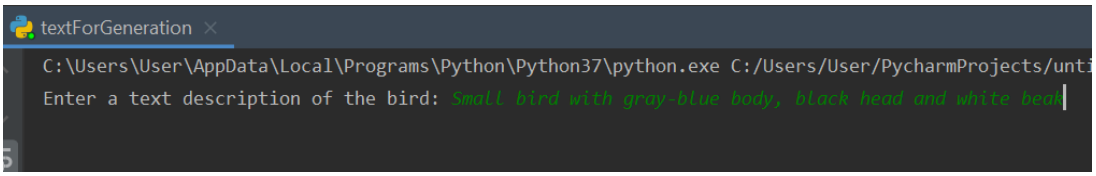

Рисунок 4.1 – Текстове поле для введення опису птиці

Коли ми вводимо опис птаха у текстове поле консолі, перша фотографія розміром 64x64x3, яка з'являється, була згенерована за допомогою першого етапу навчання StackGAN. На цій фотографії відображаються початкові кольори, висота та ширина птаха, але обриси птаха виглядають надто розмитими та невизначеними. Початкове згенероване зображення формату 64x64x3 наведено на рисунку 4.2.

Рисунок 4.2 – Згенероване зображення формату 64x64x3 на I етапі

Друге зображення має формат 256x256x3 та є остаточною фотографією після проходження другого етапу навчання StackGAN. У цьому етапі, GAN Stage-II використовував результати та текстові описи Stage-I як вхідні дані і згенерував зображення з високою роздільною здатністю та фотореалістичними деталями. Цей етап дозволив виправити дефекти в результатах етапу I та додати переконливі деталі у процесі уточнення зображення. Таким чином, друга фотографія представляє більш чітке, детальне та реалістичне зображення птаха. Згенероване зображення формату 256x256x3 зображено на рисунку 4.3.

Рисунок 4.3 – Згенероване зображення формату 256x256x3 на II етапі

Після проходження 500 епох навчання мережа досягне досить високого рівня продуктивності в генерації зображень, і зможе створювати цілком задовільні результати.

Приклади дослідження порівняння інших згенерованих зображень для різних текстових описів на I етапі та II етапі зображено на рисунку 4.4.

Рисунок 4.4 – Порівняння результатів на Stage-I та Stage-II

Для більш реалістичних зображень необхідно було продовжити навчання на 1000 епох. Якщо виконати все правильно, то після 1000 епох мережа генератора стане здатною генерувати зображення більш високої роздільної здатності, які будуть якісними та реалістичними. Приклад дослідження проходження навчання мережі після 1000 епох зображено на рисунку 4.5.

Рисунок 4.5 – Реалістичні згенеровані зображення після проходження навчання в 1000 епох

4.6 Результати візуалізації втрат

Щоб візуалізувати функції втрат, необхідно було запустити сервер TensorBoard за допомогою команди «tensorboard--logdir=logs». Потім необхідно було відкрити у браузері localhost:6006, після чого у вікні SCALARS на сервері TensorBoard відображаються графіки обох функцій втрат, що мають такий вигляд, як зображено на рисунках 4.6 – 4.7.

