МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА СТИСНЕННЯ ДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 «Комп'ютерні науки»

122 – КРМ – 601.21810207

Виконав студент 6-го курсу, групи 601
_____ Г. Г. Горбатко
«19» лютого 2024 р.

Керівник: д-р техн. наук, професор
_____ О. П. Гожий
«19» лютого 2024 р.

Миколаїв – 2024

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20__р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Горбатко Геннадій Геннадійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Інтелектуальна система стиснення даних на основі нейронних мереж».

Керівник роботи Гожий Олександр Петрович, доктор технічних наук, професор

Затв. наказом Ректора ЧНУ ім. Петра Могили від «19» жовтня 2023 р. № 201

2. Строк подання студентом роботи 19 лютого 2024 р.

3. Вхідні (початкові) дані до роботи: набір зображень які підлягають стисненню.

Очікуваний результат роботи: повнофункціональна система стиснення даних на базі нейронних мереж, яка дозволяє ефективно стискати різні типи вхідних даних .

4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

- аналіз актуальності обраної теми;
- розгляд та вивчення загальної теорії;
- аналіз методів та алгоритмів стиснення даних.
- програмна реалізація інтелектуальної системи стиснення даних.

5. Перелік графічного матеріалу: презентація

6. Завдання до спеціальної частини: Забезпечення електробезпеки у робочій кімнаті у житловому будинку.

_____

_____

_____

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	Григор'єва Л. І., д-р біол. наук, професор	
Методична частина	Гожий О. П., д-р техн. наук, професор.	

Керівник роботи _ д-р техн. наук, проф. Гожий О. П._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Горбатко Г. Г._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__31__» _____жовтня____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи магістра

Тема: Інтелектуальна система стиснення даних на основі нейронних мереж

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРМ	01.10.2023	10.10.2023	Виконано
2	Отримання завдання на виконання КРМ	25.10.2023	01.11.2023	Виконано
3	Складання календарного плану роботи на весь період виконання КРМ	06.11.2023	10.11.2023	Виконано
4	Отримання завдання на переддипломну практику	13.11.2023	26.11.2023	Виконано
5	Проходження передатестаційної практики, збір та аналіз матеріалів до КРМ	27.11.2023	23.12.2023	Виконано
6	Розробка звіту з передатестаційної практики	24.12.2023	26.12.2023	Виконано
7	Виконання КРМ: аналіз сучасного стану задачі огляд існуючих методів, алгоритмів, та технологій, розробка методів стиснення даних	04.01.2024	08.01.2024	Виконано
8	Перший попередній захист КРМ на засіданні комісії кафедри	26.01.2024	02.02.2024	Виконано
9	Розробка спеціальної частини з охорони праці та методичної частини	29.01.2024	29.01.2024	Виконано
10	Корегування роботи за результатами попереднього захисту	30.01.2024	05.02.2024	Виконано
11	Доробка та остаточне оформлення КРМ	06.02.2024	11.02.2024	Виконано
12	Другий попередній захист КРМ на засіданні комісії кафедри	12.02.2024	12.02.2024	Виконано
13	Подання КРМ, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.02.2024	20.02.2024	Виконано
14	Захист КРМ перед екзаменаційною комісією (ЕК)	26.02.2024	27.02.2024	Виконано

Розробив студент Горбатко Г . Г. _____ _____

(прізвище та ініціали) (*підпис*)

Керівник роботи д-р техн. наук, проф. Гожий О. П._____ _____

(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«09» листопада 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
студента групи 601 ЧНУ ім. Петра Могили

Горбатко Геннадія Геннадійовича

на тему: “ **ІНТЕЛЕКТУАЛЬНА СИСТЕМА СТИСНЕННЯ ДАНИХ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ** ”

Дана кваліфікаційна робота присвячена розробці та налагодженню інтелектуальної системи стиснення даних, що ґрунтується на використанні нейронних мереж. Дослідження спрямоване на вдосконалення процесу стиснення та зменшення обсягу інформації без втрати суттєвої якості. Було використано підходи нейронних мереж для автоматизованого визначення оптимальних параметрів стиснення.

Об’єктом кваліфікаційної роботи є процес стиснення даних за допомогою нейронних мереж.

Предметом кваліфікаційної роботи є методи ефективного стиснення даних за допомогою нейронних мереж.

Метою кваліфікаційної роботи є дослідження методів ефективного стиснення даних з використанням нейронних мереж.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, трьох розділів, висновків та додатків.

У роботі розглядаються теоретичні аспекти функціонування нейронних мереж, та алгоритмів у контексті стиснення даних, а також проводиться практичне застосування розробленої системи на реальних наборах даних.

Кваліфікаційна робота містить 105 сторінок, 73 рисунків, 2 таблиці та 1 додаток. В роботі використано 64 джерела.

Ключові слова: CNN, tensorflow, нейроні мережі, JPEG, стиснення даних, pandas, NumPy

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Horbatko Hennadii

“ AN INTELLIGENT DATA COMPRESSION SYSTEM BASED ON NEURAL NETWORKS ”

This qualification work is devoted to the development and adjustment of an intelligent data compression system based on the use of neural networks. The research is aimed at improving the compression process and reducing the amount of information without losing significant quality. Neural network approaches were used for automated determination of optimal embossing parameters.

An object of the qualification work is the process of data compression using neural networks.

A subject of the qualification work is methods of effective data compression using neural networks.

A purpose of qualification work is the study of methods of effective data compression using neural networks.

The work consists of a professional section and a special part on labor protection. The explanatory note consists of an introduction, three sections, conclusions and appendices. The work considers the theoretical aspects of the functioning of neural networks and algorithms in the context of data compression, and also provides for the practical application of the developed system on real data sets.

The qualification paper contains 105 pages, 73 figures, 2 tables and 1 appendix. The work uses 64 sources.

Key words: CNN, tensorflow, neural networks, JPEG, data compression, pandas, NumPy

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БМП – Багато Шаровий Перцепторн

БНМ – Багатошарова нейрона мережа

ЗНМ – Згорткова нейронна мережа

НМ – Нейронна мережа

ШНМ – Штучна нейрона мережа

CNN – Convolutional neural network

JPEG – Joint Photographic Experts Group

JPEG-LS – Joint Photographic Experts Group - lossless

NN – Neural Network

RNN – Recurrent neural network

ВСТУП

У сучасному світі обсяги генерованих даних постійно зростають. За даними дослідницької компанії IDC, до 2025 року обсяг генерованих даних у світі досягне 175 зетабайти. Це означає, що щорічно генерується 10,5 трильйонів гігабайтів даних.

Це зростання пов'язано з розвитком цифрових технологій, таких як штучний інтелект, машинне навчання, інтернет речей, віртуальна та доповнена реальність. Ці технології генерують величезні обсяги даних, які потрібно зберігати та обробляти.

При стисканні даних важливо зберегти їхню якість. Якщо дані будуть стиснутими надмірно, вони можуть бути пошкоджені або втрачені. Це може бути неприйнятно для деяких застосувань, таких як зберігання медичних зображень або відео.

Традиційні методи стиснення даних часто призводять до втрати якості даних. Це може бути пов'язано з тим, що вони використовують прості алгоритми, які не враховують всі закономірності даних.

Нейронні мережі є потужним інструментом для вирішення різних завдань, зокрема, стиснення даних. Розвиток нейронних мереж призвів до створення нових, більш ефективних алгоритмів стиснення даних.

Нейронні мережі можуть навчатися на великих наборах даних і виявляти закономірності, які неможливо виявити за допомогою традиційних методів. Це дозволяє їм досягати більш високого ступеня стиснення без втрати якості даних.

Об’єктом кваліфікаційної роботи є процес стиснення даних за допомогою нейронних мереж.

Предметом кваліфікаційної роботи є методи ефективного стиснення даних за допомогою нейронних мереж.

Метою кваліфікаційної роботи є дослідження методів ефективного стиснення даних з використанням нейронних мереж.

1 ОПИС ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ СТИСНЕННЯ ДАНИХ ПОСТАНОВКА ЗАДАЧІ

Серед інформаційних технологій, що постійно розвиваються, ефективне обробка передача даних відіграють ключову роль у формуванні функціональності та продуктивності різних програмних систем. У сучасному світі дуже багато даних, які генеруються в різних галузях, починаючи від телекомунікацій і охорони здоров'я до фінансів і розваг. Як наслідок, потреба в передових системах стиснення даних стає дедалі гострішою, що спонукає дослідників та інженерів досліджувати інноваційні рішення, які виходять за рамки звичайних методів.

На відміну від традиційних методів стиснення, які часто покладаються на заздалегідь визначені алгоритми, інтелектуальна система стиснення даних використовує передові технології, такі як машинне навчання та штучний інтелект, для адаптивної оптимізації процесу стиснення. Цей підхід обіцяє вирішити динамічну та складну природу сучасних наборів даних, що призведе до підвищення коефіцієнтів стиснення, зменшення вимог до зберігання та прискореної швидкості передачі даних.

1.1 Сучасні алгоритми стиснення даних

Сучасні алгоритми стиснення даних представляють собою підхід на основі традиційних методів та передових математичних і обчислювальних методів.

Однією з помітних особливостей сучасних алгоритмів стиснення даних є їхня здатність знаходити баланс між ефективністю стиснення та складністю обчислень. На відміну від попередніх алгоритмів, які надавали перевагу простоті над адаптивністю, сучасні алгоритми використовують складні стратегії, часто черпаючи натхнення з таких галузей, як теорія інформації, машинне навчання та обробка сигналів. Ці підходи дозволяють алгоритмам краще обробляти ті надмірності в різних типах даних, що призводить до більш ефективних процесів стиснення та розпакування.

Існує значна кількість алгоритмів стиснення даних. Розглянемо основні сучасні методи та алгоритми стиснення даних та дослідим, що означає поняття стиснення даних.

Стиснення даних – це процес зменшення обсягу або кількості бітів, байтів чи символів у наборі даних. Головна мета стиснення даних полягає в економії місця для зберігання та більш ефективному передаванні цих даних через мережу чи зберіганні на носіях. Цей процес може бути затратний (зменшення обсягу з певними втратами інформації) або безвтратний (збереження всієї інформації).

Стиснення без втрат - це спосіб зменшити розмір файлу, не змінюючи його вмісту. Цей метод використовується, коли важливо, щоб відновлений файл був ідентичним оригіналу.

Існує багато різних алгоритмів стиснення без втрат, кожен з яких оптимізований для певного типу даних. Наприклад, для текстових файлів використовуються алгоритми, які виявляють повторювані шаблони символів. Для графічних файлів використовуються алгоритми, які використовують статичні особливості зображення. Для аудіо файлів використовуються алгоритми, які використовують статичні особливості звуку.

Стиснення без втрат використовується в багатьох програмах, таких як файлові архіватори, текстові редактори та музичні плеєри.

Переваги стиснення без втрат:

– відновлені файли ідентичні оригіналам;

– цей метод використовується для зберігання та передачі даних, які повинні бути точними.

Стиснення з втратами - це спосіб зменшити розмір файлу, не відновлюючи його вміст до оригіналу. Цей метод працює шляхом видалення або заміни деяких даних, які можуть бути непомітні для людського ока або вуха.

Наприклад, щоб зменшити розмір зображення, можна видалити деякі пікселі, які не сильно впливають на загальну якість зображення. Або, щоб зменшити розмір

аудіофайлу, можна видалити деякі високочастотні компоненти звуку, які можуть бути непомітні для людського вуха.

Стиснення з втратами використовується в багатьох програмах, таких як музичні плеєри, потокові відеосервіси та програми для обробки фотографій.

Переваги стиснення з втратами:

– цей метод може забезпечити значне зменшення розміру файлу;

– він може бути використаний для мультимедійних даних, таких як аудіо та відео, які можуть бути сприйнятливі до незначних змін.

Алгоритм RLE (Run-Length Encoding) - це один з найстаріших і найпростіших алгоритмів стиснення графіки. Він працює, замінюючи послідовності однакових пікселів на пару лічильник-значення.

Лічильник - це число, яке вказує, скільки разів повторюється один і той же піксель. Значення - це сам піксель.

Наприклад, якщо в оригінальному зображенні є послідовність з 64 синіх пікселів, то алгоритм RLE замінить її на пару (64, 0).

RLE добре працює для зображень, в яких є багато однорідних областей. Наприклад, для ділової графіки, в якій часто використовуються однотонні фони.

Однак, алгоритм RLE може збільшити розмір файлу, якщо в оригінальному зображенні є багато послідовностей пікселів, які не повторюються. Наприклад, для кольорових фотографій.

Для того, щоб збільшити зображення в два рази, алгоритм RLE можна застосувати до зображення, в якому всі пікселі мають значення більше 11000000 і не повторюються попарно.

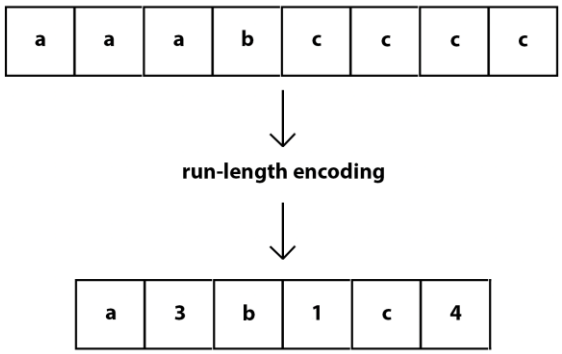

Рисунок 1.1 – Приклад роботи алгоритму RLE

Алгоритм Лемпеля-Зіва-Велча (LZW) - це універсальний алгоритм стиснення даних без втрат, який працює, прогнозуючи наступні символи в стрічці даних і кодуючи їх за допомогою коротких кодів.

Безпосереднім попередником LZW був алгоритм LZ78, який був опублікований в 1978 році Абрахамом Лемпелем і Якобом Зівом [1]. Цей алгоритм працює аналогічно LZW, але він не зберігає таблицю кодів. Це означає, що він не може стискати дані, які містять повторювані послідовності символів, які раніше не зустрічалися.

LZW був розроблений Террі Велчем в 1984 році [2]. Він усунув недолік LZ78, зберігаючи таблицю кодів. Це дозволило LZW стискати дані, які містять будь-які повторювані послідовності символів.

LZW використовується в багатьох програмах і форматах файлів, включаючи:

– архіватори, такі як ZIP і gzip;
– формати графічних файлів, такі як TIFF і GIF;
– формати аудіофайлів, такі як FLAC і ALAC.

LZW є ефективним алгоритмом стиснення, який часто може скоротити розмір файлу на 50-70%. Він також є відносно простим у реалізації, що робить його популярним вибором для багатьох програм.

Алгоритм Хаффмана - це алгоритм стиснення даних без втрат, який працює, присвоюючи символам алфавіту коди з мінімальною довжиною.

Алгоритм Хаффмана складається з двох етапів:

– побудова оптимального кодового дерева. На цьому етапі алгоритм будує дерево, в якому вузли представляють символи алфавіту, а ребра представляють коди символів;

– побудова відображення код-символ. На цьому етапі алгоритм створює таблицю, яка відображає символи алфавіту на їхні коди.

Побудова оптимального кодового дерева здійснюється за допомогою жадібного алгоритму. На кожному кроці алгоритм вибирає два вузли з найменшими вагами і об'єднує їх у новий вузол. Вага вузла дорівнює ймовірності зустрічі відповідного символу в даних.

Побудова відображення код-символ здійснюється шляхом обходу кодового дерева зверху вниз. Символи, які відповідають вузлам на одному рівні дерева, отримують однакові коди.

Алгоритм Хаффмана є ефективним алгоритмом стиснення, який часто може скоротити розмір файлу на 50-70%. Він також є відносно простим у реалізації, що робить його популярним вибором для багатьох програм.

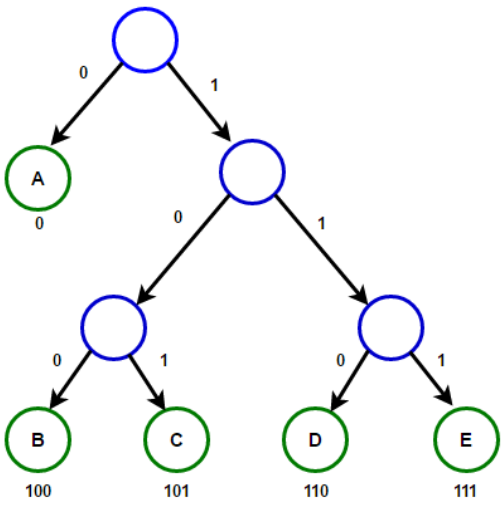

Рисунок 1.2 – Візуалізація дерева Хаффмана

Lossless JPEG - це алгоритм стиснення даних без втрат, який був розроблений групою експертів в області фотографії в 1993 році [3]. Він є спеціальною версією стандарту JPEG, яка забезпечує побітову відповідність між вихідним і стиснутим зображенням.

Основна цільова група Lossless JPEG - це повноколірні зображення (24-бітні) або зображення в градаціях сірого без палітри (8-бітові). Ці типи зображень часто використовуються в медичних, наукових і виробничих додатках, де важлива точність відтворення даних

Алгоритм Lossless JPEG працює шляхом застосування дискретного косинусного перетворення (DCT) до кожного пікселя зображення. DCT розкладає піксель на ряд гармонік, які потім кодуються за допомогою алгоритму Хаффмана.

Lossless JPEG забезпечує ефективне стиснення даних без втрат. Він може скоротити розмір файлу зображення до 50-70%. Однак, він не так популярний, як стандартний JPEG, який забезпечує більший коефіцієнт стиснення, але з деякою втратою якості.

JBIG - це стандарт стиснення зображень без втрат, який був розроблений Групою експертів зі стиснення бінарних зображень (Joint Bi-level Image Experts Group) і стандартизований Міжнародною організацією зі стандартизації ISO/IEC як стандарт номер 11544 і як ITU-T рекомендації T.82. [4]

JBIG був розроблений в першу чергу для стиснення факсимільних зображень, які часто містять великі області однотонних пікселів. Він також може використовуватися для інших типів зображень, таких як текстові документи, схеми та графіки.

JBIG працює за допомогою алгоритму арифметичного стиснення, який називається Q-coder. Цей алгоритм оцінює ймовірність кожного біта на основі попередніх бітів та попередніх рядків зображення.

JBIG забезпечує ефективне стиснення зображень без втрат. Він може скоротити розмір файлу зображення до 50-70% [5], а в деяких випадках навіть більше.

JBIG також підтримує прогресивну передачу, що дозволяє отримувачу поступово отримувати зображення по мірі його передачі. Це може бути корисно для таких додатків, як передача факсимілів по телефонній лінії.

Трансформуючі кодеки - це тип кодеків, які стискають дані, перетворюючи їх у іншу форму, в якій вони містять менше інформації. Ці кодеки використовують знання про об'єкт кодування, щоб визначити, яку інформацію можна відкинути без втрати якості.

Трансформуючі кодеки працюють у кілька етапів:

— розподілення. Об'єкт кодування розбивається на невеликі частини;

— трансформація. Кожна частина об'єкта кодування перетворюється в іншу форму, в якій вона містить менше інформації;

— стиснення. Перетворена форма об'єкта кодування стискається за допомогою одного з методів стиснення даних.

Фінальний файл, отриманий за допомогою трансформуючих кодеків, може відрізнятися від вихідного, однак він достатньо схожий на нього, щоб дозволяти подальше використання.

Предиктові кодеки - це тип алгоритмів стиснення даних, які використовують попередні дані для прогнозування поточних даних. Потім різниця між прогнозом і фактичними даними кодується для зменшення розміру файлу.

Предиктові кодеки можуть використовуватися для стиснення будь-якого типу даних, включаючи зображення, звук і текст. Вони особливо ефективні для даних, які мають тенденцію до повторення, таких як зображення з великими однотонними областями.

Принцип роботи предиктових кодеків полягає в наступному:

— алгоритм прогнозує поточний елемент на основі попередніх даних;

— алгоритм розраховує різницю між прогнозом і фактичними даними;

— різниця кодується для зменшення розміру файлу.

JPEG - це формат зберігання графічної інформації, який використовує стиснення з втратами якості зображення. Він був розроблений групою експертів з фотографії в 1992 році і є одним з найпоширеніших форматів зображень у світі. [6]

JPEG працює за допомогою наступних етапів:

— дискретний косинусний перетворення (DCT) розкладає зображення на ряд гармонік;

— зигзагоподібне сканування перетворень дозволяє зберегти лише найбільш важливі гармоніки;

— DPCM-передбачення середніх значень блоку зменшує кількість інформації, яку потрібно зберігати;

— скалярне квантування зменшує точність гармонік;

— ентропійне кодування кодує результати попередніх етапів для зменшення розміру файлу.

1.2 Технології нейронних мереж в стисненні даних

Нейронні мережі - це тип обчислювальних моделей, які натхненні людським мозком. Вони складаються з вузлів, які пов'язані між собою вагами. Ваги визначають, як інформація передається від одного вузла до іншого.

З математичної точки зору, нейронні мережі представляють собою метод розв'язання нелінійних завдань оптимізації. Кібернетика використовує теорію нейронних мереж для вирішення завдань адаптивного управління та створення алгоритмів для робототехніки. У сфері програмування нейронні мережі використовуються як засіб ефективного паралелізму.

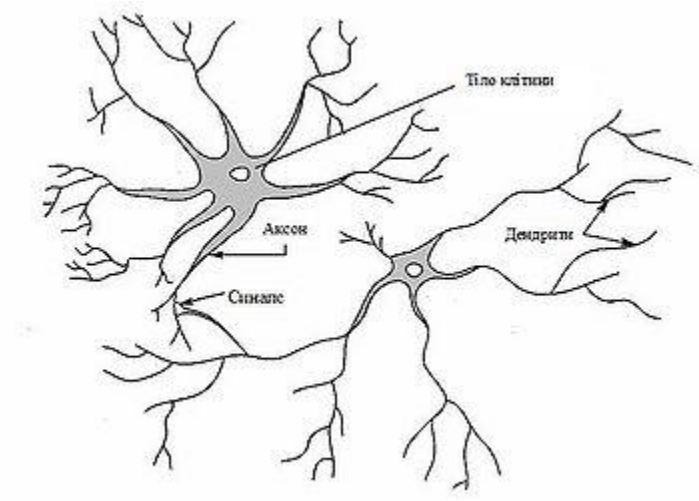

Рисунок 1.3 – Візуалізація біологічного нейрона

Під час програмування нейронних мереж основний акцент робиться на процесі навчання мережі, а не на написанні програмного коду. Цей підхід дозволяє мережі виявляти залежності між вхідними та вихідними даними, узагальнювати їх, спрощувати результати та використовувати отримані знання для розбиття складних завдань на більш прості.

Нейронні мережі мають ряд переваг перед іншими методами машинного навчання. Вони можуть вчитися на великих наборах даних, які можуть бути занадто великими для інших методів машинного навчання. Вони також можуть обробляти складні дані, які можуть бути важкими для розуміння для людей.

Нейронні мережі використовуються в широкому спектрі застосувань, включаючи класифікацію, рекомендацію та генерацію.

Історія нейронних мереж . Найпростіший тип нейронної мережі - це лінійна нейронна мережа. Вона має один шар вузлів виходу, і входи подаються безпосередньо на виходи. В кожному вузлі обчислюється сума добутків ваг та даних входів. Середньоквадратичні похибки між цими обчисленими виходами та заданими цільовими значеннями мінімізуються шляхом підлаштовування ваг.

Цей метод відомий як метод найменших квадратів або лінійна регресія. Він був розроблений понад два століття тому і використовується для вирішення різних завдань, таких як прогнозування та класифікація.

Історія нейронних мереж почалася в 19 столітті. Вільгельм Ленц та Ернст Ізінг створили модель Ізінга, яка є рекурентною нейронною мережею без навчання. Воррен Маккалох та Волтер Піттс представили ненавчену обчислювальну модель для нейронних мереж. Наприкінці 1940-х років Д. О. Гебб створив гіпотезу навчання, яка стала основою для геббового навчання [7].

У 1958 році Френк Розенблат винайшов перцептрон, першу втілену штучну нейронну мережу. Він фінансувався Управлінням військово-морських досліджень США [8].

У 1969 році Мінскі та Пейперт опублікували книгу, в якій стверджували, що базові перцептрони не здатні обробляти схему виключного «або». Це призвело до застою в дослідженнях нейронних мереж [9].

Однак на момент виходу цієї книги вже були відомі методи тренування багатошарових перцептронів.

Перший БШП глибокого навчання був опублікований Олексієм Григоровичем Івахненко та Валентином Лапою 1965 року. Перший БШП глибокого навчання, навчений стохастичним градієнтним спуском, був опублікований Сунь'їті Амарі 1967 року [10].

Самоорганізаційні карти були описані Теуво Кохоненом 1982 року. Вони є нейрофізіологічно натхненими нейронними мережами, які навчаються низьковимірного подання високовимірних даних [11].

Архітектуру згорткової нейронної мережі (ЗНМ) запропонував Куніхіко Фукусіма 1980 року [12]. Він назвав її неокогнітроном. 1969 року він також запропонував передавальну функцію ReLU [13].

ЗНМ стали важливим інструментом комп'ютерного бачення. Вони використовуються для розпізнавання об'єктів, розпізнавання обличь і інших завдань.

У 1980-х роках глибоке навчання було важко реалізувати за допомогою зворотного поширення. Це було пов'язано з тим, що зворотне поширення стає менш ефективним з кожним додатковим шаром нейронної мережі [14].

Щоб подолати цю проблему, Юрген Шмідхубер запропонував ієрархію рекурентної нейронної мережі (РНМ), попередньо тренованої по одному рівню самокерованим навчанням. Ця мережа використовує передбачувальне кодування для навчання внутрішніх подань у кількох самоорганізованих масштабах часу. Це може істотно полегшити подальше глибоке навчання.

1992 року Шмідхубер також опублікував альтернативу РНМ, яка називається лінійним трансформером. Він навчається внутрішніх центрів уваги, які спрямовують швидкі ваги іншої нейронної мережі[15].

У 2017 році Ашиш Васвані зі співавторами запропонували сучасний трансформер, який поєднує увагу з оператором softmax та проєкційною матрицею. Трансформери все частіше використовуються для обробки природної мови та комп'ютерного бачення[16].

У 2018 році Nvidia розробила StyleGAN, яка є GAN з прогресивним навчанням. У StyleGAN породжувач вирощується від малого до великого пірамідним чином. Це дозволяє досягти відмінної якості зображення.[17]

Класифікація нейронних мереж. Нейронні мережі (NN) є одними з найпопулярніших і найпотужніших моделей машинного навчання. Вони можуть використовуватися для вирішення широкого спектру завдань, включаючи розпізнавання зображень, розпізнавання мови, машинний переклад, прогнозування та багато іншого.

Нейронні мережі можна використовувати для вирішення широкого спектру завдань, включаючи розпізнавання образів, розпізнавання мовлення, машинний переклад, прогнозування та багато іншого.

Конкретний вид виконуваного нейронною мережею перетворення інформації залежить від характеристик нейронів та особливостей архітектури мережі.

Нейроні мережі можна класифікувати за різними ознаками а саме:

1) за типом вхідної інформації:

— аналогові мережі використовують інформацію у формі дійсних чисел;

— бінарні мережі (виконавчі) операційні з інформацією, представленою в двійковому вигляді;

2) за типом функції активації нейронів:

— неперервні мережі використовують диференційовані функції активації та операцію над дійсними числами;

— дискретні мережі мають недиференційовані функції активації та працюють з бінарною інформацією;

— дискретно-неперервні мережі поєднують елементи з диференційованими та недиференційованими функціями;

3) за типом графа міжнейронних зв'язків:

— мережі без циклів (ациклічні);

— мережі з циклами, які можуть бути рівноважними або обмеженими циклами;

4) за типом структур нейронів:

— гомогенні (однорідні) мережі складаються з нейронів одного типу та єдиною функцією активації;

— гетерогенні (неоднорідні) мережі мають нейрони з різними функціями активації;

5) за кількістю шарів нейронів:

— одношарові мережі містять лише один шар нейронів;

– багатошарові мережі (БНМ) мають більше одного шару взаємопов'язаних нейронів;

6) за типом дискримінантної функції:

– мережі зважені мають ваги для зв'язків між нейронами;

– мережі без ваг зв'язків працюють без врахування вагових коефіцієнтів;

7) за принципом синтезу:

– мережі, що навчаються: графи між нейронних зв’язків та ваги входів змінюються під час навчання за використанням певного методу;

– мережі, що конструюються: кількість і тип нейронів, граф між нейронних зв’язків та ваги входів нейронів визначаються при створенні мережі з урахуванням постановки задачі;

8) за топологією зв’язків:

– повно зв’язні (інтерактивні) мережі: всі нейрони пов’язані за принципом «кожен з кожним», включаючи самого себе. всі вхідні сигнали подаються всім нейронам;

– неповно зв’язні (шаруваті, багатошарові, ієрархічні) мережі: мають шарувату організацію з латеральними (бічними) та проекційними (аферентними) зв’язками між нейронами різних шарів. Інформація рухається від вхідного до вихідного прошарку;

– ієрархічно інтерактивні мережі: шари різного рівня пов’язані двосторонніми зв’язками, існують зв’язки між елементами одного шару, структура зв’язків не є однорідною, як в повністю інтерактивній мережі;

– слабо зв’язні (з локальними зв’язками) мережі: нейрони розташовані у вузлах прямокутної або гексагональної решітки, кожен нейрон зв’язаний з обмеженим числом найближчих сусідів;

9) за характером поділу зв’язків:

– немонотонні мережі: не дозволяють визначити, як зміна будь-якого внутрішнього параметра мережі вплине на вихідний сигнал;

– монотонні мережі: кожен шар мережі, крім останнього, поділяється на два блоки - збудливий і гальмуючий. Зв’язки влаштовані так, що елементи збудливої частини шару впливають на наступний збудливий шар і гальмують наступний гальмуючий шар, і навпаки. Монотонність вимагає монотонної залежності вихідного сигналу нейрона від параметрів вхідних сигналів;

10) за типом методу навчання (налаштування синапсів):

– мережі з динамічними зв’язками і ітеративним навчанням: Використовують модель повторення шляхів нервового збудження, основний метод - зворотне поширення помилки. Гарантує асимптотичну збіжність процесу навчання, але може вимагати багаторазового повторення ітерацій навчання на всьому обсязі даних;

– мережі з фіксованими зв’язками і неітеративним навчанням: Використовують методи прямого обчислення значень матриці зв’язків. Це прискорює процес навчання порівняно з ітеративними методами, але такі мережі характеризуються надмірною кількістю нейронів і низьким обсягом інформації, що запам’ятовується.

Узагальнюючи, нейронні мережі представляють різноманітну класифікацію за різними аспектами. Вони відзначаються різними типами вхідної інформації, функцій активації нейронів, графів між нейронних зв'язків, структур нейронів, кількістю шарів, типами дискримінантних функцій, принципами синтезу та топологією зв'язків. Ця класифікація надає розуміння різноманітності і можливостей нейронних мереж у різних вирішеннях завдань, включаючи обробку різноманітної інформації та навчання за різними методами.

Типи нейронних мереж та їх застосування у стиснені даних. Багатошаровий персептрон (БМП) - це тип нейронної мережі, який складається з двох або більше шарів нейронів. Кожен шар нейронів пов'язаний з наступним шаром за допомогою вагових зв'язків.

БМП можуть виконувати більш складні завдання, ніж одношарові перцептрони, такі як багатокласове класифікування, регресія та асоціативне навчання.

Функція активації БМП може бути будь-якою функцією, яка приймає значення вхідного сигналу і перетворює його в значення вихідного сигналу. Найчастіше використовуються такі функції активації:

– сигмоїдна функція;
– функція Релу;
– функція Тангенсу гіперболічного.

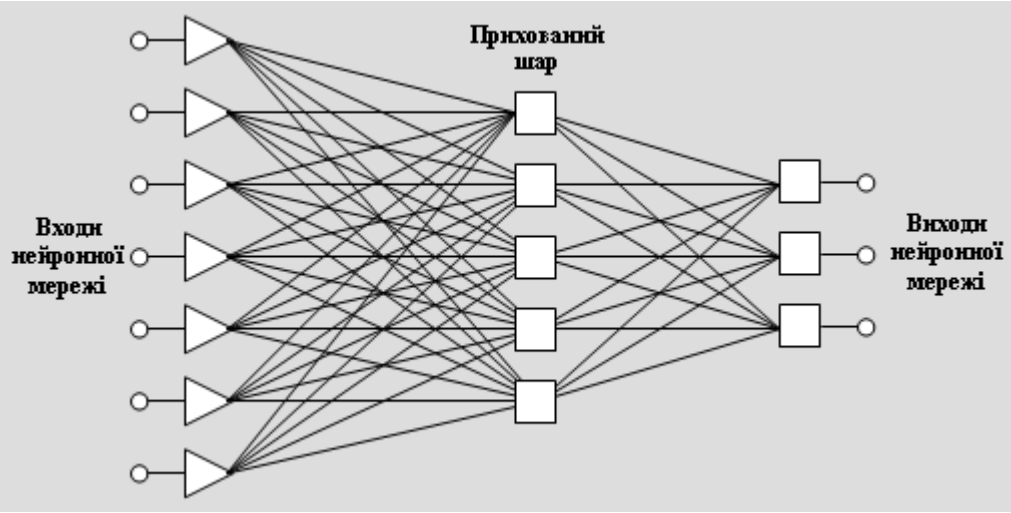

Рисунок 1.4 – Візуалізація моделі багато шарового персептрона

Згорткові нейронні мережі (ЗНМ) - це тип нейронних мереж, які спеціально розроблені для обробки даних, які мають структуру, наприклад, зображення, відео або текст.

ЗНМ складаються з кількох шарів нейронів, кожен з яких виконує певну функцію. Перший шар називається згортковим шаром і відповідає за виявлення локальних ознак у даних. Наступні шари називаються з'єднаними шарами і відповідають за об'єднання та узагальнення інформації, виявленої згортковим шаром.

Згортковий шар виконує операцію, яка називається згорткою. Згортка - це математична операція, яка об'єднує значення сусідніх пікселів зображення або сусідніх символів тексту.

З'єднаний шар виконує операцію, яка називається функцією активації. Функція активації - це функція, яка перетворює значення вхідних сигналів на значення вихідних сигналів. Найчастіше використовуються такі функції активації:

– сигмоїдна функція;
– функція Релу;
– функція Тангенту гіперболічного.

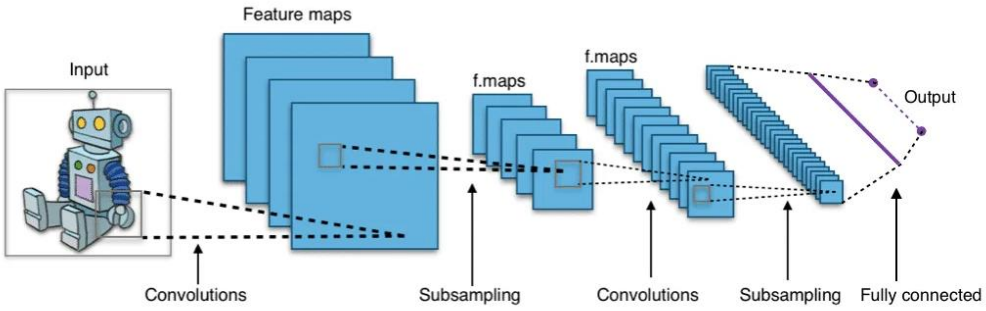

Рисунок 1.5 – Візуалізація Загорткової нейронної мережі

Рекурентні нейронні мережі (РНМ) - це тип нейронних мереж, які можуть обробляти інформацію, яка має часову структуру, наприклад, текст, мова або відео.

РНМ складаються з кількох шарів нейронів, кожен з яких пов'язаний з попереднім і наступним шарами. Це дозволяє мережі зберігати інформацію про попередні вхідні сигнали та використовувати її для обробки поточних вхідних сигналів.

РНМ знаходять широке застосування в різних галузях, таких як:

– розпізнавання мовлення;
– машинний переклад;
– генерація тексту;

– прогнозування;

– сприйняття мовлення.

Автоенкодери (autoencoders) - це тип нейронної мережі, яка використовує для зменшення розмірності даних через його кодування та декодування. Основна ідея полягає в тому, щоб навчити модель відтворювати вхідні дані у вихідному шарі за допомогою скрижалей, названих кодером та декодером.[18]

Структура автоенкодера включає дві основні частини:

Кодер (Encoder): Перетворює вхідні дані у вектор, який представляє собою зменшену розмірність (зазвичай, це код).

Декодер (Decoder): Відтворює вхідні дані з отриманого коду.

Автоенкодери використовуються для вирішення задач різних типів:

– зменшення розмірності даних: вони можуть виділяти основні ознаки та зменшувати розмірність вхідних даних, корисно для візуалізації та обробки великих обсягів даних;

– генерація нових даних: після тренування можна використовувати декодер для створення нових прикладів, що подібні до тих, які були використані під час тренування:

– вилучення ознак (Feature learning): автоенкодери допомагають вилучити важливі ознаки з даних, які можна використовувати для інших задач, таких як класифікація чи кластеризація.

Загальний принцип роботи автоенкодера - навчання моделі відтворювати свій вхід, і тим самим здобути корисні представлення даних.

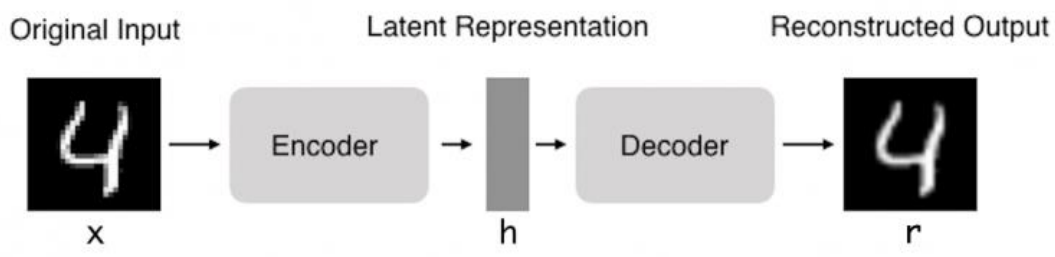

Рисунок 1.6 – Візуалізація автоенкодера

Довга короткострокова пам'ять (LSTM) - це тип рекурентних нейронних мереж (RNN), розроблений для ефективного моделювання довгострокових залежностей в послідовностях даних. Виникли для подолання проблем, пов'язаних зі зниканням та вибуванням градієнтів, які можуть виникати при тренуванні традиційних RNN.

Основні компоненти LSTM включають:

літинний стан (Cell State): це внутрішнє представлення, яке зберігає інформацію про контекст та довгострокові залежності. Вона може додавати та видаляти інформацію за допомогою воріт.

Ворота (Gates): LSTM має три типи воріт: вхідні ворота (input gate), забуті ворота (forget gate) та вихідні ворота (output gate). Вони керують потоком інформації в та з клітинного стану.

– вхідні ворота контролюють, які вхідні дані впливають на стан комірки;

– забуті ворота контролюють, які частини стану комірки мають бути збережені або забуті;

– вихідні ворота контролюють, які частини стану комірки виходять з комірки.

Вхідні ворота отримують вхідні дані та попередній стан комірки як вхідні дані. Вони використовують ці дані для обчислення значення, яке множиться на вхідні дані. Це значення визначає, які вхідні дані впливають на стан комірки.

Забуті ворота отримують попередній стан комірки як вхідні дані. Вони використовують ці дані для обчислення значення, яке множиться на попередній стан комірки. Це значення визначає, які частини попереднього стану комірки мають бути збережені або забуті.

Вихідні ворота отримують попередній стан комірки як вхідні дані. Вони використовують ці дані для обчислення значення, яке множиться на попередній стан комірки. Це значення визначає, які частини попереднього стану комірки виходять з комірки.

Тангенціальна функція активації (Hyperbolic Tangent Activation): Використовується для створення векторів значень, які додаються або видаляються з клітинного стану.

LSTM працюють, змінюючи свій стан за допомогою цих трьох воріт. На кожному кроці часу нові вхідні дані надходять до LSTM. Вхідні ворота контролюють, які з цих нових даних впливають на стан комірки. Забуті ворота контролюють, які частини попереднього стану комірки мають бути збережені або забуті. Вихідні ворота контролюють, які частини стану комірки виходять з комірки.

Ці ворота працюють разом, щоб забезпечити, щоб LSTM могла запам’ятовувати та обробляти інформацію в послідовних даних. Наприклад, LSTM може використовуватись для перекладу мови, розпізнавання рукописного тексту або прогнозування погоди.

Нейронні мережі можна використовувати для стиснення даних за допомогою різних методів. Один із найпоширеніших методів - це використання автоенкодерів.

Автоенкодери складаються з двох частин: кодувальника та декодувальника. Кодувальник перетворює вхідні дані у код, який є меншим за розміром і містить меншу кількість інформації. Декодувальник перетворює код у вихідні дані, які повинні бути якомога подібні до вхідних даних.

Навчання автоенкодера здійснюється за допомогою методу зворотного поширення помилки. Цей метод передбачає, що мережа отримує набір даних, який містить вхідні та вихідні значення. Потім мережа використовується для прогнозування вихідних значень для кожного вхідного значення. Похибка між прогнозованими та фактичними вихідними значеннями використовується для зміни вагових зв'язків мережі. Цей процес повторюється до тих пір, поки помилка не стане прийнятно малою.

Автоенкодери можуть бути використані для стиснення різних типів даних, таких як зображення, відео та аудіо. Вони можуть бути ефективними для стиснення даних, які мають структуру або повторювані елементи.

Ось кілька прикладів застосування автоенкодерів для стиснення даних:

– зображення: автоенкодери можуть бути використані для стиснення зображень, не втративши при цьому значущої інформації. Це дозволяє зберігати зображення в меншому обсязі, що може бути корисним для передачі зображень по мережі або для зберігання зображень на пристроях з обмеженим обсягом пам'яті;

– відео: автоенкодери можуть бути використані для стиснення відео, не втративши при цьому значущої інформації. Це дозволяє зберігати відео в меншому обсязі, що може бути корисним для передачі відео по мережі або для зберігання відео на пристроях з обмеженим обсягом пам'яті:

– аудіо: автоенкодери можуть бути використані для стиснення аудіо, не втративши при цьому значущої інформації. Це дозволяє зберігати аудіо в меншому обсязі, що може бути корисним для передачі аудіо по мережі або для зберігання аудіо на пристроях з обмеженим обсягом пам'яті.

Крім автоенкодерів, нейронні мережі можна використовувати для стиснення даних за допомогою інших методів, таких як глибинне стиснення та субсемплювання.

Глибинне стиснення - це метод стиснення даних, який використовує глибинні нейронні мережі для виявлення повторюваних елементів у даних.

Субсемплювання - це метод стиснення даних, який полягає в вибірковому видаленні деяких даних. Нейронні мережі можуть бути використані для вибору даних, які можна видалити без втрати значущої інформації.

Переваги застосування нейронних мереж для стиснення даних:

– нейронні мережі можуть досягати високої ефективності стиснення, не втративши при цьому значущої інформації;

– нейронні мережі можуть бути адаптовані до різних типів даних.

Недоліки застосування нейронних мереж для стиснення даних:

– нейронні мережі можуть бути складними для навчання;

– нейронні мережі можуть бути чутливі до шуму в даних.

Перспективи розвитку застосування нейронних мереж для стиснення даних:

– вчені працюють над розробкою нових методів навчання нейронних мереж для стиснення даних, які будуть більш ефективними та менш чутливими до шуму в даних;

– вчені також працюють над розробкою нових методів стиснення даних за допомогою нейронних мереж, які будуть більш ефективними для різних типів даних.

1.3 Постановка задачі

Як було задано вище мета роботи полягає в розробці та вивченні ефективних методів стиснення даних з використанням нейронних мереж для зменшення обсягу збереженої і переданої інформації, зберігаючи при цьому якість та корисність даних.

Для досягнення мети потрібно вирішити наступні задачі:

– розробити дата сет для стиснення зображення;

– обробити данні допомогою різних методів та підходів очищення даних;

– розробити архітектури нейронних мереж для стиснення даних;

– порівняти розроблених методів стиснення з існуючими стандартами стиснення даних щодо ступеня стиснення і якості відтворення;

– провести оцінку обчислювальної складності розроблених методів і їх придатності для реальних застосувань;

– дослідити вплив параметрів нейронних мереж на результат стиснення, таких як розмір моделі, кількість шарів, рівень компресії та інші гіпер параметри.

Висновки до розділу 1

Отже в цьому розділу було розглянуто сучасні алгоритми стиснення даних. Ці алгоритми можна класифікувати за типом стиснення (з втратами або без втрат), за типом даних (текст, зображення, звук, відео) та за методом стиснення.

Було також розглянуто поняття, типи, та класифікацію нейронних мереж. Нейронні мережі - це тип обчислювальних моделей, які натхненні людським мозком. Вони складаються з вузлів, які пов'язані між собою вагами. Ваги визначають, як інформація передається від одного вузла до іншого.

Нейронні мережі використовуються в широкому спектрі застосувань, включаючи класифікацію, рекомендацію та генерацію, а також стиснення даних.

2 МАТЕМАТИЧНІ МОДЕЛІ МЕТОДИ ДЛЯ ВИРІШЕННЯ ЗАВДАННЯ СТИСНЕННЯ ДАНИХ

З розвитком сучасних технологій і зростанням обсягів інформації, проблема ефективного зберігання та передавання даних стає важливішою ніж коли-небудь. Відновлення та оптимізація простору, який займають дані, є ключовим завданням, і для цього широко використовуються методи стиснення даних. Розділ 2 цієї роботи присвячений вивченню математичних моделей, методів та алгоритмів, які використовуються для ефективного стискання різноманітних типів інформації.

У цьому розділі ми розглянемо різні підходи та методи стиснення даних. Зокрема, буде досліджено алгоритми стиснення, що базуються на різноманітних математичних моделях, розберемосяя із поняттям стиснення з втратами та без втрат, та розглянемо методи штучного інтелекту які можна, і використовують у завданні стиснення даних.

2.1 Методи та алгоритми стиснення даних без втрат

Стиснення даних без втрат — це процес скорочення обсягу інформації без втрати якості чи важливої інформації. Цей підхід забезпечує ефективну оптимізацію великих обсягів даних, зберігаючи при цьому їхню повноту та точність.

Алгоритм Хафмана. У інформатиці та теорії інформації кодування Хаффмана — це алгоритм ентропійного кодування, який використовується для стиснення даних без втрат [22]. Термін стосується використання кодової таблиці змінної довжини для кодування вихідного символу (наприклад, символу у файлі), де кодову таблицю змінної довжини було отримано певним чином на основі оціненої ймовірності появи для кожного можливе значення вихідного символу.

Кодування Хаффмана базується на частоті появи елемента даних [19]. Принцип полягає у використанні меншої кількості бітів для кодування даних, які

зустрічаються частіше[20]. Середня довжина коду Хаффмана залежить від статистичної частоти, з якою джерело створює кожен символ зі свого алфавіту.