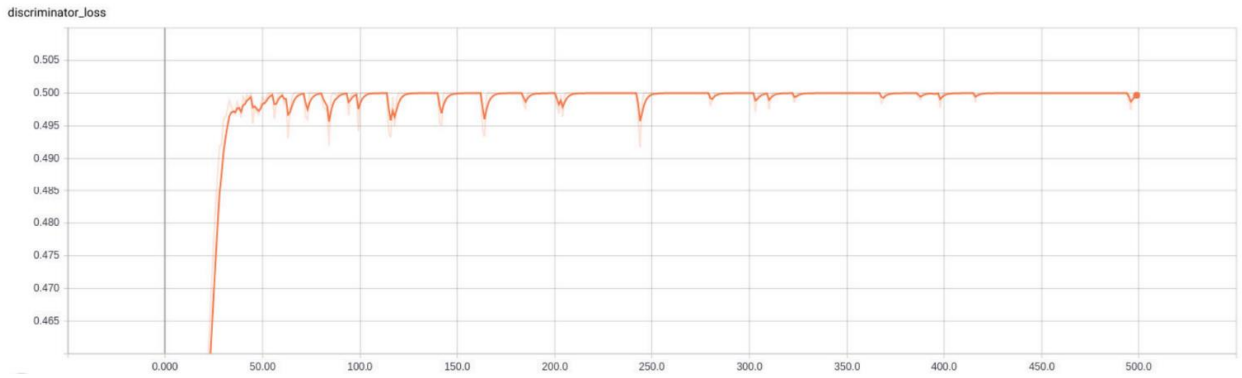

Рисунок 4.6 – Графік втрат мережі етапу I

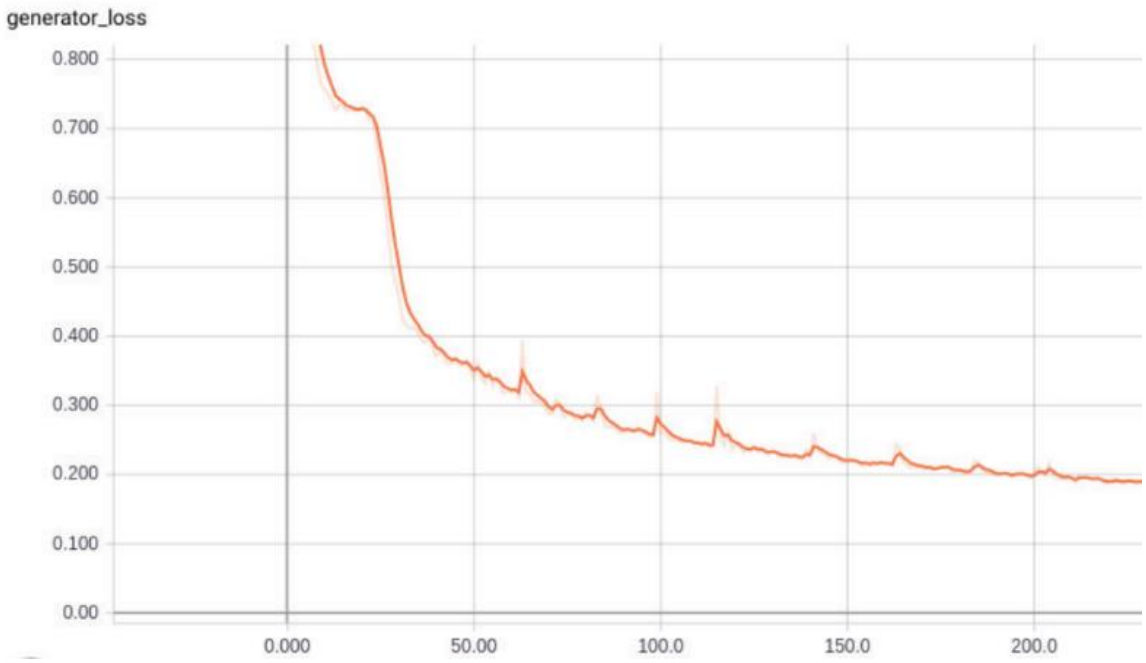

Рисунок 4.7 – Графік втрат мережі етапу II

За допомогою даних графіків можна було приймати рішення про продовження або зупинення навчання. Якщо значення втрат не зменшувалися, можна було припинити навчання, оскільки ймовірність поліпшення подальших результатів була невисока. Якщо значення втрат продовжували зростати, також слід зупинити навчання. Для досягнення найкращих результатів проводили експерименти з гіперпараметрами.

Проведення дослідження моделі StackGAN дозволило отримати глибоке розуміння того, як цей метод працює для синтезу зображень з текстових описів.

Однією з основних проблем, з якою ми зіткнулися, було навчання моделі на досить великому наборі даних. Для досягнення високої реалістичності зображень необхідно провести навчання на великому наборі зображень і текстових описів, що вимагає потужних обчислювальних ресурсів.

Крім того, ми зіткнулися з проблемою оптимізації гіперпараметрів моделі, таких як швидкість навчання та кількість епох. Підбір цих властивостей може суттєво впливати на результат навчання.

Загалом згенеровані зображення були досить реалістичними, особливо після навчання на великій кількості епох. Ми також виявили, що другий етап StackGAN, який використовує зображення з низькою роздільною здатністю, створені на першому етапі, дозволяє створювати більш деталізовані та реалістичні зображення.