Словник кодів Хаффмана [21] , який асоціює кожен символ даних із кодовим словом, має властивість, що жодне кодове слово в словнику не є префіксом будь-якого іншого кодового слова в словнику[28] . Основою для цього кодування є кодове дерево відповідно до Хаффмана, яке призначає короткі кодові слова символам, які часто використовуються, і довгі кодові слова символам, які рідко використовуються як для коефіцієнтів постійного, так і для змінного струму, кожен символ кодується кодом змінної довжини від Хаффмана набір таблиць, призначений компоненту зображення блоку 8x8. Коди Хаффмана повинні вказуватися зовні як вхідні дані для кодувальників JPEG. Зауважте, що форма, в якій таблиці Хаффмана представлені в потоці даних, є непрямою специфікацією, за допомогою якої декодер повинен побудувати самі таблиці перед декомпресією [24]. Алгоритм побудови кодування дотримується цього алгоритму, кожен символ є листом і коренем. Блок-схема алгоритму Хаффмана зображена на малюнку 2.1.

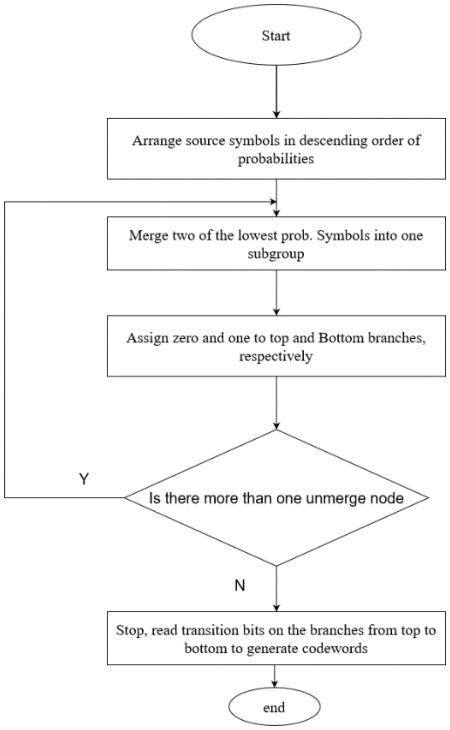

Рисунок 2.1 – Блок-схема алгоритма Хаффмана.

Розглянемо як цей алгоритм працює більш детально на прикладі. Припустімо, що наведений нижче рядок потрібно надіслати через мережу.

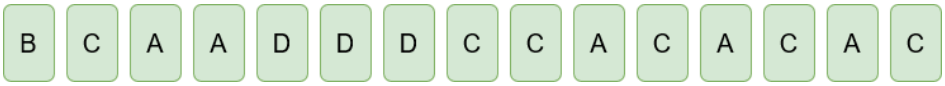

Рисунок 2.2 – Початковий рядок

Кожен символ займає 8 біт. Загалом у наведеному вище рядку 15 символів. Таким чином, для надсилання цього рядка потрібно загалом 8 * 15 = 120 біт. Використовуючи техніку кодування Хаффмана, ми можемо стиснути рядок до меншого розміру. Кодування Хаффмана спочатку створюємо дерево, використовуючи частоти символу, а потім генеруємо код для кожного символу. Коли дані закодовані, їх необхідно декодувати. Декодування здійснюється за допомогою того ж дерева. Кодування Хаффмана запобігає будь-якій неоднозначності в процесі декодування за допомогою концепції префіксного коду, тобто. код, пов'язаний із символом, не повинен бути присутнім у префіксі будь-якого іншого коду. Кодування Хаффмана виконується за допомогою наступних кроків. Обчислюємо частоту кожного символу в рядку.

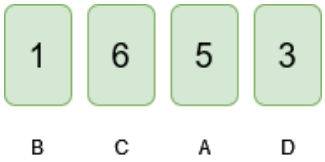

Рисунок 2.3 – Частота рядка

Відсортуємо символи в порядку збільшення частоти. Вони зберігаються в пріоритетній черзі Q. Зробимо кожен унікальний символ листковим вузлом. Створимо порожній вузол z. Призначимо мінімальну частоту лівому нащадку z і призначмо другу мінімальну частоту правому нащадку z. Встановимо значення z як суму двох зазначених вище мінімальних частот.

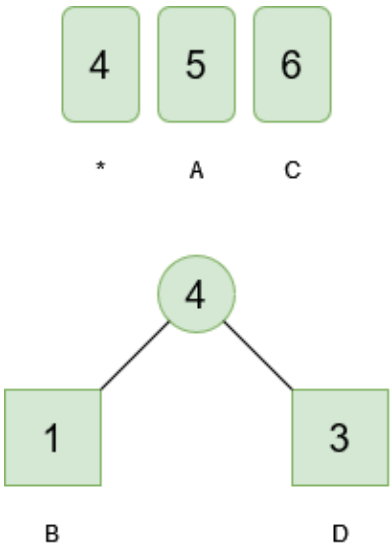

Рисунок 2.4 – Отримання суми найменших чисел

Вилучимо ці дві мінімальні частоти з Q і додаємо суму до списку частот (* позначемо внутрішні вузли на малюнку вище). Вставимо вузол z у дерево, та повторимо кроки 3-5 для всіх символів.

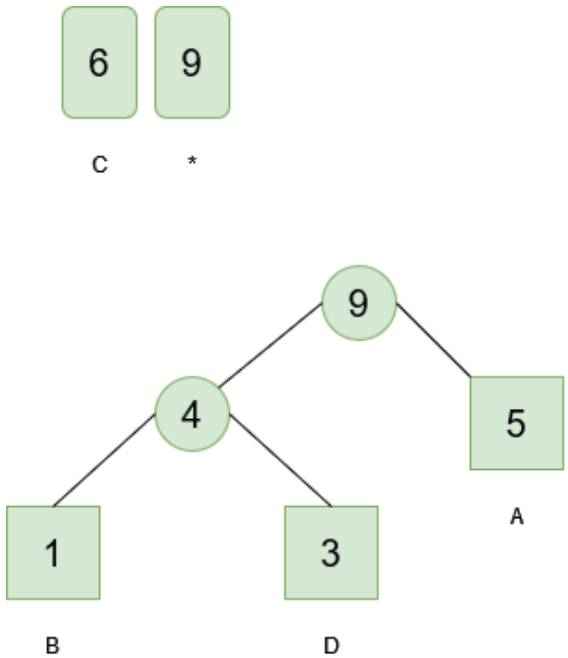

Рисунок 2.5 – Повторіть кроків 3-5 для всіх символів.

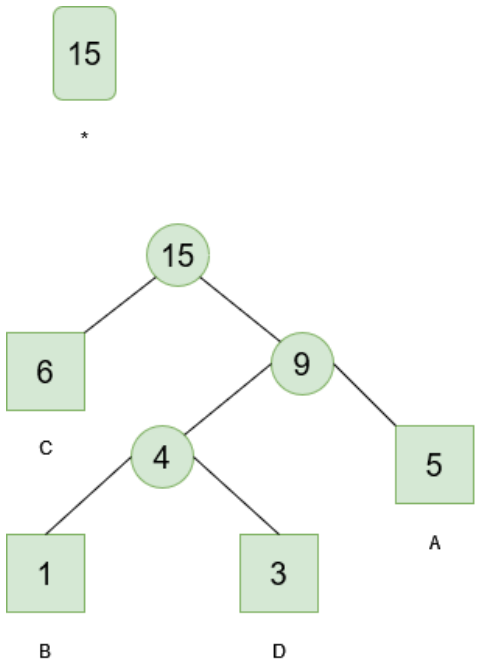

Рисунок 2.6 – Повторення кроків 3-5 для всіх символів.

Для кожного нелистового вузла призначимо 0 лівому краю та 1 правому краю.

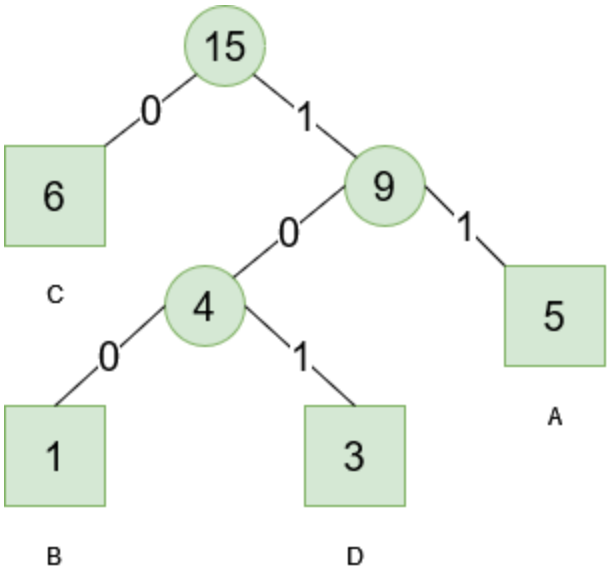

Рисунок 2.7 – Призначимо 0 лівому краю та 1 правому краю

Щоб надіслати наведений вище рядок через мережу, ми повинні надіслати дерево, а також наведений вище стиснутий код. Загальний розмір наведено в таблиці нижче.

Таблиця 2.1 – кодування для передачі рядка через мережу

Символ	Частота	код	розмір
A	5	11	5*2 = 10
B	1	100	1*3 = 3
C	6	0	6*1 = 6
D	3	101	3*3 = 9
4 * 8 = 32 bits	15 bits		28 bits

Без кодування загальний розмір рядка становив 120 біт. Після кодування розмір зменшується до 32 + 15 + 28 = 75.

Для декодування коду ми можемо взяти код і пройти по дереву, щоб знайти символ. Нехай 101 має бути декодовано, ми можемо перейти від кореня, як показано на малюнку нижче.

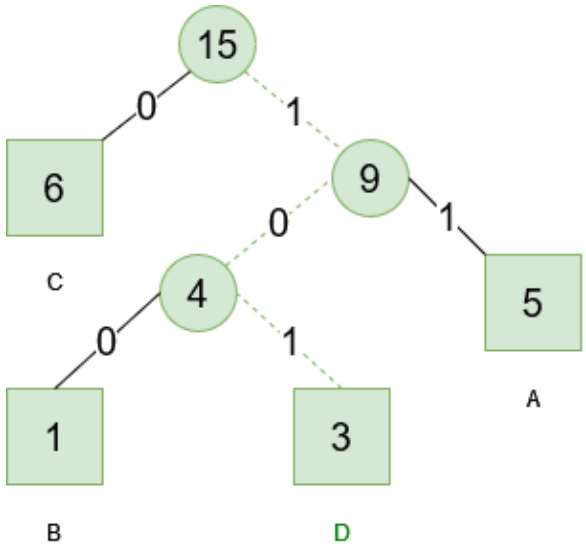

Рисунок 2.8 – Декодування

На прикладі рядка символів було продемонстровано, як за допомогою кодування Хаффмана можна значно зменшити розмір переданих даних. У цьому випадку, розмір рядка скоротився з 120 біт до 75 біт, що свідчить про високу ефективність цього методу стиснення.

В цілому, алгоритм Хаффмана є важливим інструментом у сучасних технологіях обробки та передачі даних, що забезпечує ефективне використання ресурсів та оптимальну передачу інформації.

Алгоритм JPEG-LS. Метод стиснення JPEG-LS, який є стандартом стиснення зображень без втрат, заснований на алгоритмі LOCO-I [29]. Він відрізняється від методів JPEG та JPEG2000, незважаючи на схожу назву [32, 34]. JPEG-LS рекомендується використовувати в системах з обмеженими ресурсами, наприклад, космічних станціях або веб-камерах.

Блок-схеми алгоритмів стиснення JPEG (рис. 2.9), JPEG2000 (рис. 2.10) та JPEG-LS (рис. 2.11) наочно показують їхню структуру.

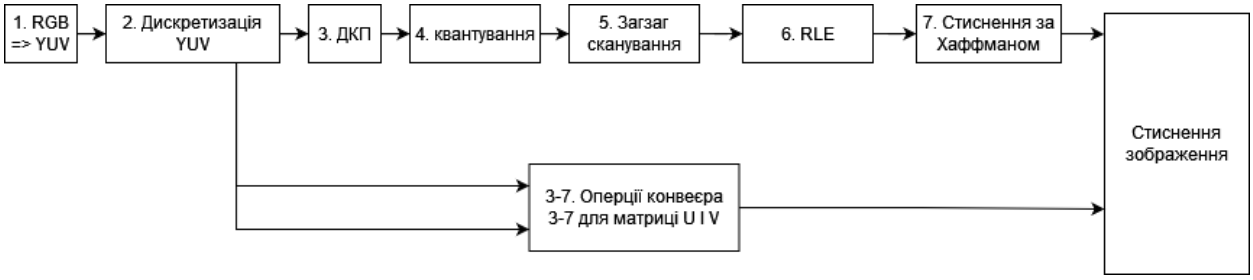

Рисунок 2.9 – Схема алгоритму стиснення зображень JPEG

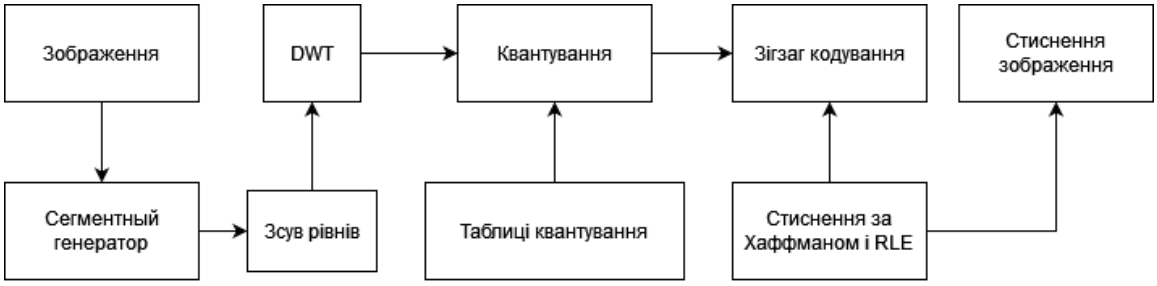

Рисунок 2.10– Схема алгоритму стиснення зображень JPEG2000

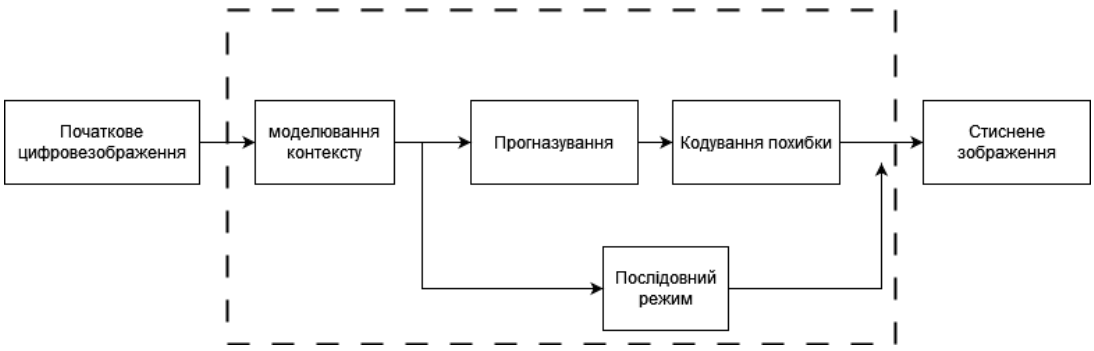

Рисунок 2.11 – Схема алгоритму стиснення зображень JPEG-LS

Алгоритм JPEG-LS для стиснення зображень без втрат складається з трьох основних етапів: моделювання, прогнозування та кодування. Основною його відмінністю від інших алгоритмів є модуль прогнозування, який використовує контекст для зменшення розміру коду [29].

Для прогнозування значення чергового пікселя x використовуються пікселі контексту a, b, c, d, розташовані навколо нього. Залежно від контексту вибирається режим прогнозування: послідовний або регулярний.

Послідовний режим використовується, якщо очікується, що значення пікселів x і y будуть близькими. У цьому режимі кодер починає перегляд поточного рядка з пікселя x і знаходить найбільшу послідовність пікселів, що збігаються з контекстним пікселем a. Ця послідовність кодується довжиною.

Регулярний режим використовується, якщо очікується, що значення пікселів x і y будуть різними. У цьому режимі для прогнозування значення пікселя x використовується значення пікселів a, b і c. Потім обчислюється помилка прогнозу Errval, яка дорівнює різниці значення x і прогнозованого значення. Errval коригується, а потім кодується за допомогою кодів Голомба. Код Голомба залежить від контексту, тобто від значень пікселів a, b, c, d і Errval. Прогнозоване значення для поточної вибірки x можна розрахувати за формулою 2 .

$$\hat{x}_{MED} \triangleq \begin{cases} \min(a,b) & \text{if } c \geq \max(a,b) \\ \max(a,b) & \text{if } c \leq \min(a,b) \\ a+b-c & \text{otherwise} \end{cases}, \quad (2.1)$$

Прогнозоване значення для поточної вибірки x визначається на основі значень пікселів a, b, c і D, розташованих навколо неї.

Вузол передбачення виконує простий тест для виявлення вертикального або горизонтального краю. Якщо вертикальне ребро існує зліва від поточного місця розташування, то прогнозоване значення дорівнює значенню пікселя b. Якщо горизонтальний край існує над поточним місцезнаходженням, то прогнозоване значення дорівнює значенню пікселя a. Якщо жодне ребро не виявлено, то прогнозоване значення дорівнює середньому значенню пікселів a, b і c.

2.2 Методи та алгоритми стиснення даних з втратами

Стиснення даних з втратами – це процес, під час якого інформація скорочується або зменшується з метою зменшення обсягу даних, проте цей процес супроводжується втратою певної частини оригінальної інформації. Цей вид стиснення особливо важливий у випадках, коли обсяг зберігання чи передачі даних є критичним, і деяка втрата якості прийнятна.

Одним із загальновизнаних методів стиснення даних з втратами є метод втратного стиснення зображень. Наприклад, використання алгоритмів JPEG дозволяє зменшити обсяг файлів зображень, при цьому здійснюючи втрати в якості зображення. Це досягається шляхом видалення частини деталей, які важко відрізнити для людського зору, або застосування стиску в кольоровому просторі.

Алгоритм JPEG — це стандарт стиснення зображень, розроблений Joint Photographic Expert Group. Це було офіційно прийнятий як міжнародний у 1992 році. JPEG складається з кількох кроків, кожен із яких сприяє компресії [38].

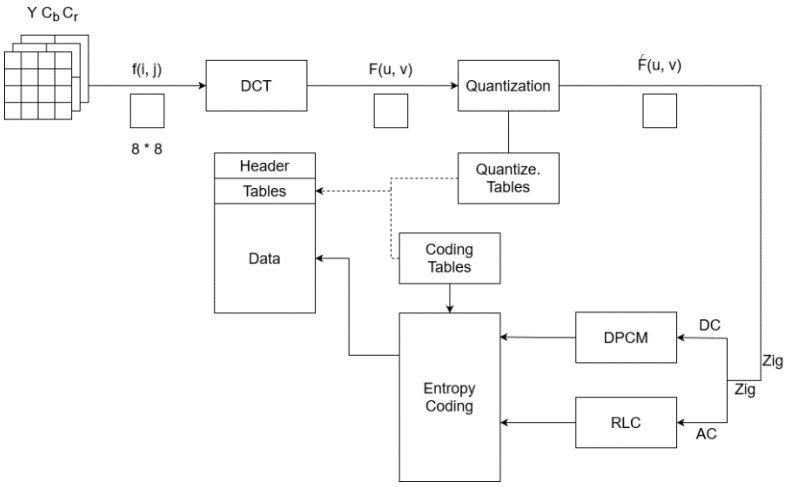

Рисунок 2.12 – Блок-схема кодера jpeg

На рисунку 2.12 показана блок-схема кодера JPEG. Якщо ми перевернемо стрілки на малюнку, ми в основному отримати декодер JPEG. Кодер JPEG складається з таких основних етапів. Перший крок стосується перетворення простору кольорів. Багато кольорових зображень представлені за допомогою колірного простору RGB. Представлення RGB, однак, сильно корельовані, що означає, що колірний простір RGB не є добре підходить для незалежного кодування . Так як зорова система людини менш чутлива до положення і руху кольору, ніж яскравість . Тому використовуються деякі перетворення колірного простору, такі як RGB в YCbCr [39] . Наступним кроком стандарту JPEG є дискретне косинусне перетворення (DCT). DCT виражає а послідовність скінченної кількості точок даних у термінах суми функцій косинусів, що коливаються на різних частоти. DCT є важливою частиною численних застосувань у науці та техніці без втрат стиснення мультимедійних даних . DCT розділяє зображення на частини з різними частотами. Вища частоти представляють швидкі зміни між пікселями зображення, а низькі частоти представляють поступові зміни між пікселями зображення. Щоб виконати DCT на зображенні, зображення має бути розділене на 8 × 8 або 16 × 16 блоків. Щоб зберегти деякі важливі коефіцієнти DCT, до перетвореного блоку застосовується квантування . Після цього кроку використовується зигзагоподібне сканування. У зображенні, яке було квантовано, багато серій нулів по всій матриці,

тому блоки 8 × 8 перевпорядковані як одиничні 64-елементні стовпці. Отримуємо вектор відсортовані за критерієм просторової частоти, що дає довгі серії нулів. Коефіцієнт DC обробляється окремо від 63 коефіцієнтів AC. Коефіцієнт постійного струму є мірою середнього значення зображення 64 зразки. Нарешті, на останніх етапах алгоритми кодування, такі як кодування довжини циклу (RLC) і код диференціального імпульсу Застосовується модуляція (DPCM) і ентропійне кодування. RLC є простим і популярним стисненням даних алгоритм . Він заснований на ідеї замінити довгу послідовність того самого символу коротшою послідовністю. Коефіцієнти постійного струму кодуються окремо від коефіцієнтів змінного струму. Коефіцієнт постійного струму кодується DPCM, який це техніка стиснення даних без втрат. Тоді як коефіцієнти AC кодуються за допомогою алгоритму RLC [48]. DPCM алгоритм записує різницю між коефіцієнтами DC поточного блоку та попереднього блоку . Оскільки зазвичай існує сильна кореляція між коефіцієнтами постійного струму сусідніх блоків 8×8, це призводить до набір подібних чисел з високою частотою. DPCM, проведений на пікселях із кореляцією між ними послідовні зразки призводять до хороших коефіцієнтів стиснення. Ентропійне кодування забезпечує додаткове стиснення використовуючи більш компактне кодування квантованих коефіцієнтів DCT на основі їх статистичних характеристик. В основному ентропійне кодування є критичним кроком стандарту JPEG, оскільки всі попередні кроки залежать від ентропійного кодування і важливо, який алгоритм використовується,. Пропозиція JPEG визначає два кодування ентропії алгоритми Хаффмана та арифметичне кодування.

Алгоритм Фрактального стиснення — це метод стиснення цифрових зображень із втратами на основі фракталів. Цей метод найкраще підходить для текстур і природних зображень, покладаючись на той факт, що частини зображення часто нагадують інші частини того самого зображення. Фрактальні алгоритми перетворюють ці частини в математичні дані, які називаються «фрактальними кодами», які використовуються для відтворення закодованого зображення.

Рисунок 2.13 – приклад результату алгоритму фрактального стиснення

Представлення фрактального зображення може бути описано математично як система ітерованих функцій (IFS). Розпочнемо з представлення бінарного зображення, де зображення можна розглядати як підмножину R^2. IFS — це набір скорочувальних відображень $f_1, \dots \dots \dots, f_N$.

$$f_i : R^2 \to R^2, \tag{2.2}$$

Відповідно до цих функцій відображення IFS описує двовимірну множину S як нерухому точку оператора Хатчінсона.

$$H(A) = \bigcup_{i=1}^{N} f_i(A), \;\; A \subset R^2, \tag{2.3}$$

Тобто H є оператором, що відображає множини на множини, а S є унікальною множиною, яка задовольняє $H(S) = S$. Ідея полягає в тому, щоб побудувати IFS так, щоб ця множина S була вхідним двійковим зображенням. Набір S можна

відновити з IFS шляхом ітерації з фіксованою точкою: для будь-якого непорожнього компактного початкового набору A_0, ітерація $A_{k+1} = H(A_k)$ сходиться до S.

Набір S є самоподібним, оскільки $H(S) = S$ означає, що S є об'єднанням відображених копій самого себе:

$$S = f_1(S) \cup f_2(S) \cup \ldots \cup f_N(S), \tag{2.4}$$

Отже, ми бачимо, що IFS є фрактальним представленням S. Представлення IFS можна розширити до зображення у градаціях сірого, розглядаючи графік зображення як підмножину R^3 Для зображення градацій сірого u(x,y) розглянемо набір $S = \{(x, y, u(x, y))\}$. Тоді, подібно до бінарного випадку, S описується IFS з використанням набору скорочувальних відображень $f_1, \ldots, f_N$, але в R^3.

$$f_i : R^3 \to R^3, \tag{2.5}$$

Складною проблемою поточних досліджень представлення фрактальних зображень є те, як вибрати $f_1, \ldots, f_N$ таким чином, щоб його фіксована точка наближала вхідне зображення, і як це зробити ефективно.

Простим підходом для цього є наступна система розділених ітерованих функцій (PIFS):

- розбийте область зображення на блоки діапазону R_i розміром s×s;
- для кожного R_i шукайте зображення, щоб знайти блок D_i розміром 2s×2s; який дуже схожий на R_i;
- виберіть функції відображення так, щоб $H(D_i) = R_i$ для кожного i.

На другому кроці важливо знайти подібний блок, щоб IFS точно представляв вхідне зображення, тому необхідно розглянути достатню кількість блоків-кандидатів для D_i. З іншого боку, великий пошук з урахуванням багатьох блоків

обчислюється дорого. Це вузьке місце пошуку подібних блоків є причиною того, що фрактальне кодування PIFS набагато повільніше, ніж, наприклад, представлення зображень на основі DCT і вейвлетів. DCT ми розглядали раніше давайте розглянемо що таке вейвлет.

Вейвлет - це хвилеподібне коливання з амплітудою, яка починається з нуля, збільшується або зменшується, а потім повертається до нуля один або кілька разів. Вейвлети називаються «короткими коливаннями». Встановлено таксономію вейвлетів на основі кількості та напрямку їх імпульсів. Вейвлети мають певні властивості, які роблять їх корисними для обробки сигналів.

Наприклад, можна створити вейвлет із частотою Middle C і короткою тривалістю приблизно одну десяту секунди. Якби цей вейвлет був згорнутий із сигналом, створеним із запису мелодії, тоді отриманий сигнал був би корисним для визначення того, коли в пісні з’явилася нота середнього до. Математично вейвлет корелює із сигналом, якщо частина сигналу схожа. Кореляція лежить в основі багатьох практичних вейвлет-додатків.

Як математичний інструмент, вейвлети можна використовувати для вилучення інформації з багатьох типів даних, включаючи аудіосигнали та зображення. Набори вейвлетів необхідні для повного аналізу даних. «Додаткові» вейвлети розкладають сигнал без проміжків або накладання, так що процес розкладання є математично оборотним. Таким чином, набори додаткових вейвлетів корисні в алгоритмах стиснення/декомпресії на основі вейвлетів, де бажано відновити вихідну інформацію з мінімальними втратами.

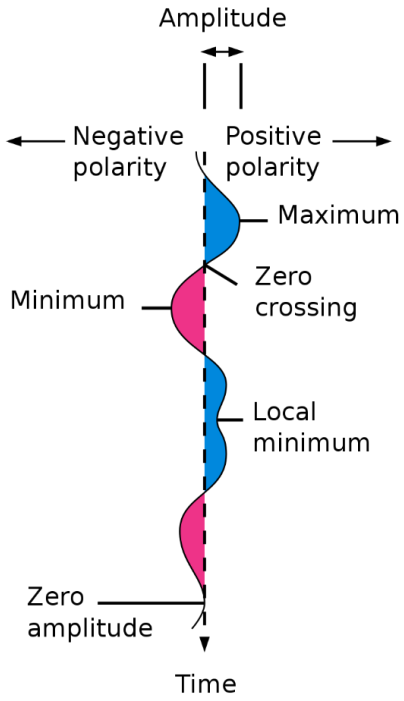

Рисунок 2.14 – Приклад сейсмічного вейвлету

Повернемося до фрактального кодування PIFS. Початкове квадратне розділення та алгоритм грубого пошуку, представлений Jacquin, є відправною точкою для подальших досліджень і розширень у багатьох можливих напрямках - різні способи розділення зображення на блоки діапазону різних розмірів і форм; швидкі методи швидкого пошуку досить близько відповідного блоку домену для кожного блоку діапазону, а не грубий пошук, наприклад, швидкі алгоритми оцінки руху; різні способи кодування відображення з блоку домену на блок діапазону; тощо

Інші дослідники намагаються знайти алгоритми для автоматичного кодування довільного зображення як RIFS (системи повторюваних ітерованих функцій) або глобальної IFS, а не PIFS; і алгоритми для фрактального стиснення відео, включаючи компенсацію руху та тривимірні ітеровані функціональні системи.

2.3 Огляд методів штучного інтелекту для завдання стиснення даних

Нейронна мережа — це математична модель, яка намагається відтворити структуру та функції нейронних зв'язків у мозку людини. Вона складається з великої кількості з'єднаних штучних нейронів, які працюють спільно для вирішення різноманітних завдань, використовуючи ваги та активації.

Кожен нейрон у мережі приймає вхідні сигнали, має ваги для цих сигналів, і, збираючи їх у взважену суму, передає результат через функцію активації. Такі мережі можуть бути одношаровими чи має багатошарову структуру, в залежності від завдання.

Нейронні мережі знаходять своє застосування в завданнях стиснення даних, зокрема у сучасних методах, таких як автоенкодери, Згорткові, рекуренті нейроні мережі, давайте розглянемо кожну модель більш детально.

Згорткова нейронна мережа (CNN) — це системи штучного інтелекту, засновані на багатошарових нейронних мережах, які можуть ідентифікувати, розпізнавати та класифікувати об’єкти, а також виявляти та сегментувати об’єкти на зображеннях. Насправді CNN або ConvNet — це популярна дискримінаційна архітектура глибокого навчання, яку можна вивчати безпосередньо з вхідного об’єкта без зобов’язань щодо вилучення функцій людиною [54]. Ця мережа часто використовується у візуальній ідентифікації, аналізі медичних зображень, сегментації зображень, НЛП та багатьох інших програмах, оскільки вона спеціально розроблена для роботи з низкою двовимірних форм. Він більш ефективний, ніж звичайна мережа, оскільки може автоматично ідентифікувати ключові елементи з вхідних даних без участі людини.

Розуміння різних компонентів CNN та їх застосування має вирішальне значення для розуміння прогресу в архітектурі CNN. На рисунку 2 показано кілька частин CNN.

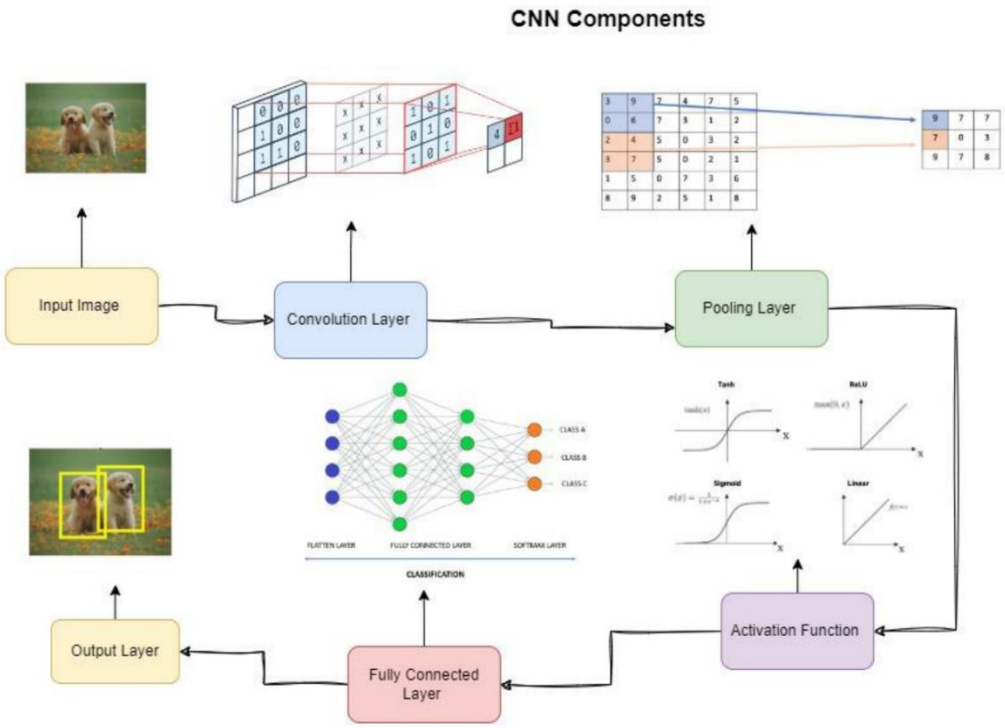

Рисунок 2.15 – Компоненти CNN

Шари CNN зазвичай складається з чотирьох типів

- згорткові рівні;
- об'єднання;
- функція активації.

Вхідне зображення Будівельні блоки комп’ютерного зображення називаються пікселями. Вони є двійковими для візуального представлення даних. Від 0 до 255 пікселів послідовно організовано в матричне розташування в компонуванні цифрового зображення. Яскравість і відтінок кожного пікселя визначаються значенням пікселя. Під час першого перегляду зображення мозок людини засвоює величезну кількість інформації.[57] Рівні CNN навчені спочатку розпізнавати основні візерунки, такі як лінії та криві, перш ніж переходити до більш складних візерунків, таких як обличчя та об’єкти. У результаті можна стверджувати, що використання CNN могло б забезпечити комп’ютери баченням

Згортковий рівень є важливою частиною загальної структури CNN. Це набір фільтрів або ядер, які застосовуються до даних перед їх використанням. Ширина,

висота та вага кожного ядра використовуються для отримання характеристик із вхідних даних. Ваги в ядрі спочатку призначаються випадковим чином, але поступово вони стають більш обґрунтованими даними навчання. Іншими словами, карта ознак створюється шляхом поєднання вхідного зображення (яке показано N-вимірними метриками) з цими фільтрами. Ядро — це набір дискретних значень або цілих чисел. Для кожного числа вага ядра вказана як довідкова. Початкові ваги ядра для CNN - це набір цілих чисел, вибраних навмання. Крім того, ваги ініціалізуються різними способами. У свою чергу, ядро вчиться отримувати значущі функції, оскільки ці ваги змінюються під час процесу навчання. Ядро дозволяє їм виконувати операції у багатовимірному неявному просторі функцій без обчислення координат даних у цьому просторі. Натомість вони обчислюють внутрішній добуток зображень усіх пар даних у просторі ознак. Застосовуючи трюк ядра до лінійної моделі, її можна перетворити на нелінійну модель. Вхідний формат CNN спочатку надається перед початком процесу згортки. Класична нейронна мережа приймає дані у векторному форматі, тоді як CNN приймає багатоканальне зображення. У той час як зображення RGB містить три кольорові «канали», зображення у градаціях сірого має лише один. Перегляньте це зображення у градаціях сірого розміром 4 × 4 із випадковим ініціалізованим ядром розміром 2 × 2, щоб дізнатися про згортки в дії. Ядро спочатку панорамується по горизонталі та вертикалі повного зображення. Скалярний добуток між вхідним зображенням і ядром також обчислюється паралельно; це досягається множенням відповідних значень і додаванням результатів для отримання єдиного скалярного значення. Після цього процедуру повторюють, доки ковзання більше не стане можливим [59].

$$(K - L + 1), \tag{2.6}$$

4 − 2 + 1 = 3, тож вихід 3 × 3 Насправді значення скалярного добутку вказують на карту характеристик результату. На рисунку 3 наочно представлені приклад

первинного обчислення, які виконуються на кожному етапі. На цій діаграмі менший квадрат (2 × 2) представляє ядро, тоді як більший квадрат (4 × 4) представляє вхідне зображення. Потім добуток представляється як число після множення на обидва, і ця сума забезпечує вхідне значення для вихідної карти ознак .

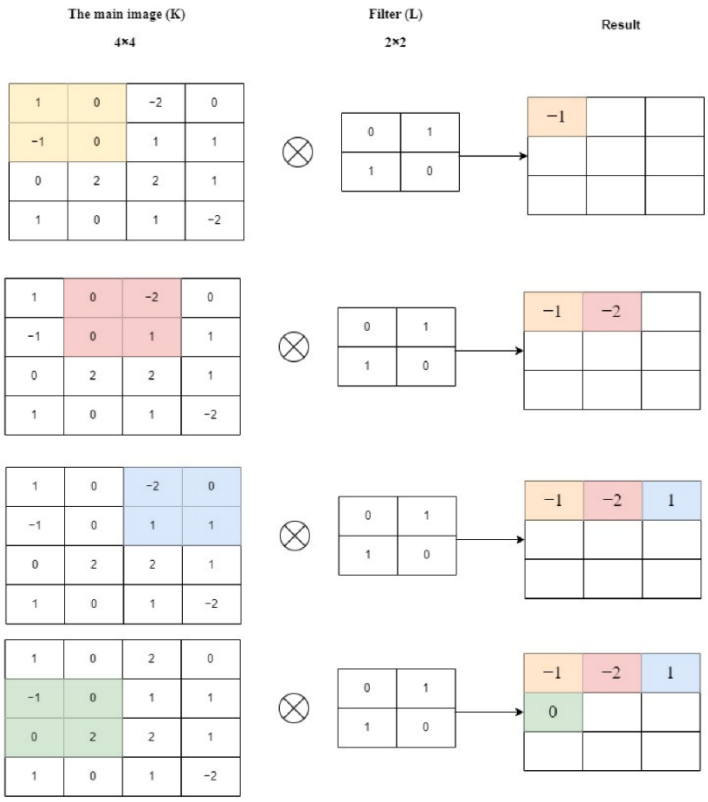

Рисунок 2.16 – Приклад візуального представлення первинних розрахунків.

Однак у попередньому прикладі ядру надається крок 1 (що позначає бажаний загальний розмір кроку для вертикальних або горизонтальних розташувань), але вхідне зображення не доповнюється. Дійсно, ви можете замінити інше значення кроку, якщо захочете. Додатковою перевагою збільшення значення кроку є зменшення розмірності результуючої карти ознак . Однак на розмір рамки наданого зображення значною мірою впливає відступ. Навпаки, характеристики сторони кордону різко змінюються з часом. Заповнення збільшує вхідне зображення, що також збільшує розмір карти функцій. Кожен фільтр може представляти функцію.

Фільтр не активується, коли він переміщується над зображенням і не виявляє відповідності. CNN використовує цей метод для виявлення найефективніших фільтрів опису об’єкта. На рисунку 16 показано приклад, як матриця може бути налаштована для пошуку країв зображення. Через те, що вони поводяться як звичайні фільтри, які використовуються в методах обробки зображень, ці матриці також відомі як фільтри.

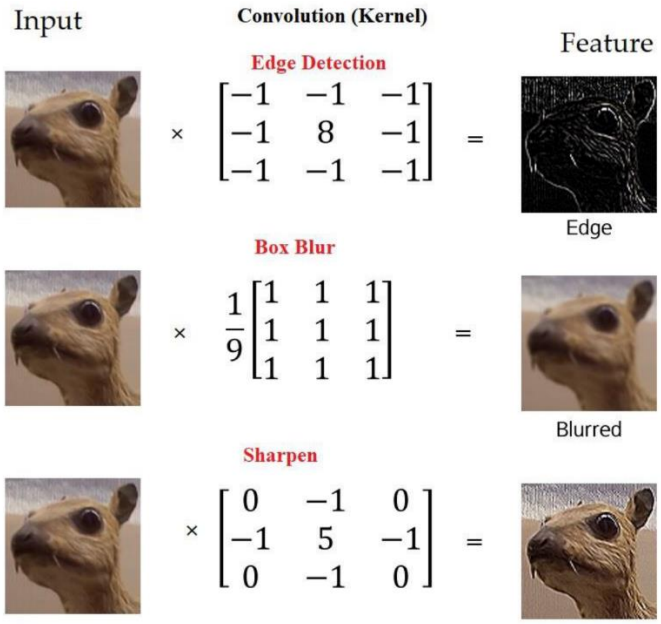

Рисунок 2.17 – Приклад ефекту різних матриць згортки.

Однак у CNN ці фільтри запускаються перед фільтрами форм, що використовуються в процесі навчання, які краще підходять для поточної роботи. Розподіл ваги: оскільки весь набір вагових коефіцієнтів у CNN діє на кожен піксель вхідної матриці, між будь-якими двома нейронами в сусідніх шарах немає призначених вагових коефіцієнтів. Вивчення однієї групи вагових коефіцієнтів для всього входу значно скоротить необхідний час навчання та різні витрати, оскільки додаткові вагові коефіцієнти для кожного нейрона не потрібно вивчати.

Насправді CNN пропонує додаткові параметри, які надають кілька можливостей для подальшого звуження налаштувань, а також зменшують деякі небажані наслідки. Один із таких варіантів називається кроком. У вищезгаданому

сценарії передбачається, що вузол наступного рівня має численні перекриття з сусідніми лише на основі дослідження областей. Ми можемо змінити перекриття, змінивши крок. Унікальне зображення 6 × 6 показано на малюнку 5. Оскільки фільтр можна переміщувати лише з кроком в один вузол, максимальний розмір виходу, якого ми можемо досягти, становить 4 × 4. Як видно на малюнку 5, існує перекриття між вихід трьох лівих матриць (а також трьох середніх разом і трьох правих також). Однак, якщо ми йдемо, вважаючи кожен крок за 2, загальна сума буде 3 помножена на 3. Іншими словами, загальний розмір виведення та загальне перекриття буде зменшено.

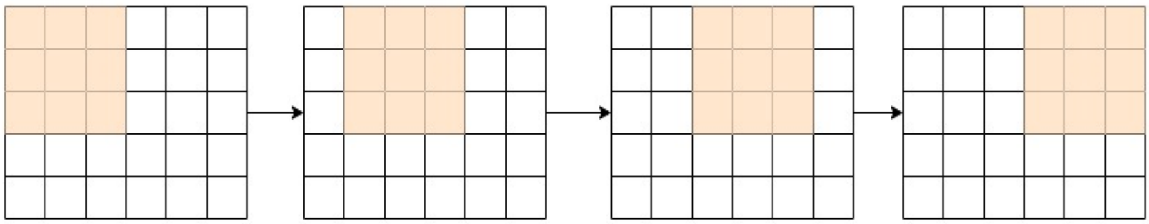

Рисунок 2.18 – Приклад першого кроку, вікна фільтрів переміщуються лише один раз для кожного з’єднання.

Рівняння (2) формалізує це, в результаті чого вихідний розмір O, враховуючи розмір зображення NXN і розмір фільтра FXF.

$$0 = 1 + \frac{(N - F)}{S}, \tag{2.7}$$

де N — вхідний розмір, F — розмір фільтра, а S — розмір кроку.

Одним із недоліків етапу згортання є можлива втрата деталей на краях зображення. Вони захоплюються лише під час переміщення фільтра, тому насправді їх ніколи не видно. Одним із простих і практичних рішень є використання нульового доповнення. Ви також можете керувати розміром виведення за допомогою нульового доповнення. На малюнку 6, наприклад, результат буде 4 × 4 (що зменшується від вхідного 6 × 6) з N = 6, F = 3 і кроком 1. Однак, включивши одне доповнення нулем, результат буде 6 × 6, що ідентично

початковому введенню (обчислення для фактичного N тепер становить 9). Формула є модифікованою формулою з нульовим доповненням (3).

$$O = 1 + \frac{N + 2P - F}{S}, \tag{2.8}$$

де P — це кількість шарів нульового заповнення (наприклад, $P = 1$ на малюнку 21). Використовуючи цю концепцію заповнення, ми можемо уникнути зменшення розміру мережевого виводу з глибиною. Отже, можлива будь-яка кількість глибоких згорткових мереж.

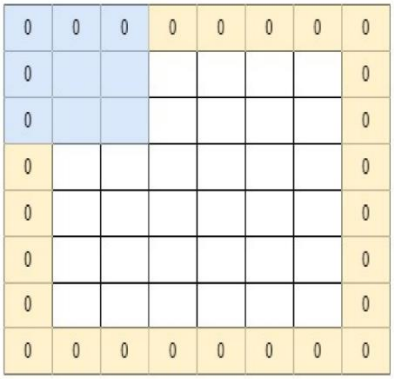

Рисунок 2.19 – Приклад першого кроку, вікна фільтрів переміщуються лише один раз для кожного з’єднання.

Завдяки вищезазначеному розподілу ваги модель також є інваріантною відносно поступальних змін. Функцію навчання можна відфільтрувати, щоб допомогти в будь-якому середовищі. Якщо початок із випадкових значень для фільтрів покращує продуктивність, тоді фільтри навчаться виявляти край (як на малюнку 3). Важливо відзначити, що спільна вага є поганою ідеєю при оцінці просторової значущості вхідних даних. Об'єднання Рівень об’єднання, також відомий як рівень понижуючої дискретизації, використовується для зменшення розмірності карт функцій, зберігаючи найважливіші дані. Фільтр застосовує операцію об’єднання до вхідних даних, ковзаючи по них у шарі об’єднання (max, min, avg). У літературі найчастіше використовується максимальне об’єднання. Важливою частиною об’єднання, яка використовується для зменшення складності

верхніх рівнів, є зменшення вибірки. З точки зору обробки зображення, це можна порівняти зі зменшенням роздільної здатності. Об’єднання не впливає на кількість фільтрів. Max-pooling є одним з найбільш часто використовуваних методів об'єднання. Зображення поділено на прямокутні під області, і повертається лише найбільше значення, виявлене в кожній під області. Один із найпоширеніших максимальних розмірів об’єднання — 2 × 2. Як показано на малюнку 23, коли об’єднання використовується для блоків 2 на 2 у верхньому лівому куті, увага перенаправляється у верхній правий кут і переміщується на два кроки. У результаті крок 2 використовується для об’єднання. Можна використовувати крок 1, який є незвичайним, щоб запобігти зниженню дискретизації. Майте на увазі, що зменшення дискретизації не зберігає позицію даних.

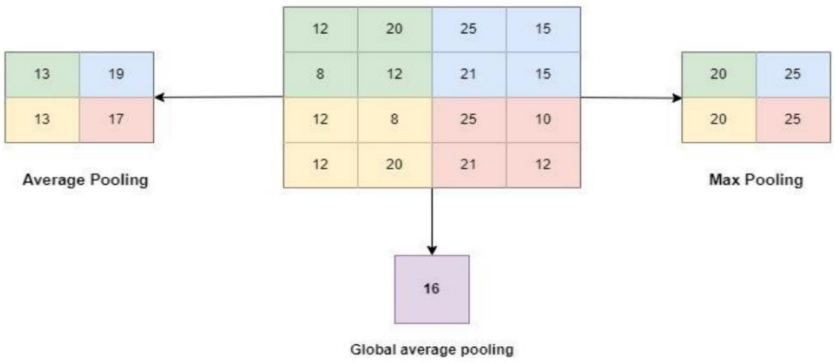

Рисунок 2.20 – Шар об'єднання.