Таким чином, дослідження моделі StackGAN показало, що це потужний метод для синтезу зображень із текстових описів, але потребує серйозних зусиль для досягнення високої реалістичності та точності згенерованих зображень.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи було дослідження роботи генеративно-змагальних моделей в задачах синтезу тексту в зображення.

Дане дослідження було спрямоване на вивчення унікальних характеристик генеративного моделювання з виявлення більш ефективних моделей на навчання генерації реалістичних зображень з урахуванням текстового опису. Воно також включає перевірку якості та швидкості процесу генерації шляхом реалізації алгоритму для наочного відображення отриманих результатів.

Щоб досягти мети кваліфікаційної роботи, успішно було виконано такі завдання:

– провести аналіз предметної області;

– здійснити теоретичні дослідження роботи мережі GAN;

– проаналізувати існуючі моделі GAN для синтезу тексту зображення;

– вибрати кілька моделей та вивчити принцип роботи алгоритмів;

– проаналізувати переваги та недоліки кожної моделі та обрати найкращу, для досягнення мети.

– вибрати датасети для проведення дослідження;

– реалізувати обрану модель у Keras та TensorFlow;

– навчити вибрані мережі;

– проаналізувати та порівняти результати, отримані після роботи.

Тема автоматичного створення високоякісних зображень на основі текстових описів є захоплюючим та корисним напрямком з безліччю практичних застосувань. Одним з основних інструментів вирішення задачі в цьому напрямку є моделі GAN, які дозволяють створювати умовні зображення, що мають високий рівень візуального реалізму, різноманітності та семантичної точності.

В рамках проведеного дослідження було глибоко проаналізовано та порівняно декілька моделей GAN, таких як Obj-GAN, DALL-E2, MSG-GAN, StackGAN, FusedGAN, MirrorGAN та AttnGAN. Для кожної моделі було визначено якісні переваги та недоліки при роботі з ними.

В рамках поставленої мети було запропоновано модель StackGAN, яка призначена для створення фотореалістичних зображень розміром 256x256 на основі текстових описів. Ця модель здатна генерувати зображення з високою роздільною здатністю, але було виявлено, що вона має деякі обмеження – деякі з генерованих зображень часто виявляються нерозбірливими, і може виникнути колапс режиму. У зв'язку з цим, в рамках даного дослідження, ми прагнули вирішити ці дві проблеми, щоб покращити якість зображень, що генеруються.

Для вирішення цих проблем у дослідженні були застосовані різні методи, такі як навчання генератора на декількох масштабах та використання додаткових втрат для більш точної відповідності зображень описів.

Було проаналізовано архітектуру мережі Stack-GAN та визначено втрати, що використовувались під час її навчання. Наступним кроком було створення реалізації мережі StackGAN у структурі Keras, яка була успішно навчена. Ми провели оцінку якості отриманої моделі та зберегли її для використання у майбутньому.

В цілому, можна сказати, що модель StackGAN є потужним інструментом для синтезу зображень на основі текстових описів, і має потенціал для використання в багатьох практичних додатках, таких як генерація зображень для медичних цілей, створення віртуальних світів і т. д.

Результати дослідження відображені у тезах доповіді [26].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Державний стандарт України ДСТУ 3008-2015: Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: Держстандарт України, 2015. 31с.

2. Методичні вказівки до організації й захисту кваліфікаційної роботи на здобуття першого (бакалаврського) рівня вищої освіти для студентів усіх форм навчання спеціальності 122 Комп'ютерні науки за освітньою програмою «Комп'ютерні науки» [Електронне видання]. Харків: ХНУРЕ, 2019. 44с.

3. Text-to-Image Generation. URL: https://paperswithcode.com/task/text-to-image-generation (дата звернення: 10.04.2023).

4. Reed S., Logeswaran L., Schiele B., and Lee H. Generative adversarial text-to-image synthesis. URL: https://proceedings.mlr.press/v48/reed16.html (дата звернення 10.04.2023).

5. Генерація зображення за текстом. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BF%D0%BE_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%83 (дата звернення: 11.04.2023).