На різних рівнях об'єднання можуть застосовуватися різні методи об'єднання. Глобальне середнє об’єднання (GAP), глобальне максимальне об’єднання, середнє об’єднання, мінімальне об’єднання та закрите об’єднання – це деякі з цих методів. На рисунку 8 зображено кожен із цих трьох методів об’єднання . Основна проблема з рівнем об’єднання полягає в тому, що він не допомагає CNN визначити, чи присутній елемент у вхідному зображенні, а скоріше лише, де він розташований. Тому бувають моменти, коли загальні рейтинги CNN падають. Однак модель CNN залишає необхідну інформацію.

На різних рівнях об'єднання можуть застосовуватися різні методи об'єднання. Глобальне середнє об’єднання (GAP), глобальне максимальне об’єднання, середнє об’єднання, мінімальне об’єднання та закрите об’єднання – це деякі з цих методів. На рисунку 8 зображено кожен із цих трьох методів об’єднання . Основна проблема з рівнем об’єднання полягає в тому, що він не допомагає CNN визначити, чи присутній елемент у вхідному зображенні, а скоріше лише, де він розташований. Тому бувають моменти, коли загальні рейтинги CNN падають. Однак модель CNN залишає необхідну інформацію.

Шар нелінійності слідує за згорткою. Нелінійність дозволяє змінювати або припиняти згенерований вихід. Цей шар використовується для обмеження або перенасичення виходу. Кожен тип функції активації в кожному типі нейронної мережі виконує основну функцію відображення вхідних даних і вихідних даних. Вхідне значення обчислюється шляхом обчислення зваженої суми вхідного сигналу нейрона та його зміщення (якщо воно є). Це вказує на те, що функція активації визначає, запускати чи ні нейрон у відповідь на певний вхід, генеруючи відповідний вихід. В архітектурі CNN нелінійні шари активації використовуються після всіх шарів із вагами (також відомі як навчальні шари, такі як шари FC і згорткові шари). Відображення вхідних даних і вихідних даних буде нелінійним через нелінійну продуктивність рівнів активації, і ці рівні також дозволяють CNN вивчати надзвичайно складні речі. Крім того, здатність диференціювати є важливою вимогою для функції активації, оскільки вона дозволяє використовувати зворотне поширення помилок для навчання мережі. Нижче наведено найпопулярніші функції активації в CNN та інших глибоких нейронних мережах: Sigmoid: Ця функція активації дозволяє виводити лише значення від 0 до 1 і приймає дійсні числа як вхідні дані. Tanh: її можна порівняти з сигмоїдною функцією, оскільки вона приймає дійсні числа як вхідні дані, але її вихідний діапазон становить лише від одиниці до одиниці. ReLU є найпопулярнішою функцією в контексті CNN. Усі вхідні значення перетворюються на позитивний

діапазон. Основною перевагою ReLU перед іншими алгоритмами є економія часу та ресурсів під час використання. Довгий час найпоширенішими були нелінійності Тан та сигмоїда. Нелінійності бувають у багатьох формах, і вони показані на малюнку 24.

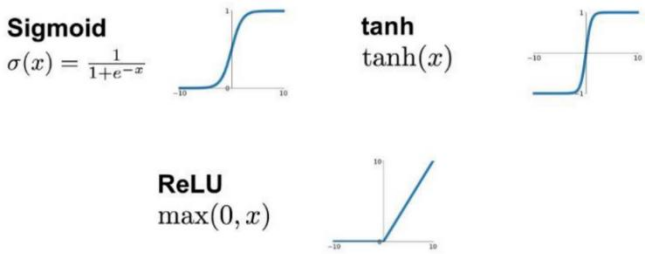

Рисунок 2.21 – Функції активації.

Однак з цих причин останніми роками випрямлена лінійна одиниця (ReLU) стала популярною. Визначення функцій і градієнтів за допомогою ReLU простіше. Насичені функції, такі як sigmoid і tanh, мають проблеми зі зворотним поширенням. Це явище, відоме як «зникаючий градієнт», виникає, коли сигнал градієнта поступово зменшується в міру того, як зростає глибина архітектури нейронної мережі. Це відбувається тому, що градієнт цих функцій по суті дорівнює нулю з усіх боків від центру. Тим не менш, ReLU має постійний градієнт для позитивного входу. Хоча функцію неможливо розрізнити, її можна ігнорувати під час впровадження. По-третє, ReLU генерує більш розріджене представлення, оскільки повний нуль створюється нульовим градієнтом. Для sigmoid і tanh результати градієнта ніколи не дорівнюють нулю, що може бути контрпродуктивним під час навчання. Під час використання ReLU час від часу можуть виникати кілька серйозних проблем. Розглянемо, наприклад, метод зворотного поширення помилок із більшим градієнтом, що протікає через нього. Вагові коефіцієнти будуть оновлені шляхом пропускання цього градієнта через функцію ReLU таким чином, щоб нейрон більше не стимулювався. Ця проблема відома як «Dying ReLU». Щоб вирішити ці проблеми, існує кілька замінників ReLU. Ось деякі з них, про які

йдеться нижче. Leaky ReLU: Ця функція активації гарантує, що негативні входи ніколи не ігноруються, на відміну від негативних входів, які зменшуються ReLU. Він використовується для вирішення проблеми Dying ReLU.

Нейрони організовані в групи в повністю пов’язаному шарі, які нагадують ті, що спостерігаються в традиційних нейронних мережах. Як показано на рисунку 2.22, будь-який повністю пов’язаний вузол шару безпосередньо пов’язаний з кожним вузлом шару над і під ним. На малюнку 10 показано, що кожен вузол в останніх кадрах рівня об’єднання з’єднаний як вектор від повністю підключеного рівня до верхнього рівня. Це параметри CNN, які найчастіше використовуються на цих рівнях; однак їм потрібно багато часу на навчання.

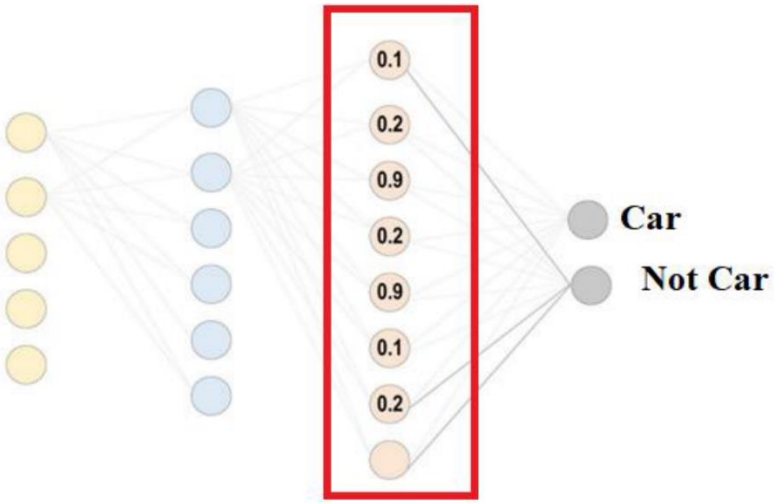

Рисунок 2.22 – Повністю зв'язаний шар.

Найбільшим недоліком повнозв’язного шару є велика кількість параметрів, які вимагають трудомістких розрахунків у навчальних вибірках. Тому ми намагаємося мінімізувати кількість з’єднань і вузлів. Усунуті вузли та з’єднання можуть бути задоволені за допомогою підходу відключення. LeNet і AlexNet, наприклад, розробили велику та глибоку мережу, зберігаючи постійну обчислювальну складність. Згортка, яка є основним елементом мережі CNN, стає відкритою, коли додається рівень нелінійності та об’єднання. Три, які найчастіше

використовуються в архітектурі: – Якщо перефразувати, то в повністю зв’язаному шарі всі нейрони спілкуються зі своїми аналогами в шарі нижче. Це класифікатор, який використовує CNN. – Будучи ШНМ прямого зв’язку, він працює подібно до звичайної багаторівневої мережі перцептронів. Вхідні дані до рівня FC надходять з останнього рівня об’єднання або згортки. – Це векторний вхід, створений шляхом збільшення товщини карт функцій [24]. На малюнку 10 показано, що результат FC-рівня узгоджується з остаточним результатом CNN[63]. У попередній частині обговорювалися різні типи шарів, що використовуються в дизайні CNN; цей розділ буде зосереджений на функціях втрат. Крім того, остаточна класифікація досягається шляхом використання вихідного рівня, останнього рівня архітектури CNN. Кілька функцій втрат використовуються у вихідному рівні моделі CNN для обчислення прогнозованої помилки в навчальних даних. В результаті цієї помилки виділяється розбіжність між фактичним і прогнозованим виходом. Потім його буде вдосконалено за допомогою підходу навчання CNN. Функція втрат, однак, використовує два входи, щоб точно визначити джерело помилки. Для CNN першим параметром є прогноз або оцінка випуску. Другий вхід - це бажаний результат або мітка. Існує багато різних типів функцій втрат, які використовуються для різних типів задач . Нижче наведено базове пояснення багатьох видів функцій втрат: Навчання: навчальний набір даних, що складається з колекції зображень і міток (класів, обмежувальних рамок і масок), використовується для навчання моделі CNN. Зворотне поширення — це процедура навчання CNN, яка вимірює значення помилки, використовуючи вихідне значення попереднього рівня. Вага кожного нейрона в цьому шарі оновлюється з використанням значення помилки. Щоб виміряти неправильне значення та переглянути старі ваги, використовуються нові ваги, як показано на малюнку 2.23.

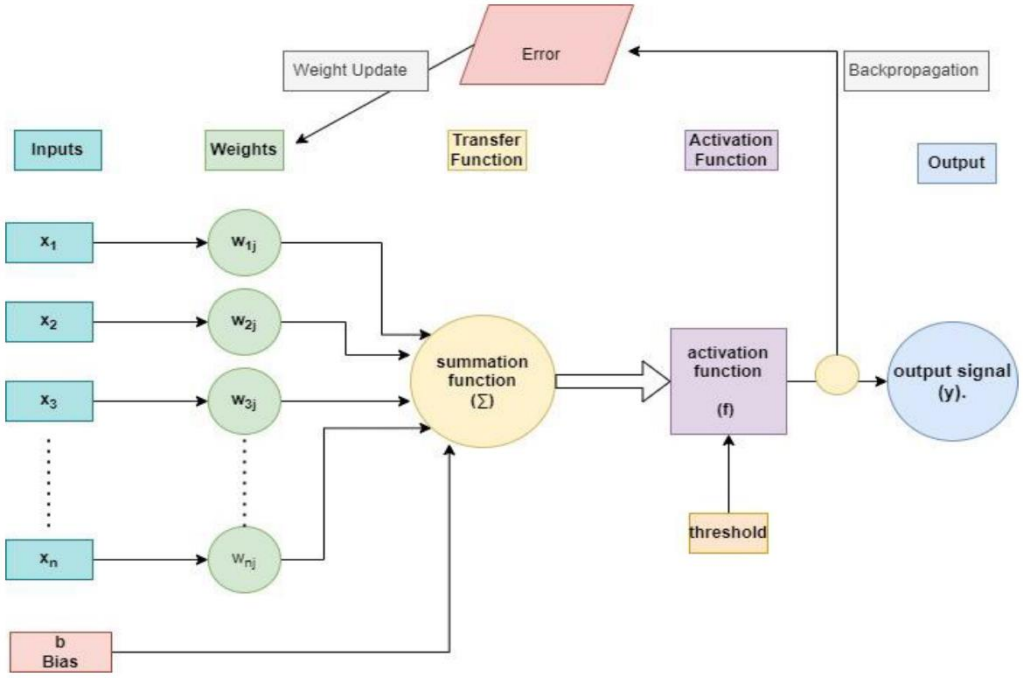

Рисунок 2.23 – Пряме і зворотне поширення в прихованих шарах CNN

Поки не досягне першого шару, алгоритм повторює процедуру. Усі входи, включаючи блок зміщення, підсумовуються блоком активації; потім скористайтеся функцією активації, щоб обчислити результат. Потім мережа розрахує функцію витрат і надішле помилку назад, щоб оновити ваги, доки вартість не буде мінімізована.

Автоенкодери. Щоб краще зрозуміти функціонування цих моделей, ми розкладемо їх на окремі компоненти. Ми опишемо структуру в тому ж порядку, в якому дані переміщуються нашою нейронною мережею, обговорюючи спочатку кодери, потім вузьке місце або «код», а потім декодери. Ці сегменти автокодерів можна розглядати як різні шари вузлів у моделі. Зображення нижче допомагає інтерпретувати ці три різні компоненти.

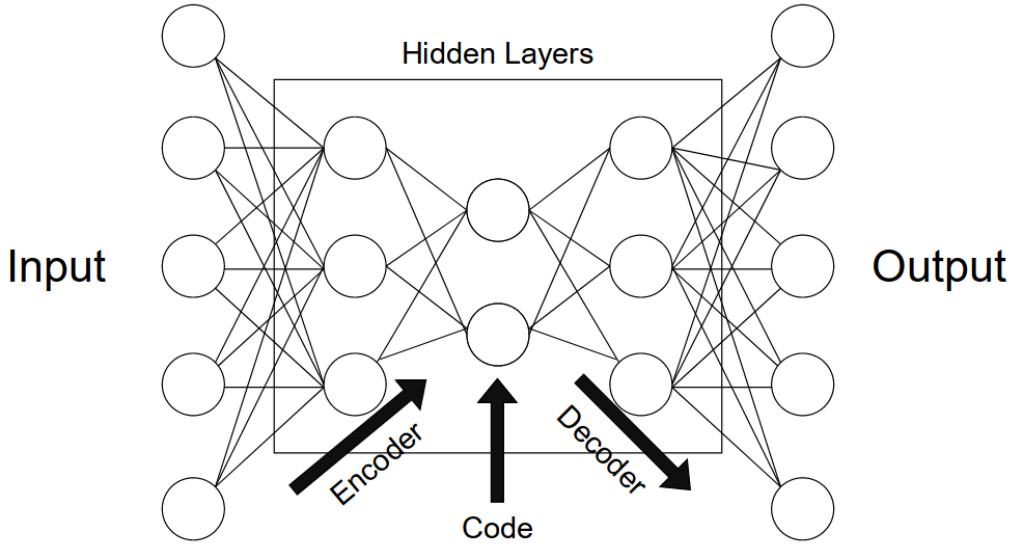

Рисунок 2.24 – Компоненти декодера

Компонент декодера мережі діє як інтерпретатор коду. У випадку згорткових нейронних мереж він може реконструювати зображення на основі певного коду. Ми можемо розглядати цей компонент як інструмент вилучення, інтерпретації чи декомпресії. Сегментація зображення зазвичай також є роботою декодерів. Якщо ми збережемо наш останній приклад класифікації зображень собак, ми зможемо продовжити декодування, додаючи копію попередніх стислих шарів до кожного шару вузлів декодера. Роблячи це, ми повинні отримати фотографію з підмножиною пікселів, які ідентифікують собаку на зображенні та відокремлюють її від фону та інших об’єктів. Генеративні моделі, такі як варіаційні автокодери (VAE), можуть навіть використовувати декодер для відтворення даних, яких не існує. Це може бути корисним для збільшення даних. Маючи різноманітні набори даних із генеративних моделей, інші моделі можуть більш ретельно вивчати дані.

Автокодери застосовуються в різних сферах машинного навчання та інформатики. Ось основні типи, з якими ми можемо зіткнутися, і їх відповідні загальні застосування.

Autoencoders для зняття шумів ці типи автокодерів призначені для ефективного кодування шумових даних, щоб уникнути випадкових шумів у коді.

При цьому вихід автокодувальника має бути знешумлений і, отже, відрізняється від входу. Ми можемо побачити, як це виглядатиме, використовуючи популярний набір даних MNIST, як показано на зображенні нижче:

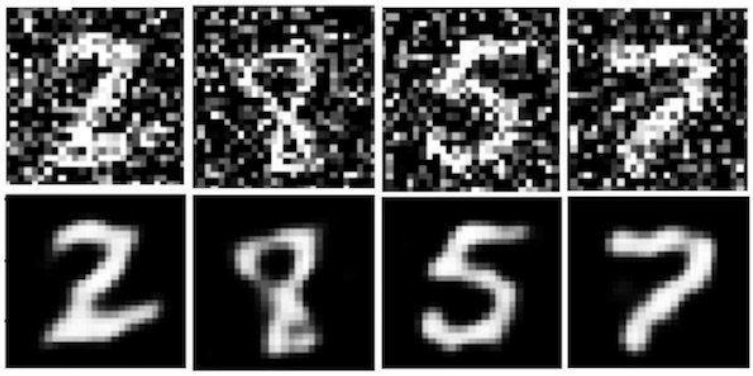

Рисунок 2.25 – Приклад знешумлення за допомогою автокодерів

Ці типи автокодерів можна використовувати для виділення функцій і усунення шумів у даних.

розріджений autoencoder цей тип автокодувальника явно забороняє використання прихованих підключень вузлів. Це впорядковує модель, утримуючи її від переобладнання даних. Цей «штраф за розрідженість» додається до втрат при реконструкції, щоб отримати функцію глобальних втрат. Крім того, можна просто видалити певну кількість підключень у прихованому шарі.

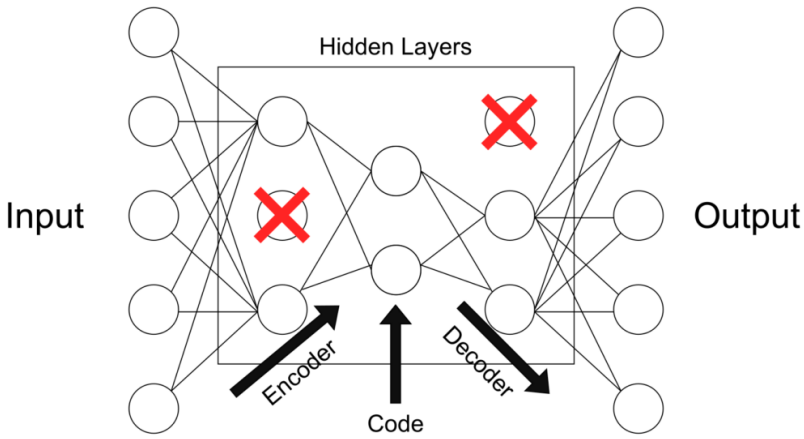

Рисунок 2.26 – Видалення певного кількістю підключень у прихованому шарі

Це скоріше метод регулярізації, який можна використовувати з широким набором типів кодерів. Застосування можуть відрізнятися.

Глибинні автокодери складаються з двох симетричних мереж глибокого переконання. Ця структура схожа на представлення «загальної структури» автокодувальника з використанням вузлів і з'єднань, знайдених вище. Ці дзеркальні компоненти можна описати як дві обмежені машини Больцмана, що діють як кодери та декодери:

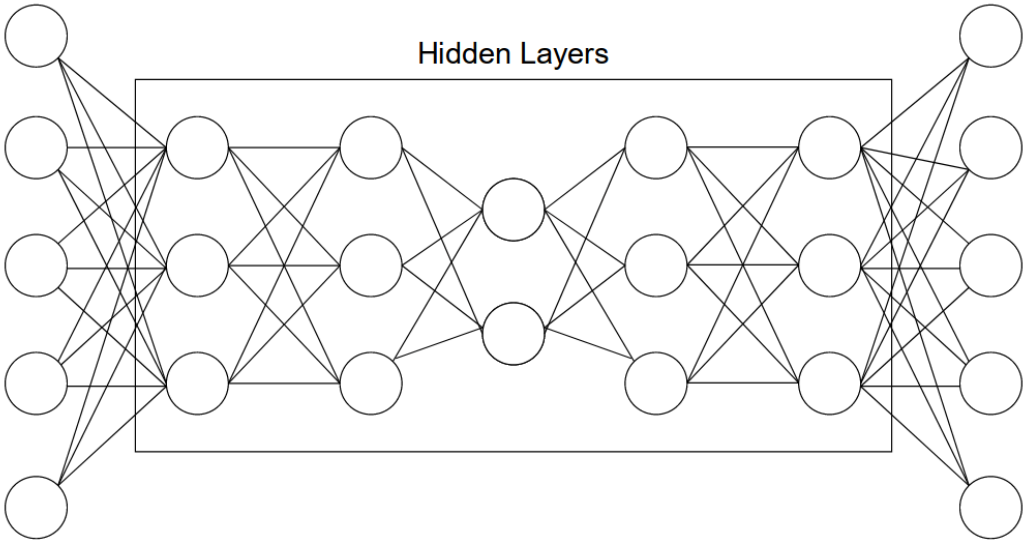

Рисунок 2.27 – Схема глибокого autoencoder

Автокодери цього типу використовуються для широкого спектру цілей, таких як виділення ознак, зменшення розмірності та стиснення даних.

Неповний автокодер ці автокодери мають менші приховані розміри порівняно з вхідними. Це означає, що вони чудово фіксують лише найважливіші характеристики даних. Ці типи автокодерів зазвичай не потребують регулярізації, оскільки вони не націлені на відтворення виходів, подібних до вхідних, а радше покладаються на стадію стиснення для захоплення значущих особливостей у даних:

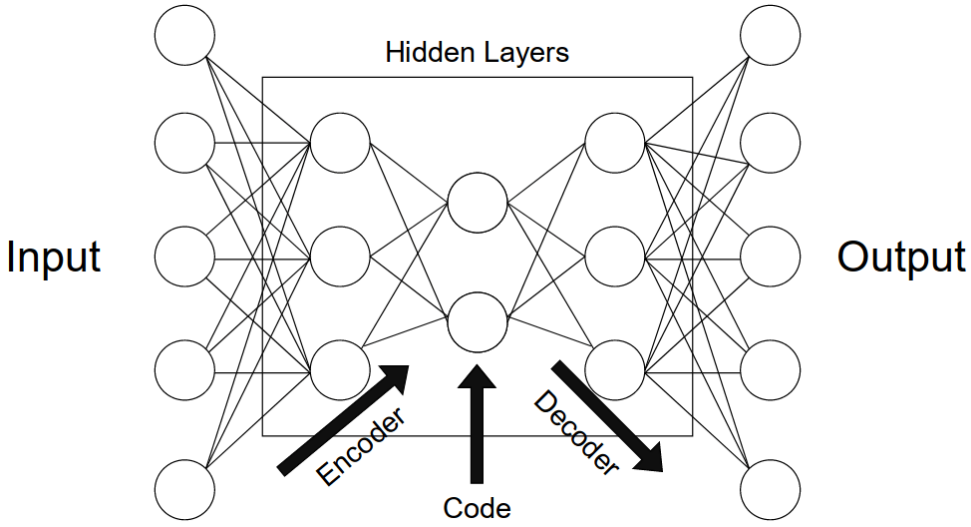

Рисунок 2.28 – Схема неповного autoencoder

Вилучення функцій є основним типом застосування для цього типу автокодерів.

Згорткові автокодери можуть використовувати суму різних сигналів для кодування та декодування. Найбільш поширеною версією цього є, ймовірно, згорточна модель U-Net. Ця модель, розроблена для програм біологічної обробки зображень, інтерпретуватиме результати різних фільтрів на зображенні, щоб класифікувати та остаточно сегментувати дані зображення. Подібні згорточні моделі використовуються сьогодні для сегментації зображень. Ці автокодери знаходять застосування в комп’ютерному зорі та розпізнаванні образів.

Варіаційні автокодери або VAE припускають кодування даних як розподіл на відміну від окремих точок у просторі. Це спосіб мати більш регулярний латентний простір, який ми можемо краще використовувати для створення нових даних. Отже, VAE називають генеративними моделями. Оригінальна стаття «Автоматичне кодування варіаційних байєсів», опублікована Дідеріком П. Кінгмою та Максом Веллінгом, детально описує повне функціонування цих конкретних моделей. Процес досягнення цього прихованого розподілу простору робить процес навчання дещо іншим. По-перше, замість відображення екземпляра як точки в просторі, він відображається як центр нормального розподілу. Далі ми

беремо точку з цього розподілу, щоб декодувати та обчислювати помилку реконструкції, яка буде розповсюджена в мережі.

Висновки до розділу 2

У даному розділі вивчалась проблематика стиснення даних, яка включає в себе розгляд методів та моделей для обох типів стиснення - без втрат та з втратами. Розділ розпочався з огляду методів безвтратного стиснення, де детально проаналізовані алгоритми, такі як Алгоритм Хаффмана, JBIG та Lossless JPEG.

Далі було досліджено методи стиснення даних з втратами, зосереджуючись на популярних підходах, таких як JPEG (Joint Photographic Experts Group) та Алгоритм фрактального стиснення. Кожен з цих методів має свої переваги та обмеження, які важливо розуміти при виборі конкретного методу для конкретного завдання.

Надалі у розділі досліджено використання методів штучного інтелекту для завдання стиснення даних. Аналізовані та порівнювані моделі, такі як згорткові нейронні мережі (CNN), автоенкодери та рекурентні нейронні мережі, які входять у різноманітні аспекти обробки та стискання даних

3 ДОСЛІДЖЕННЯ НЕЙРОМЕРЕЖЕВИХ МОДЕЛЕЙ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧ ЗГОРТКИ

В даному розділі проводиться дослідження та порівняння чотирьох моделей стиснення даних для обробки зображень. Зокрема, досліджуються дві моделі автоенкодерів та дві моделі згорткових нейронних мереж. Мета цього дослідження - визначити ефективність кожної моделі у стисненні та відновленні зображень, а також визначити найкращу модель для конкретної задачі стиснення зображень.

Кожна модель буде проаналізована з точки зору її можливостей у стисканні зображень, зокрема її швидкості, якості відновлення та обсягу стиснення. Крім того, в розділі буде визначено програмну реалізацію кожної моделі, що дозволить отримати більше інформації про їх функціонування та реалізацію на практиці.

На основі отриманих результатів буде здійснено порівняльний аналіз моделей та обрано найкращу для подальшого використання у програмі стиснення зображень.

3.1 автоенкодер для зображень 28 на 28 пікселів

У погоні за ефективними методами стиснення зображень без втрати інформації, дослідницькі команди з усього світу звертають увагу на моделі автоенкодерів. У цьому підрозділі ми ретельно вивчаємо та аналізуємо модель автоенкодера, спрямовану на стиснення зображень розміром 28 на 28 пікселів. Наша мета полягає в розумінні його потенційних можливостей, оптимізації та застосуванні для вирішення реальних завдань у сфері комп’ютерного зору та обробки зображень.

```
# Dimension of the encoded representation
encoding_dim = 49

# Encoder
input_img = Input(shape=(28, 28, 1))
flat_img = Flatten()(input_img)
x = Dense(encoding_dim*3, activation='relu')(flat_img)
x = Dense(encoding_dim*2, activation='relu')(x)
encoded = Dense(encoding_dim, activation='linear')(x)

# Decoder
input_encoded = Input(shape=(encoding_dim,))
x = Dense(encoding_dim*2, activation='relu')(input_encoded)
x = Dense(encoding_dim*3, activation='relu')(x)
flat_decoded = Dense(28*28, activation='sigmoid')(x)
decoded = Reshape((28, 28, 1))(flat_decoded)

# Models
encoder = Model(input_img, encoded, name="encoder")
decoder = Model(input_encoded, decoded, name="decoder")
autoencoder = Model(input_img, decoder(encoder(input_img)), name="autoencoder")
return encoder, decoder, autoencoder
```

Рисунок 3.1 – Скріншот коду моделі автоекнодера 28 на 28 пікселів

Архітектура моделі, яку буде використано для автокодування, базується на глибокій зворотній нейронній мережі. Ось опис архітектури цієї моделі:

— вхідний шар: вхідним шаром моделі є зображення розміром 28x28 пікселів у відтінках сірого (з одним каналом, оскільки це чорно-білі зображення). Цей шар використовується для передачі вхідних зображень до автокодера;

— енкодер: енкодер складається з кількох повно зв'язних шарів. Після вхідного шару, зображення згладжується у вектор за допомогою шару Flatten(). Потім вектор проходить через декілька Dense шарів з функцією активації ReLU. Останній Dense шар має кількість нейронів, визначену розміром кодованого представлення, з використанням лінійної активації;

— кодоване представлення: цей шар представляє собою кодований вектор, який є стиснутим представленням вхідного зображення. Розмірність цього кодованого представлення визначається параметром encoding_dim, який може бути встановлений вами;

– декодер: Декодер також складається з кількох повно зв'язних шарів. Він приймає кодоване представлення як вхід та пропускає його через Dense шари з активацією ReLU, щоб розгорнути його назад до вихідного розміру. Останній Dense шар використовує сигмоїдну активацію, щоб отримати значення пікселів у діапазоні [0, 1];

– вихідний шар: вихідним шаром є розгорнуті зображення розміром 28x28 пікселів, які є відновленими версіями вхідних зображень.

Ця архітектура моделі представляє собою класичний автоенкодер, який використовується для автоматичного вивчення подань зображень та їхнього подальшого відновлення. За допомогою такої моделі можна стиснути інформацію, що міститься у зображеннях до більш компактного кодованого представлення, а потім відновити їхню вихідну форму з цього представлення.

Навчання моделі. Зосередимося на навчанні моделі автоенкодера для стиснення зображень розміром 28 на 28 пікселів. В загалом для навчання використовується функція fit, на Рис. 3.2 зображено виклик функції цієї функції.

```
# Train the autoencoder
d_autoencoder.fit(x_train, x_train,
                  epochs=10,
                  batch_size=256,
                  shuffle=True,
                  validation_data=(x_test, x_test),
                  callbacks=[history])
```

Рисунок 3.2 – Скріншот коду виклику функції навчання fit

Код d_autoencoder.fit() використовується для навчання моделі автокодувальника d_autoencoder на навчальних даних x_train, які є вхідними та вихідними зображеннями. Ось розшифровка параметрів цього виклику:

– x_train, x_train: це навчальні дані, які складаються з набору вхідних зображень (перший аргумент) та відповідних вихідних зображень (другий

аргумент). У випадку автокодувальника, ми хочемо, щоб вихідні дані були такими ж, як і вхідні дані, оскільки ми навчаємо модель відтворювати вхід на виході;

– epochs=10: це кількість епох навчання, тобто кількість разів, коли весь навчальний набір даних буде пройдено через мережу. У цьому випадку навчання триватиме 10 епох;

– batch_size=256: цей параметр визначає кількість зразків, які будуть використані для оновлення ваг моделі перед кожним кроком градієнтного спуску. У цьому випадку мережа буде оновлюватися після обробки кожного пакету з 256 зображень;

– shuffle=True: цей параметр вказує на те, що дані будуть перемішані перед кожною епохою. Це допомагає уникнути впливу порядку даних на процес навчання та покращує здатність моделі узагальнювати;

– validation_data=(x_test, x_test): цей параметр визначає дані для перевірки на кожній епохі навчання. В даному випадку використовуються дані перевірки x_test, які допомагають оцінити продуктивність моделі на незалежному наборі даних. У цьому випадку x_test також використовується як вхідні та вихідні дані для оцінки втрат під час навчання.

Цей виклик fit() запускає процес навчання моделі автокодувальника за допомогою навчальних даних, що наведені, з використанням параметрів, які ви вказали. Після завершення навчання модель буде навчена реконструювати вхідні зображення на виході.

Дата сет використовується MNIST, цей набір є одним з найпопулярніших наборів даних у сфері машинного навчання та глибокого навчання. Він містить 70 000 рукописних зображень цифр розміром 28x28 пікселів, які складаються з чисел від 0 до 9. Цей дата сет зазвичай використовується для навчання та перевірки різних моделей класифікації зображень.

Для завантаження дата сету MNIST за допомогою фреймворку Keras можна використовувати функцію mnist.load_data() з модуля keras.datasets.

```
Epoch 6/10
235/235 [==============================] - 3s 12ms/step - loss: 0.0991 - val_loss: 0.0959
Epoch 7/10
235/235 [==============================] - 3s 12ms/step - loss: 0.0959 - val_loss: 0.0933
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.0936 - val_loss: 0.0914
Epoch 9/10
235/235 [==============================] - 3s 12ms/step - loss: 0.0916 - val_loss: 0.0898
Epoch 10/10
235/235 [==============================] - 3s 12ms/step - loss: 0.0900 - val_loss: 0.0884
```

Рисунок 3.3 – Процес навчання моделі автоенкодера

Як можна побачити похибка в останніх епохах дорівнює 0.09 це означає що точність моделі дорівнює (1-0.09) * 100 % = 91% це дуже не поганий результат, враховуючи що автоенкодеру дуже тяжко запам’ятовувати деталі.

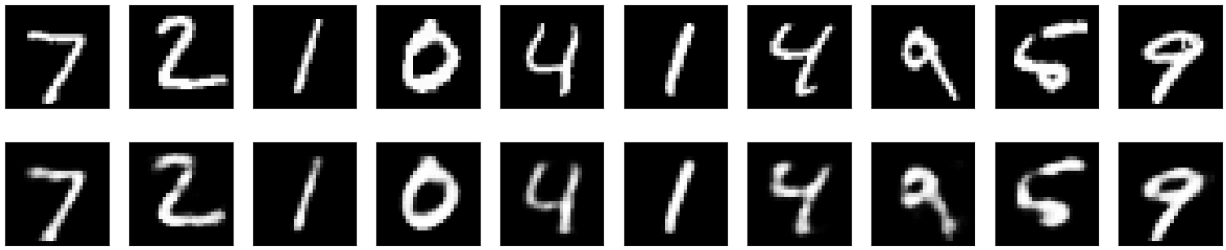

Рисунок 3.4 – Результат навчання моделі автоенкодера

Як ми можемо побачити автоенкодер за 10 епох стиснув зображення та відновив його фактично без змін.

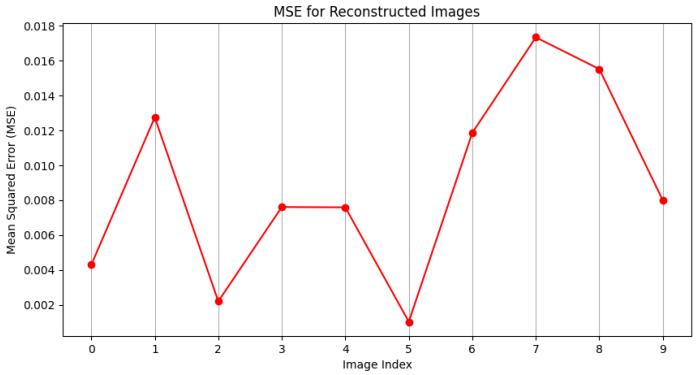

Рисунок 3.5 – Графік MSE для першої моделі автоекодера

За графіком можна побачити чим менше значення MSE, тим краще реконструйоване зображення. У цьому випадку, зображення 6 має найменше значення MSE, що означає, що його відновлення є найбільш точним. Зображення 8 має найбільше значення MSE, що означає найбільше відхилення між оригіналом та відновленою версією.

Рисунок 3.6 – Графік MAE для першої моделі автоекодера

Як можна побачити значення MAE відрізняються на різних зображеннях, що вказує на різні рівні точності реконструкції. Нижчі значення MAE вказують на кращу якість реконструкції, оскільки вони передбачають менші середні відмінності між вихідним і реконструйованим зображеннями. І навпаки, вищі значення MAE свідчать про більші розбіжності між вихідним і реконструйованим зображеннями.

З наданих значень можна побачити, що зображення 6 має найнижчий MAE (0,00752), що вказує на високий рівень точності реконструкції для цього конкретного зображення. З іншого боку, зображення 8 має найвищий MAE (0,04530), що вказує на більшу середню різницю між вихідним і реконструйованим зображенням для цього випадку.

Загалом, аналіз значень MAE дає змогу зрозуміти ефективність процесу реконструкції, причому нижчі значення MAE відповідають кращій якості реконструкції.

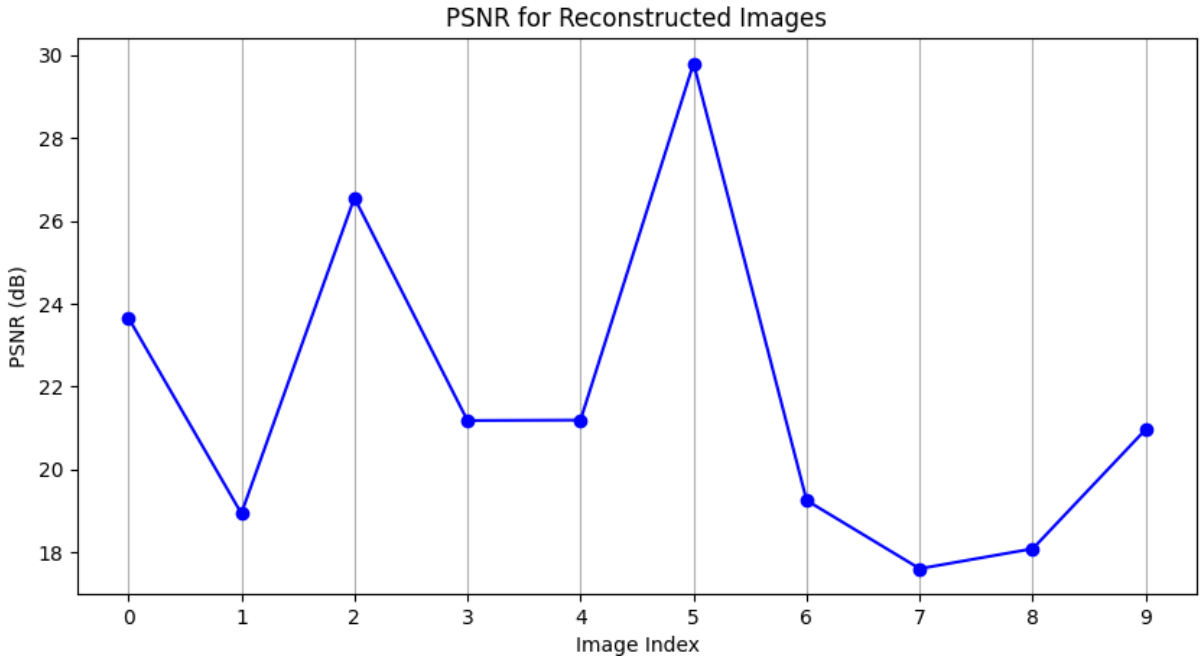

Рисунок 3.7 – Графік PSNR для першої моделі автоекодера

Таким чином, можна помітити, що вищі значення PSNR (наприклад, Image 6 з PSNR = 29.80 dB) вказують на вищу якість відновлення зображення, тоді як нижчі значення PSNR (наприклад, Image 8 з PSNR = 17.61 dB) вказують на гіршу якість відновлення.

Вищі значення PSNR вказують на більшу схожість між оригінальним та відновленим зображеннями. Таким чином, вищі значення PSNR вказують на кращу якість відновлення зображення, оскільки вони вказують на меншу втрату інформації при відновленні.

З іншого боку, нижчі значення PSNR вказують на більшу різницю між оригінальним та відновленим зображеннями. Це може вказувати на велику кількість шуму або втрату деталей під час відновлення, що призводить до гіршої якості відновлення.

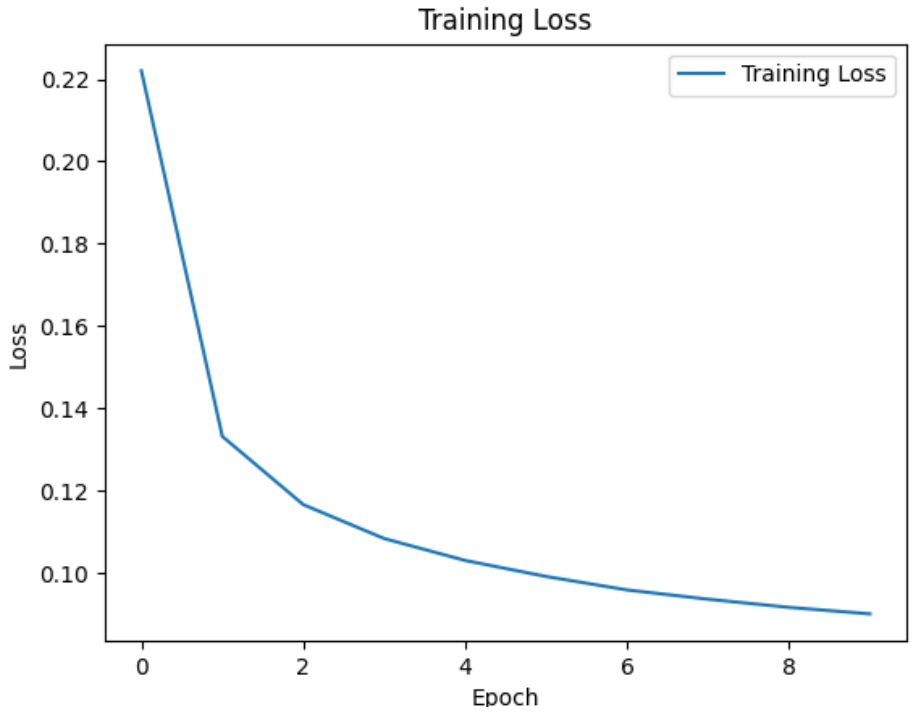

Рисунок 3.8 – Графік Loss під час навчання для першої моделі автоекодера

Даний графік показує значення втрат під час тренування та перевірки моделі під час кожної епохи. Можна побачити, що втрати зменшуються з кожною епохою. Однак важливо враховувати, що неможливо точно визначити, наскільки модель ефективна лише за значеннями втрат.

У цьому випадку втрати зменшилися з 0.2220 на тренувальному наборі на першій епохі до 0.0900 на десятій епохі. Це свідчить про те, що модель поступово навчається та покращується з кожною епохою.

3.2 Згорткова нейрона мережа для зображень 28 на 28 пікселів

У цьому підрозділі ми зосереджуємося на дослідженні моделей згорткової нейронної мережі, спрямованих на стиснення зображень розміром 28 на 28 пікселів. Наша робота включає аналіз різних архітектур та методів оптимізації, щоб досягти максимальної ефективності у зменшенні розміру зображень без втрати інформації. Наші дослідження спрямовані на вдосконалення та розширення

застосування згорткових нейронних мереж у завданнях компресії та обробки зображень.

```
input_img = Input(shape=(28, 28, 1))
x = Conv2D(128, (7, 7), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (2, 2), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x_coded = Conv2D(1, (7, 7), activation='relu', padding='same')(x)
x_input = Input(shape=(7, 7, 1))
x = Conv2D(32, (7, 7), activation='relu', padding='same')(x_input)
x = UpSampling2D((2, 2))(x)
x = Conv2D(128, (2, 2), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x_decoded = Conv2D(1, (7, 7), activation='sigmoid', padding='same')(x)
coder = Model(input_img, x_coded, name="coder")
decoder = Model(x_input, x_decoded, name="decoder")
CNN = Model(input_img, decoder(coder(input_img)), name="CNN")
```

Рисунок 3.9 – Скріншот коду першої моделі CNN

Модель складається з двох основних компонентів: кодера і декодера, які разом утворюють згорткову нейронну мережу.

– кодер: починається з вхідного шару (input_img) розміром 28x28 з одним каналом. Також застосовується згортковий шар (Conv2D) з 128 фільтрами та розміром ядра (7, 7) з функцією активації ReLU і однаковим заповненням. Після цього використовується шар пулінгу (MaxPooling2D) з розміром ядра (2, 2) для зменшення розміру зображення. Знову застосовується згортковий шар з 32 фільтрами і розміром ядра (2, 2) з активацією ReLU. Після цього застосовується фінальний згортковий шар (Conv2D) з одним фільтром та розміром ядра (7, 7) з активацією ReLU;

– декодер: починається з вхідного шару (x_input) розміром (7, 7) з одним каналом. Застосовується згортковий шар з 32 фільтрами та розміром ядра (7, 7) з активацією ReLU та однаковим заповненням. Після цього застосовується шар звичайного випрямлення (UpSampling2D) з множником (2, 2), щоб збільшити

розмір зображення. Застосовується ще один згортковий шар з 128 фільтрами та розміром ядра (2, 2) з активацією ReLU та однаковим заповненням. Після цього використовується ще один шар випрямлення (UpSampling2D) з множником (2, 2). Фінальний згортковий шар має один фільтр та розмір ядра (7, 7) з сигмоїдальною активацією, яка призначена для відновлення значень пікселів у діапазоні [0, 1];

– модель згорткової нейронної мережі (CNN): комбінує кодер і декодер, використовуючи їх як функції для обробки вхідних зображень і відновлення оригінальних зображень. Вхідними даними для цієї моделі є вхідне зображення (input_img), а вихідними - відновлені зображення, які отримані після проходження через кодер і декодер.

Ця архітектура моделі використовує згорткові шари для витягнення важливих ознак зображень та їхнього подальшого відновлення за допомогою декодера.

Навчання моделі. У вікові цифрових технологій, де обробка та передача великого обсягу даних стають нормою, виникає необхідність у розвитку ефективних методів стиснення зображень без втрати якості. Наступний кроком зосереджуємося на навчанні моделей згорткових нейронних мереж для оптимізації процесу стиснення зображень розміром 28 на 28 пікселів.

```
CNN.fit(x_train, x_train,
        epochs=10,
        batch_size=256,
        shuffle=True,
        validation_data=(x_test, x_test),
        callbacks=[history])
```

Рисунок 3.10 – Скріншот коду виклику функції навчання fit

Цей фрагмент коду відповідає за навчання моделі згорткової нейронної мережі на наборі даних. Розглянемо детальніше кожен аргумент методу fit:

– x_train, x_train: це вхідні та вихідні дані, які використовуються для навчання моделі. Обидва аргументи містять зображення, де x_train є вхідними зображеннями, а x_train - вихідними (цільовими) зображеннями. Модель навчатиметься відтворювати вихідні зображення з вхідних;

– epochs=10: цей аргумент визначає кількість повторень навчального процесу на всьому наборі даних. У цьому випадку модель буде навчатися протягом 10 епох;

– batch_size=256: розмір пакету визначає кількість зразків, які використовуються для одного оновлення градієнту. У даному випадку, модель буде оновлювати свої ваги після кожних 256 зразків;

– shuffle=True: цей аргумент вказує на те, що дані будуть перемішані перед кожною епохою. Це допомагає уникнути навчання моделі на однотипних або однакових пакетах даних, що може призвести до кращої узагальнюючої здатності моделі;

– validation_data=(x_test, x_test): цей аргумент визначає дані, на яких модель буде оцінюватися під час навчання. Тут x_test є вхідними зображеннями для перевірки, а x_test - їхніми відповідними вихідними зображеннями. Валідаційні дані допомагають контролювати процес навчання та визначати, чи відбувається перенавчання моделі.