6. GAN Deep Learning: Practical Guide. URL: https://datagen.tech/guides/computer-vision/gan-deep-learning/(дата звернення: 12.04.2023).

7. A Gentle Introduction to Generative Adversarial Networks (GANs). URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (дата звернення: 12.04.2023).

8. Inside the Generative Adversarial Networks (GAN) architecture. URL: https://medium.com/@Packt_Pub/inside-the-generative-adversarial-networks-gan-architecture-2435afbd6b3b (дата звернення: 13.04.2023).

9. Impressive Applications of Generative Adversarial Networks (GANs). URL: https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/ (дата звернення: 14.04.2023).

10. Text-to-Image Generation Using Deep Learning. URL: https://www.mdpi.com/2673-4591/20/1/16 (дата звернення: 14.04.2023).

11. Common challenges faced while working with GAN models. URL: https://hub.packtpub.com/challenges-training-gans-generative-adversarial-networks/ (дата звернення: 15.04.2023).

12. Top 3 text-to-image generators: How DALL-E 2, GLIDE and Imagen stand out. URL: https://venturebeat.com/ai/top-3-text-to-image-generators-how-dall-e-2-glide-and-imagen-stand-out (дата звернення: 15.04.2023).

13. Generating images from text with AttnGAN. URL: https://habr.com/ru/articles/409747/ (дата звернення: 16.04.2023).

14. MirrorGAN: Learning Text-to-image Generation by Redescription. URL: https://arxiv.org/abs/1903.05854 (дата звернення: 16.04.2023).

15. Navaneeth B., Gang H. Semi-supervised FusedGAN for ConditionalImage Generation, 2018, 156. (дата звернення: 17.04.2023).

16. Kailash A. Generative Adversarial. Networks Projects. 2020. P. 138–139. (дата звернення: 18.04.2023).

17. Генерація зображення за текстом. URL: http://neerc.ifmo.ru/wiki/index.php?title=%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F_%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BF%D0%BE_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%83#AttnGAN (дата звернення: 19.04.2023).

18. Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks. URL: https://ieeexplore.ieee.org/document/8237891 (дата звернення: 20.04.2023).

19. Multi-Channel Attention Selection GAN with Cascaded Semantic Guidancefor Cross-View Image Translation. URL: https://arxiv.org/pdf/1904.06807.pdf (дата звернення: 20.04.2023).

20. Top 3 text-to-image generators: How DALL-E 2, GLIDE and Imagen stand out. URL: https://venturebeat.com/ai/top-3-text-to-image-generators-how-dall-e-2-glide-and-imagen-stand-out (дата звернення: 22.04.2023).

21. Adversarial text-to-image synthesis: A review ObjGAN. URL: https://www.sciencedirect.com/science/article/pii/S0893608021002823 (дата звернення: 23.04.2023).

22. Zhang, H.; Xu, T.; Li, H.; Zhang, S.; Wang, X.; Huang, X.; Metaxas, D.N. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. IEEE Trans. Pattern Anal. Mach. Intell. 2018, 41, 1947-1962. (дата звернення: 25.04.2023).

23. A Simple and Intuitive Explanation of StackGAN. URL: https://susant.medium.com/a-simple-explanation-of-stackgan-af1ebf310d0f (дата звернення: 30.04.2023).

24. A. Kailash, Generative Adversarial Networks. Packt Publishing, 2019, 141-151. (дата звернення: 02.05.2023).

25. CUB-200-2011 (Caltech-UCSD Birds-200-2011). URL: https://paperswithcode.com/dataset/cub-200-2011 (дата звернення: 03.05.2023).

26. Рябова Н.В., Воронюк К.Л. Дослідження генеративно-змагальних моделей для синтезу тексту в зображення. Матеріали 27-го Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті». Україна, м. Харків, 10 – 12 травня 2023р.