Отже, код CNN.fit(x_train, x_train, epochs=10, batch_size=256, shuffle=True, validation_data=(x_test, x_test)) навчає згорткову нейронну мережу на наборі даних протягом 10 епох, використовуючи пакети розміром 256 зразків для оновлення градієнту. Дані перемішуються перед кожною епохою, і процес навчання оцінюється за допомогою валідаційних даних x_test. Навчальна вибірка використовується така ж як і в попередньому моделі, MNIST.

```
Epoch 6/10
235/235 [==============================] - 84s 357ms/step - loss: 0.0805 - val_loss: 0.0793
Epoch 7/10
235/235 [==============================] - 96s 410ms/step - loss: 0.0793 - val_loss: 0.0777
Epoch 8/10
235/235 [==============================] - 91s 387ms/step - loss: 0.0784 - val_loss: 0.0767
Epoch 9/10
235/235 [==============================] - 86s 367ms/step - loss: 0.0774 - val_loss: 0.0767
Epoch 10/10
235/235 [==============================] - 90s 385ms/step - loss: 0.0768 - val_loss: 0.0756
```

Рисунок 3.11 – Процес навчання моделі CNN

Як можна побачити похибка в останніх епохах дорівнює 0.07 це означає що точність моделі дорівнює (1-0.07) * 100 % = 93% це дуже хороший результат. Згортковій нейронній мережі набагато простіше запам'ятовувати деталі.

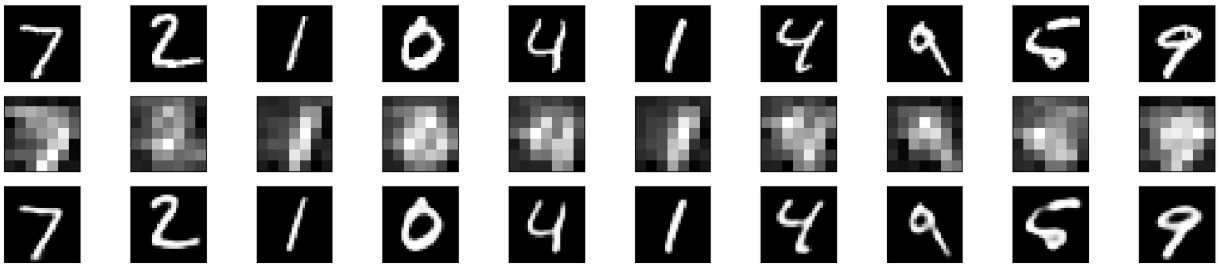

Рисунок 3.12 – Результат навчання моделі CNN

Як можна побачити стиснення за допомогою згорткової нейронної мережі стиснення дало нам набагато краще результат як і по стисненню так і по відновленню даних.

Рисунок 3.13 – Графік MSE для першої моделі CNN

Аналізуючи цей графік, можна побачити, що Image 6 є найменшим значенням MSE, тобто це зображення має найменше відхилення між оригінальним та відновленим зображенням. З іншого боку, для Images 9 та 10 маємо найбільші значення MSE, що вказує на більше відхилення та, відповідно, гіршу якість відновлення цих зображень.

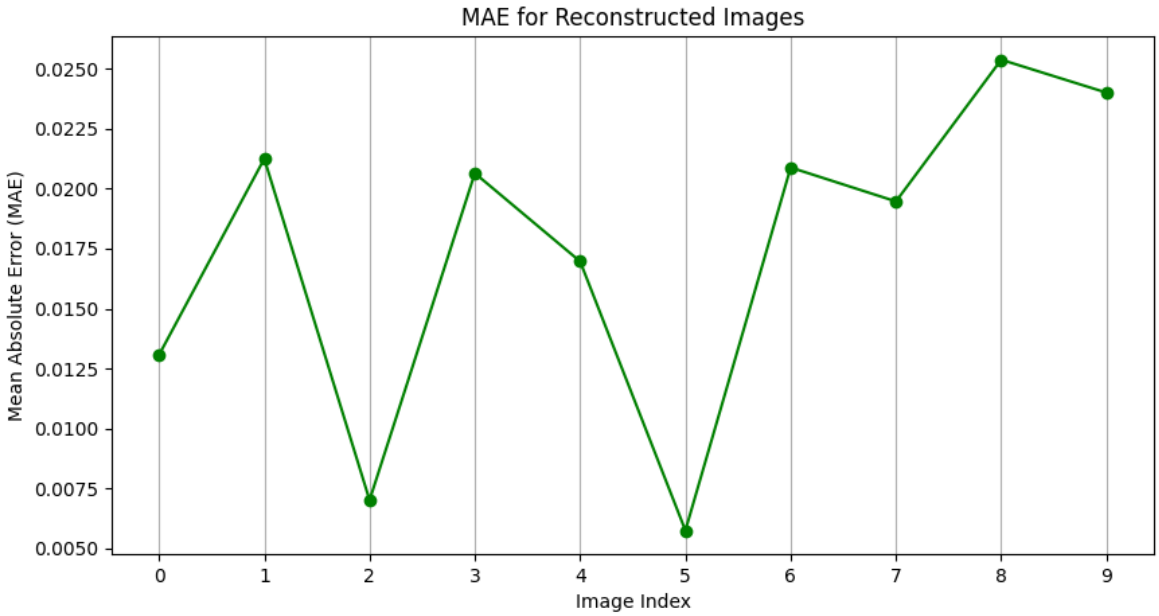

Рисунок 3.14 – Графік MAE для першої моделі CNN

Можна побачити, що значення MAE для кожного зображення варіюються. Нижчі значення MAE вказують на те, що відновлені зображення досить точно відтворюють оригінал, оскільки вони мають менші абсолютні відхилення від оригінальних зображень. Наприклад, Image 6 з MAE = 0.005746084731072187 це дуже низьке значення помилки, що вказує на дуже точне відновлення зображення. А також, Image 9 з MAE = 0.02538241818547249 має вище значення помилки, що може вказувати на менш точне відновлення зображення.

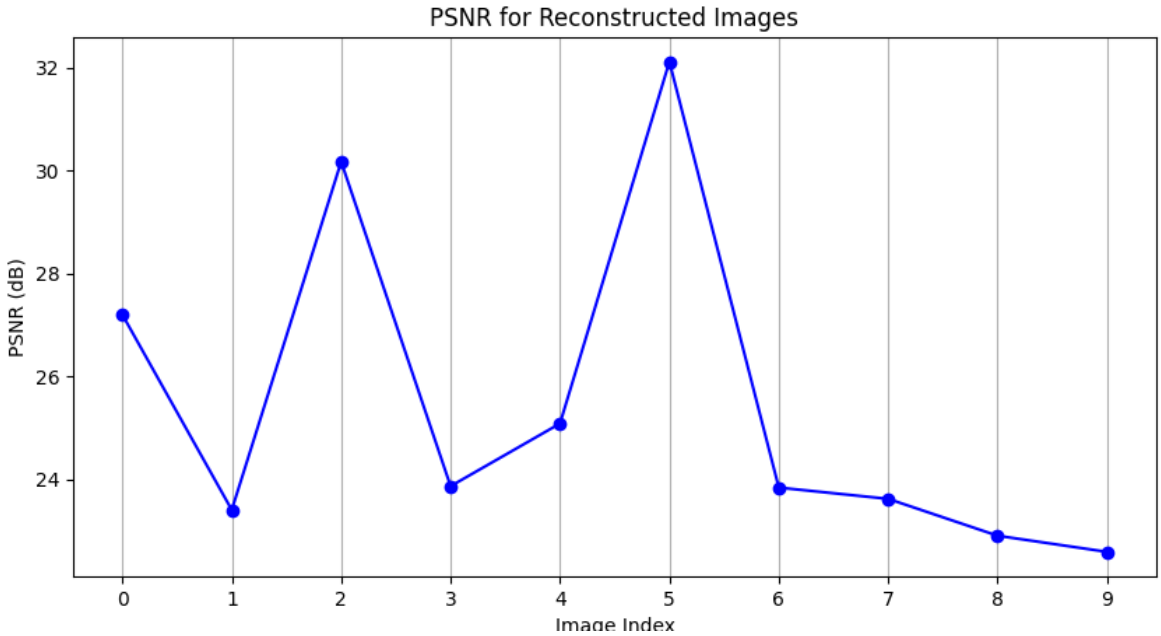

Рисунок 3.15 – Графік PSNR для першої моделі CNN

Як можна побачити за графіком значення для 10 зображень сильно змінюються. Наприклад Image 3 та Image 6 мають найвищі значення PSNR: 30.18 dB та 32.12 dB відповідно. Це свідчить про те, що вони мають високу якість відновлення та малу втрату інформації під час компресії чи обробки. Також image 9 та Image 10 мають найнижчі значення PSNR: 22.90 dB та 22.59 dB відповідно це означає, що ці зображення мають гіршу якість відновлення та більшу втрату інформації, можливо, через шум чи недостатність обробки. Image 2, Image 4, та Image 7 також мають досить низькі значення PSNR (від 23.40 dB до 23.86 dB), що

свідчить про те, що вони можуть потребувати додаткового налаштування чи оптимізації моделі для покращення якості відновлення.

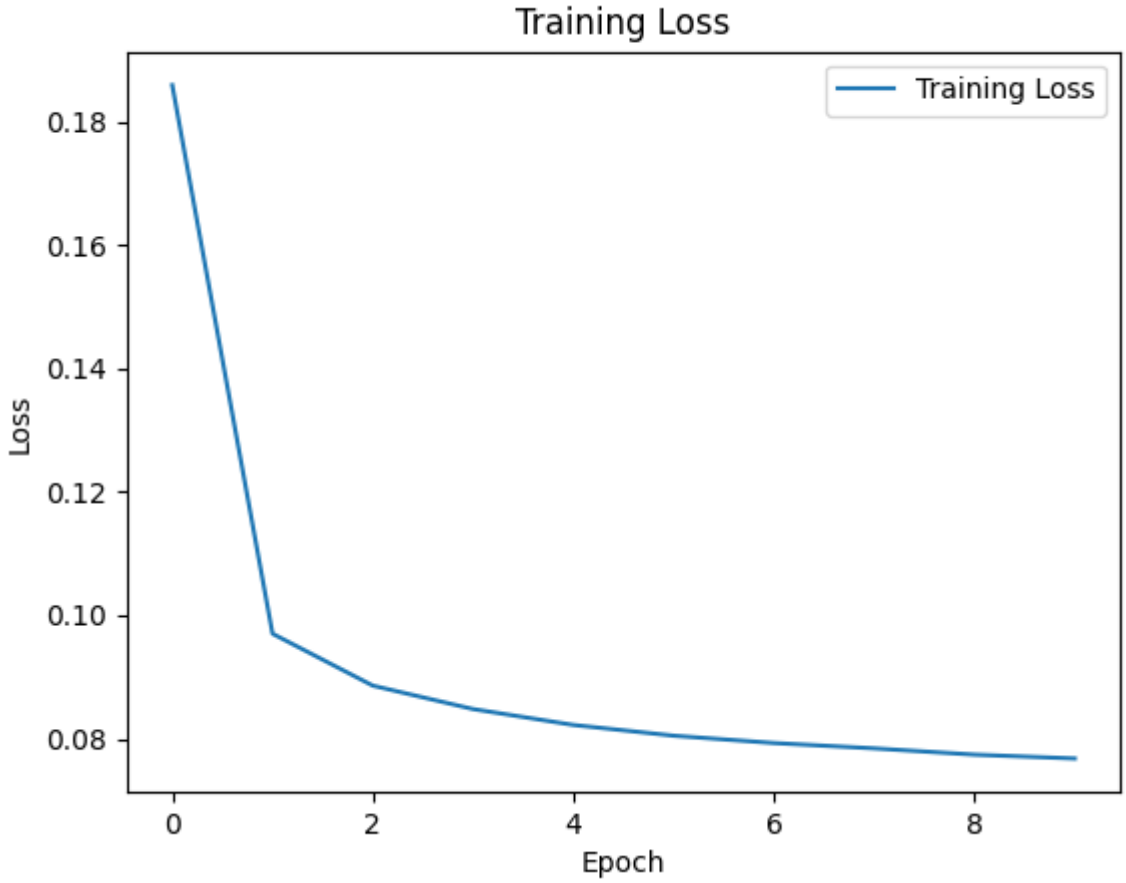

Рисунок 3.16 – Графік Loss під час навчання для першої моделі CNN

Аналізуючи втрати (loss) під час навчання моделі, які були зображені на графіку можна зазначити що початкове значення втрат на епохі було досить великим (приблизно 0.1860), що вказує на значні різниці між оригінальними та відновленими зображеннями. Протягом перших кількох епох втрати зменшувалися (0.0970 на другій епохі, 0.0886 на третій, 0.0848 на четвертій), що свідчить про покращення якості відновлення зображення з кожною наступною епохою.

У наступних епохах втрати надалі зменшувалися, хоча темп зменшення сповільнився (0.0822 на п'ятій епохі, 0.0805 на шостій).

На останній епохі значення втрат були найменшими (0.0768), що вказує на покращення якості відновлення зображення після завершення навчання моделі.

Отже, можна зробити висновок, що модель успішно навчалася і показала покращення в якості відновлення зображень з кожною наступною епохою навчання.

3.3 автоенкодер для зображень 256 на 256 пікселів

У цьому підрозділі ми зосереджуємося на дослідженні моделі автоенкодера для стиснення зображень розміром 256 на 256 пікселів. Наша мета полягає в розробці ефективного та потужного алгоритму, яка здатна не лише ефективно зменшувати розмір файлів, але й зберігати важливі візуальні аспекти зображень.

```
encoding_dim = 49

# Encoder
input_img = Input(shape=(256, 256, 1))
x = Flatten()(input_img)
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='linear')(x)

# Decoder
input_encoded = Input(shape=(encoding_dim,))
x = Dense(128, activation='relu')(input_encoded)
x = Dense(256, activation='relu')(x)
x = Dense(512, activation='relu')(x)
flat_decoded = Dense(256*256, activation='sigmoid')(x)
decoded = Reshape((256, 256, 1))(flat_decoded)

# Models
encoder = Model(input_img, encoded, name="encoder")
decoder = Model(input_encoded, decoded, name="decoder")
autoencoder = Model(input_img, decoder(encoder(input_img)), name="autoencoder")
```

Рисунок 3.17 – Скріншот коду другої моделі автоекнодера 256 на 256 пікселів

Дана архітектура моделі, є класичним прикладом простого автокодувальника. Проведемо опис цієї моделі детально:

– вхідний шар: приймає зображення розміром 256x256 пікселів у відтінках сірого (з одним каналом, оскільки це чорно-білі зображення).

– енкодер: енкодер починається зі згладжування зображення у вектор за допомогою шару Flatten(). Потім вектор проходить через декілька повнозв'язних шарів з функцією активації ReLU. Останній повнозв'язний шар має кількість нейронів, визначену параметром encoding_dim, і використовує лінійну активацію для отримання кодованого представлення.

– кодоване представлення: цей шар представляє собою кодований вектор, який є стиснутим представленням вхідного зображення. Розмірність цього кодованого представлення визначається параметром encoding_dim.

– декодер: декодер починається з вхідного кодованого представлення. він проходить через декілька повнозв'язних шарів з функцією активації relu, щоб розгорнути його назад до вихідного розміру. Останній повнозв'язний шар використовує сигмоїдальну активацію, щоб отримати значення пікселів у діапазоні [0, 1];

– вихідний шар: представляє собою відновлене зображення розміром 256x256 пікселів.

Ця архітектура автокодувальника може бути використана для стиснення інформації у великих зображеннях і подальшого відновлення їхнього вихідного вигляду.

Навчання моделі. Перейдемо до навчання моделі автоекодера для ефективного стиснення зображень розміром 256 на 256 пікселів, ми прагнемо досягти оптимального балансу між збереженням якості зображення та ефективним його стисненням, що відкриває широкі можливості для застосування в різноманітних сферах, від зберігання фотографій до передачі даних у реальному часі.

```
# Define the directory containing the images
data_dir = 'data2'

# Get the list of image files
image_files = [file for file in os.listdir(data_dir) if file.endswith('.jpg')]

# Load and format the images with color conversion
images = [rgb2gray(img_to_array(load_img(os.path.join(data_dir, file), target_size=(256, 256, 3)))) for file in image_files]
images = np.array(images)

# Normalize pixel values to the range [0, 1]
images = images.astype('float32') / 255.0

# Split into training and test sets
x_train, x_test = train_test_split(images, test_size=0.3, random_state=42)

# Reshape images if necessary to match expected format
x_train = np.reshape(x_train, (len(x_train), 256, 256, 1))  # Single channel for grayscale
x_test = np.reshape(x_test, (len(x_test), 256, 256, 1))
```

Рисунок 3.18 – Скріншот коду підключення дата сету

Так як цей дата сет був завантажений фізично на жорсткий диск. Потрібно перед навчанням підготувати дані, тобто зчитати та розділити зображення на навчальну та перевірочну вибірки в нашому випадку x_train та x_test, як це було зроблено можна побачити на малюнку вище.

```
d_autoencoder.fit(x_train, x_train,
                  epochs=50,
                  batch_size=10,
                  shuffle=True,
                  validation_data=(x_test, x_test),
                  callbacks=[history])
```

Рисунок 3.19 – Скріншот коду виклику функції навчання fit

Параметри навчання фактично не змінні від попередніх моделей єдине що потрібно було змінити кількість епох та batch_size

```
70/70 [==============================] - 33s 475ms/step - loss: 0.0273 - val_loss: 0.0385
Epoch 43/50
70/70 [==============================] - 33s 468ms/step - loss: 0.0271 - val_loss: 0.0385
Epoch 44/50
70/70 [==============================] - 32s 461ms/step - loss: 0.0269 - val_loss: 0.0383
Epoch 45/50
70/70 [==============================] - 32s 457ms/step - loss: 0.0267 - val_loss: 0.0383
Epoch 46/50
70/70 [==============================] - 32s 460ms/step - loss: 0.0264 - val_loss: 0.0384
Epoch 47/50
70/70 [==============================] - 32s 458ms/step - loss: 0.0263 - val_loss: 0.0382
Epoch 48/50
70/70 [==============================] - 34s 492ms/step - loss: 0.0261 - val_loss: 0.0382
Epoch 49/50
70/70 [==============================] - 32s 457ms/step - loss: 0.0261 - val_loss: 0.0385
Epoch 50/50
70/70 [==============================] - 34s 483ms/step - loss: 0.0260 - val_loss: 0.0388
```

Рисунок 3.20 – Процес навчання другої моделі автоенкодера

Як можна побачити похибка в останніх епохах дорівнює 0.02 це означає що точність моделі дорівнює (1-0.02) * 100 % = 98% це дуже хороший результат для автоенкодера, збільшувати кількість епох немає сенсу враховуючи те похибка мінімальна.

Рисунок 3.21 – Результат навчання другої моделі автоенкодера

За результатом можна побачити що хоча і похибка мінімальна але автоенкодеру тяжко відновити зображення після стиску.

Рисунок 3.22 – Графік MSE для другої моделі автоекодера

У наведеному вище графіку MSE для кожного зображення, низькі значення MSE (наприклад, менше 0.005) вказують на те, що модель добре відтворює вхідні дані. Високі значення MSE (більше 0.005) можуть вказувати на те, що відновлені зображення значно відрізняються від оригіналів.

Загалом, низькі значення MSE свідчать про те, що модель ефективно відновлює зображення, тоді як високі значення можуть вказувати на потребу у поліпшенні моделі або у додатковому налаштуванні параметрів.

Рисунок 3.23 – Графік MAE для другої моделі автоекодера

Середня абсолютна помилка (MAE) вказує на середнє значення абсолютних відхилень між оригінальними та відновленими зображеннями. У даному випадку, значення MAE для кожного зображення коливається від 0.006 до 0.021. Чим менше значення MAE, тим ближче відновлене зображення до оригіналу, і нижче значення свідчить про кращу якість відновлення.

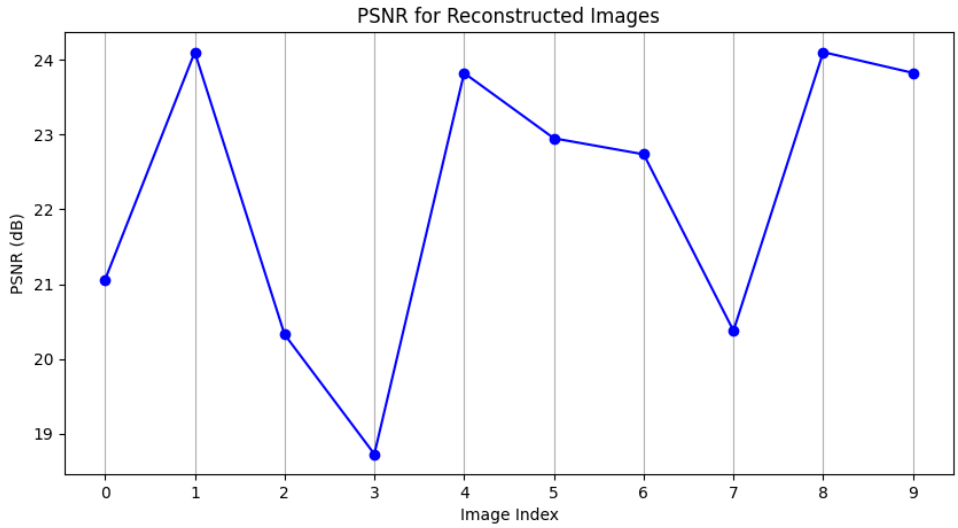

Рисунок 3.24 – Графік PSNR для другої моделі автоекодера

Як можна побачити на графіку PSNR (Пікове відношення сигнал-шум) він вимірює якість відновленого зображення порівняно з оригінальним зображенням. Чим вище значення PSNR, тим менше помилок при відновленні і, отже, якість відновленого зображення вважається кращою. На графіку PSNR розташовано в діапазоні від приблизно 18 до 24 децибел, що вказує на різний рівень якості відновлених зображень, де вищі значення PSNR вказують на кращу якість відновлення.