ДОДАТОК А

Результати показників при навчанні моделі StackGAN

```
embeddings:  (8855, 10, 1024)
Embeddings shape: (8855, 10, 1024)
embeddings:  (2933, 10, 1024)
Embeddings shape: (2933, 10, 1024)
=======================================
Epoch is: 0
Number of batches 138
Batch:1
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packag
Instructions for updating:
Use tf.cast instead.
/usr/local/lib/python3.6/dist-packages/keras/engine/training
  'Discrepancy between trainable weights and collected trair
d_loss_real:0.9976924061775208
d_loss_fake:2.0417661666870117
d_loss_wrong:5.708157062530518
d_loss:2.4363270103931427
g_loss:[1.9268906, 1.8937846, 0.016552962]
Batch:2
d_loss_real:0.7542287707328796
d_loss_fake:1.0874087810516357
d_loss_wrong:0.4822918176651001
d_loss:0.7695395350456238
g_loss:[4.2255583, 4.1922197, 0.016669236]
Batch:3
d_loss_real:3.1299123764038086
```

Рисунок А.1 – Результати навчання моделі StackGAN етапу I

```
Instructions for updating:
Colocations handled automatically by placer.
embeddings:  (8855, 10, 1024)
All embeddings shape: (8855, 10, 1024)
embeddings:  (2933, 10, 1024)
All embeddings shape: (2933, 10, 1024)
embeddings:  (8855, 10, 1024)
All embeddings shape: (8855, 10, 1024)
embeddings:  (2933, 10, 1024)
All embeddings shape: (2933, 10, 1024)
=======================================
Epoch is: 0
Number of batches:2213
Batch:1
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/pythc
Instructions for updating:
Use tf.cast instead.
/usr/local/lib/python3.6/dist-packages/keras/engine/training.py:490: UserWarnir
  'Discrepancy between trainable weights and collected trainable'
d_loss:4.560303330421448
g_loss:[0.6102203, 0.567336, 0.021442147]
Batch:2
d_loss:2.0408308804035187
g_loss:[2.0373921, 2.0077133, 0.014839375]
Batch:3
d_loss:4.207903504371643
g_loss:[2.0374453, 2.0148764, 0.011284475]
Batch:4
d_loss:3.5446853637695312
```

Рисунок А.2 – Результати навчання моделі StackGAN етапу II

ДОДАТОК Б

Лістинг ініціалізації гіперпараметрів

Лістинг Б.1 – Програмний код ініціалізації гіперпараметрів

```
    if __name__ == '__main__':
        data_dir = "/content/birds/"
        train_dir = data_dir + "/train"
        test_dir = data_dir + "/test"
        image_size = 64
        batch_size = 64
        z_dim = 100
        stage1_generator_lr = 0.0002
        stage1_discriminator_lr = 0.0002
        stage1_lr_decay_step = 600
        epochs = 1000
        condition_dim = 128
        embeddings_file_path_train = train_dir + "/char-CNN-RNN-
embeddings.pickle"
        embeddings_file_path_test = test_dir + "/char-CNN-RNN-
embeddings.pickle"
        filenames_file_path_train = train_dir + "/filenames.pickle"
        filenames_file_path_test = test_dir + "/filenames.pickle"
        class_info_file_path_train = train_dir +
"/class_info.pickle"
        class_info_file_path_test = test_dir + "/class_info.pickle"
        cub_dataset_dir = "/content/CUB_200_2011"
    # Define optimizers
        dis_optimizer = Adam(lr=stage1_discriminator_lr,
beta_1=0.5, beta_2=0.999)
        gen_optimizer = Adam(lr=stage1_generator_lr, beta_1=0.5,
beta_2=0.999)

        """
        Load datasets
        """
        X_train, y_train, embeddings_train =
load_dataset(filenames_file_path=filenames_file_path_train,
class_info_file_path=class_info_file_path_train,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_train,
image_size=(64, 64))
        X_test, y_test, embeddings_test =
load_dataset(filenames_file_path=filenames_file_path_test,
class_info_file_path=class_info_file_path_test,
cub_dataset_dir=cub_dataset_dir,
embeddings_file_path=embeddings_file_path_test,
image_size=(64, 64))
```

ДОДАТОК В

Приклад порівняння якості моделей при синтезі тексту в зображення

Рисунок В.1 – Результат порівняння згенеровних зображень різних моделей

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	81 с.
	Інші документи	
2.	Презентаційні матеріали	20 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Воронюк К.Л			Дослідження генеративно-змагальних моделей в задачах синтезу тексту в зображення
Перевірив		Рябова Н.В.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-21-2	122
ХНУРЕ кафедра ШІ	