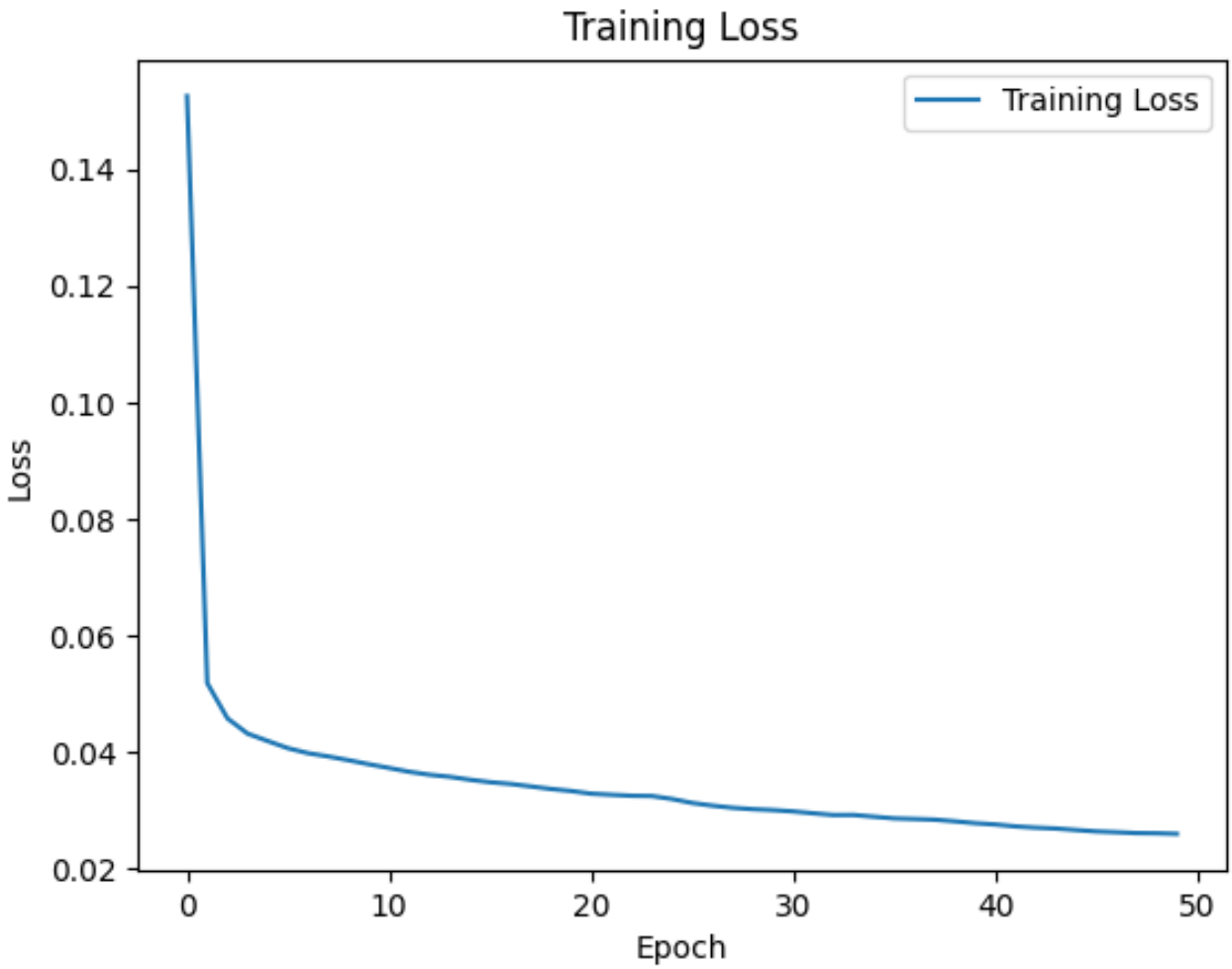

Рисунок 3.25 – Графік Loss під час навчання для другої моделі автоекодера

Аналізуючи графік втрат (loss) протягом навчання моделі, можна побачити, що на початку тренування втрати зменшуються досить швидко. Проте, коли модель наближається до кінцевих епох, зменшення втрат стає менш помітним. Це може свідчити про те, що модель навчилася до певного рівня, і подальше поліпшення може вимагати більшої складності моделі або оптимізації параметрів навчання. У цьому конкретному випадку, втрати на валідаційному наборі також зменшуються, що показує ефективність моделі на незалежних данних. Важливо слідкувати за втратами під час тренування моделі, оскільки вони відображають процес навчання і можуть вказати на необхідність змін у архітектурі моделі або параметрах навчання.

3.4 Згорткова нейрона мережа для зображень 256 на 256 пікселів

Зростаюча потреба в оптимізації обробки та передачі зображень викликає необхідність у вдосконаленні методів їх стиснення. У зв'язку з цим, в нашому підрозділі ми зосереджуємося на дослідженні та вдосконаленні моделей згорткових нейронних мереж для стиснення зображень розміром 256 на 256 пікселів. Наша мета полягає у розробці ефективних та оптимальних алгоритмів, які дозволять зберігати якість зображень при зменшенні їх розміру, забезпечуючи при цьому зменшення обсягу пам'яті та часу обробки.

```
input_img = Input(shape=(256, 256, 1))
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
coded = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x_decoded = Conv2D(64, (3, 3), activation='relu', padding='same')(coded)
x_decoded = UpSampling2D((2, 2))(x_decoded)
x_decoded = Conv2D(32, (3, 3), activation='relu', padding='same')(x_decoded)
x_decoded = UpSampling2D((2, 2))(x_decoded)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x_decoded)
coder = Model(input_img, coded, name="coder")
decoder = Model(coded, decoded, name="decoder")
CNN = Model(input_img, decoder(coder(input_img)), name="CNN")
```

Рисунок 3.26 – Скріншот коду другої моделі CNN 256 на 256 пікселів

Основна архітектура моделі базується на згорткових та пулінгових шарах, які використовуються для екстракції ознак зображень та їх подальшого відновлення. Проведемо опис архітектури більш детально:

— вхідний шар: вхідними даними є зображення розміром 256x256 пікселів з одним каналом. Цей шар використовується для передачі вхідних зображень до моделі;

— згорткові шари: модель використовує два згорткові шари з фільтрами розміром 3x3. Перший згортковий шар має 32 фільтри, а другий - 64. Після кожного згорткового шару використовується функція активації ReLU для нелинійності;

– пулінгові шари: після кожного згорткового шару використовується пулінговий шар MaxPooling2D з розміром пулінгу 2x2. Це допомагає зменшити розмір зображення та сконцентрувати важливі ознаки;

– кодований представлення: після згорткових та пулінгових шарів, отримане зображення проходить через ще один згортковий шар з 128 фільтрами. Результат цього шару представляє собою кодоване представлення зображення;

– декодер: починається зі згорткового шару з 64 фільтрами, який слугує початковим кроком у відновленні зображення. Потім використовуються шари UpSampling2D для збільшення розміру зображення. Після цього використовується додатковий згортковий шар з 32 фільтрами для зменшення кількості каналів. На виході моделі отримується відновлене зображення розміром 256x256 пікселів;

– вихідний шар: вихідним шаром моделі є реконструйоване зображення, яке відповідає оригінальному вхідному зображенню. Функція активації сигмоїдальна, щоб забезпечити значення пікселів у діапазоні [0, 1].

Ця архітектура моделі базується на принципах згорткових нейронних мереж, які ефективно використовуються для завдань обробки зображень.

Навчання моделі. Детально розберемося із процесом навчання моделі згорткової нейронної мережі для стиснення зображень розміром 256 на 256 пікселів.

```
CNN.fit(x_train, x_train,
                epochs=10,
                batch_size=10,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[history])
```

Рисунок 3.27 – Скріншот коду виклику функції навчання fit

Як можна побачити параметри для навчання є фактично не змінними як з минулого разу але модель навчаємо на іншому дата сеті під назвою Chinese MNIST він міститись 1000 зображень 256 на 256 пікселів китайських ієрогліфів, так як

зображень в стало менше batch_size було змінено на 10 для покращення навчання та меншого навантаження на оперативну пам’ять.

```
Epoch 5/10
70/70 [==============================] - 47s 672ms/step - loss: 0.0217 - val_loss: 0.0219
Epoch 6/10
70/70 [==============================] - 47s 667ms/step - loss: 0.0215 - val_loss: 0.0218
Epoch 7/10
70/70 [==============================] - 47s 673ms/step - loss: 0.0214 - val_loss: 0.0217
Epoch 8/10
70/70 [==============================] - 47s 670ms/step - loss: 0.0213 - val_loss: 0.0216
Epoch 9/10
70/70 [==============================] - 48s 686ms/step - loss: 0.0212 - val_loss: 0.0215
Epoch 10/10
70/70 [==============================] - 49s 701ms/step - loss: 0.0212 - val_loss: 0.0215
```

Рисунок 3.28 – Процес навчання другої моделі CNN

Як можна побачити похибка після зміни архітектури моделі та збільшення розміру зображення в останніх епохах дорівнює 0.02 це набагато краще а ніж в минулих моделях і означає що точність моделі дорівнює (1-0.02) * 100 % = 98% це дуже хороший результат загорткової нейронної мережі.

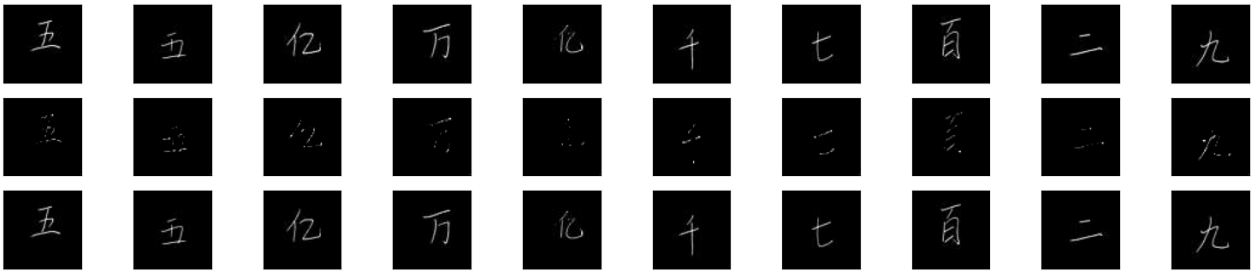

Рисунок 3.29 – Результат навчання другої моделі CNN

По результату можна побачити що стиснення та відновлення зображення стало набагато кращим.

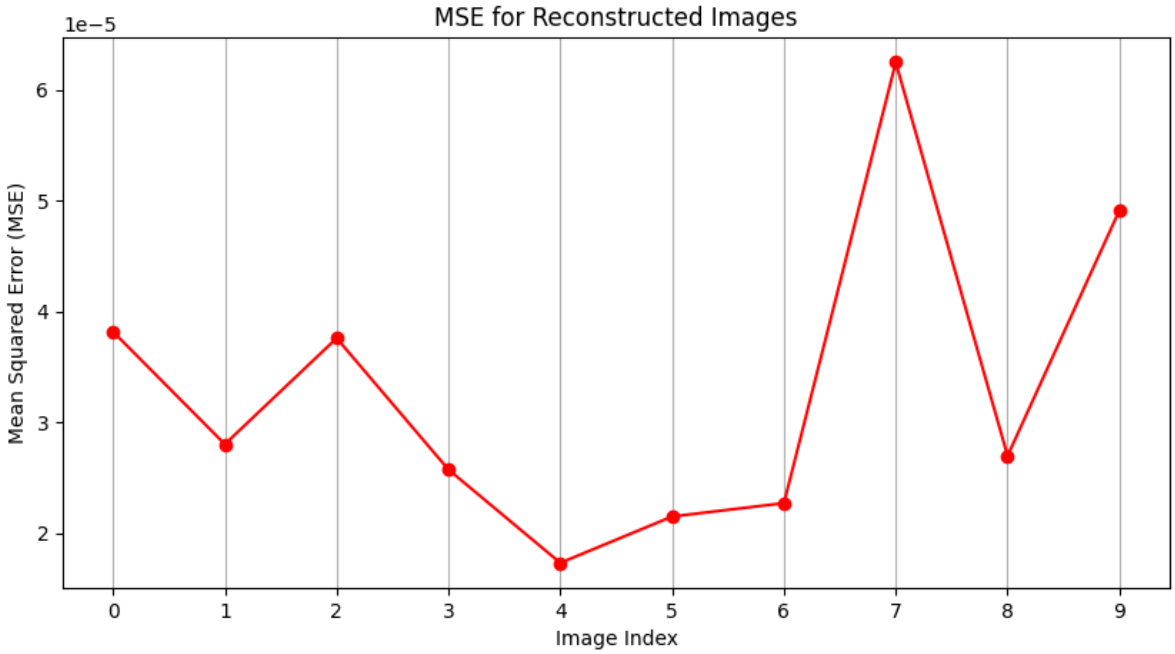

Рисунок 3.30 – Графік MSE для другої моделі CNN

Як можна побачити за графіком чим менше значення MSE, тим менше відхилення від оригінального зображення і, відповідно, тим краща якість відновленого зображення. У наведених прикладах всі значення MSE дуже малі, в діапазоні від приблизно 1.73e-05 до 6.24e-05. Це вказує на те, що всі відновлені зображення мають дуже малі відхилення від оригінальних і, отже, вважаються високоякісними.

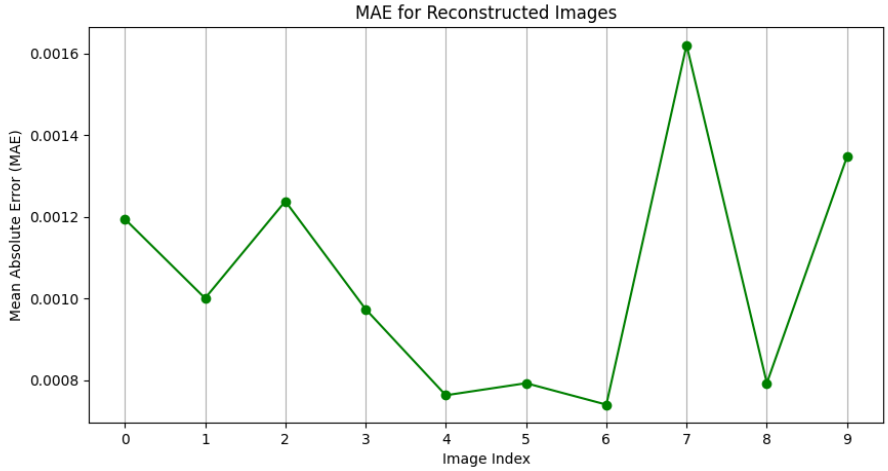

Рисунок 3.31 – Графік MAE для другої моделі CNN

Як можна побачити вище чим менше значення MAE, тим менше відмінностей між пікселями, отже, відновлене зображення вважається більш точним. У наведених прикладах значення MAE розташовані в діапазоні від приблизно 0.0007 до 0.0016, що також вказує на різний рівень точності відновлення, де менші значення MAE вказують на кращу точність відновлення.

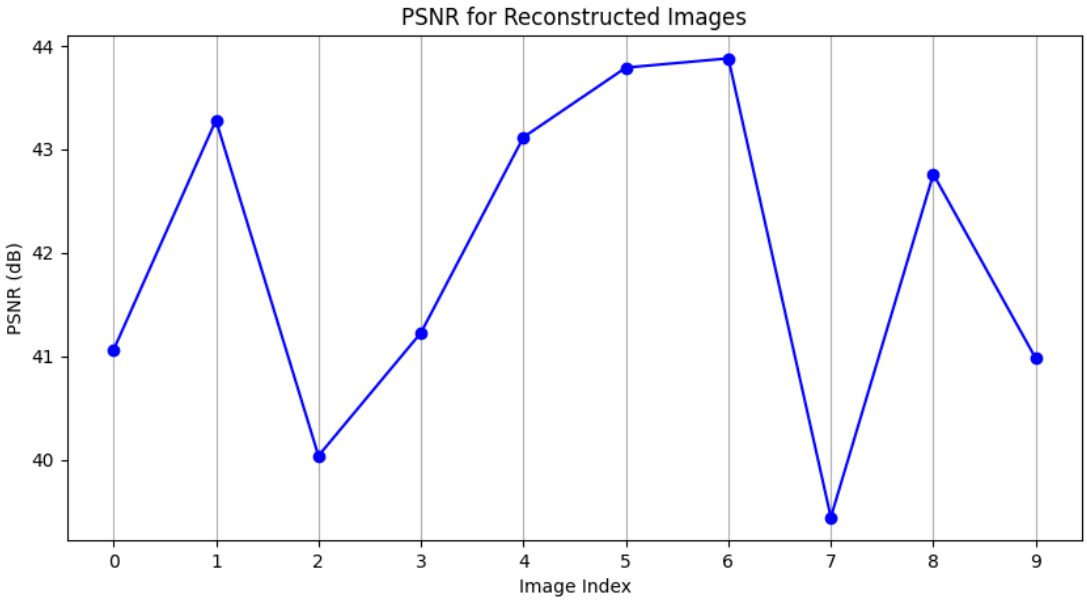

Рисунок 3.32 – Графік PSNR для першої моделі CNN

На даному графіку значення PSNR коливаються в діапазоні від приблизно 39 до 44 децибел. Загалом, вищі значення PSNR, такі як 43.28, 43.88, та 43.79 децибел, свідчать про високу якість відновлених зображень, оскільки вони мають менше артефактів та відхилень від оригіналу. На іншому кінці діапазону, значення PSNR нижче, такі як 39.44 та 40.04 децибел, вказують на меншу якість відновлених зображень, можливо, через більше кількість артефактів або втрату деталей під час процесу відновлення.

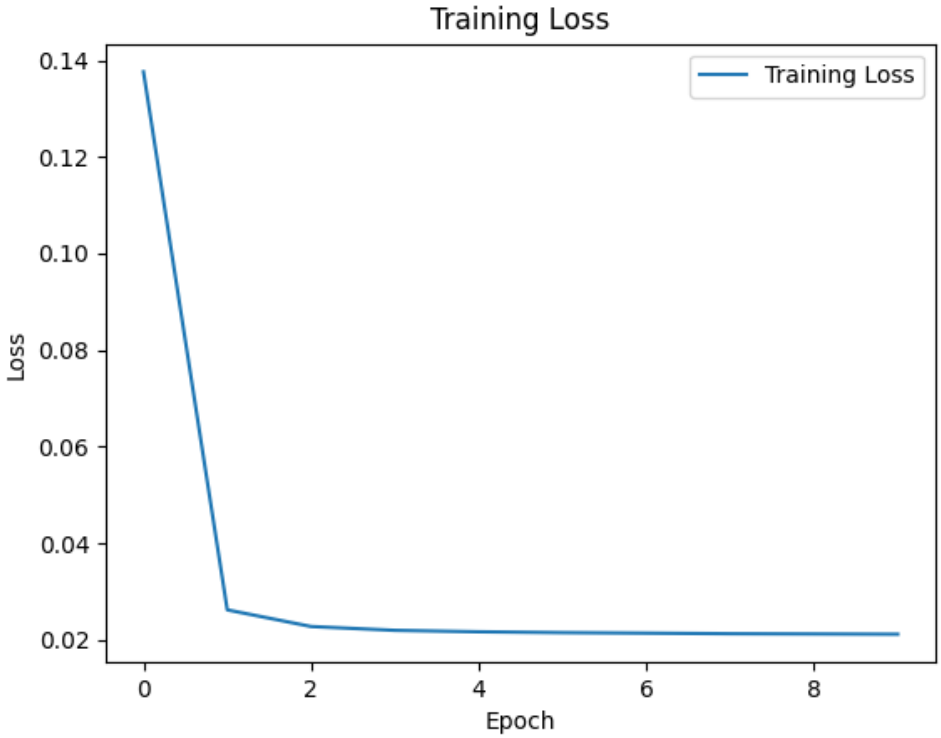

Рисунок 3.33 – Графік Loss під час навчання для другої моделі CNN

На даному графіку можна провести декілька спостережень. Наприклад зменшення значення втрат (loss) для тренувального набора даних протягом кожної епохи. Це свідчить про те, що модель поступово навчається і здатна краще адаптуватися до вхідних даних з кожною новою епохою. Зменшення значення втрат є позитивним знаком, оскільки вказує на те, що модель наближається до оптимальних параметрів для відновлення зображень.

3.5 Порівняння моделей для задачі стиснення даних

У сучасному світі обробки зображень однією з ключових задач є стиснення зображень з метою зменшення їхнього обсягу та збереження якості. Для досягнення цієї мети використовуються різноманітні методи, серед яких автоенкодери та згорткові нейронні мережі займають важливе місце.

У цьому підрозділі ми розглянемо та порівняємо два підходи до стиснення зображень. Перший підхід передбачає використання автоенкодера для стиснення

зображень розміром 28 на 28 пікселів, тоді як другий використовує автоенкодер для стиснення зображень розміром 256 на 256 пікселів. Крім того, ми також розглянемо дві згорткові нейронні мережі, одну з яких призначено для стиснення зображень розміром 28 на 28 пікселів, а іншу - для зображень розміром 256 на 256 пікселів.

Порівняння цих методів дозволить нам краще зрозуміти їхню ефективність у виконанні завдання стиснення зображень та вибрати найбільш оптимальний підхід для конкретних вимог та обмежень.

Таблиця 3.1 порівняння моделей

Моделі	Loss	MSE	MAE	PSNR
autoencoder 28x28	0.0900	0.0010	0,007	29.80
CNN 28x28	0.0768	0.0006	0.005	32.11
autoencoder 256x256	0.0260	0.0062	0.0062	24.101
CNN 256x256	0.0212	0.000021	0.00079	43.79

Модель Autoencoder 28x28 має найгірші значення за всіма метриками, окрім PSNR, де вона трохи краще, ніж CNN 28x28. Це може бути пов'язано з тим, що автокодери краще підходять для задач відновлення зображень, ніж для задач класифікації. А модель CNN 28x28 має кращі значення, ніж Autoencoder 28x28, за всіма метриками, окрім PSNR. Однак, її продуктивність все ще значно гірша, ніж у CNN 256x256. Модель CNN 256x256 має найкращі значення за всіма метриками, окрім PSNR, де вона трохи гірша, ніж Autoencoder 28x28. Однак, різниця в PSNR не є суттєвою, тому CNN 256x256 можна вважати найкращою моделлю з усіх представлених.

За таблицею видно що найкраща модель це згорткова нейрона мережа навчена для стискання 256 на 256 пікселів. Збережемо цю модель та підключимо до програми якою буде завдання стискати зображення.

3.6 програмна реалізація

Після знаходження найкращої моделі було розроблено програмну реалізацію якою була задача, використовуючи збережену модель, стискати зображення.

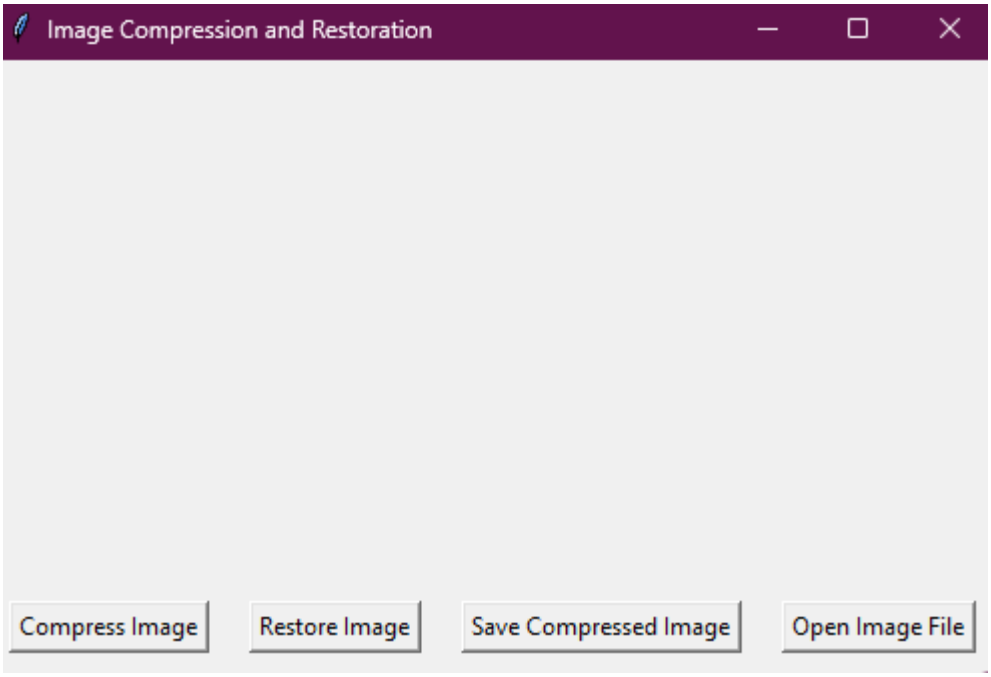

Рисунок 3.34 – Графічний інтерфейс програми

На донному малюнку зображено графічний інтерфейс програми, в програмі існує 4 кнопки це: (Compress image) – ця кнопка стискає вибране користувачем зображення; (Restore Image) – ця кнопка відновлює зображення зі створеній після стискання оригіналу; (Save compressed Image) – кнопка яка зберігає стиснене зображення; (Open Image File) – кнопка яка відкриває зображення.

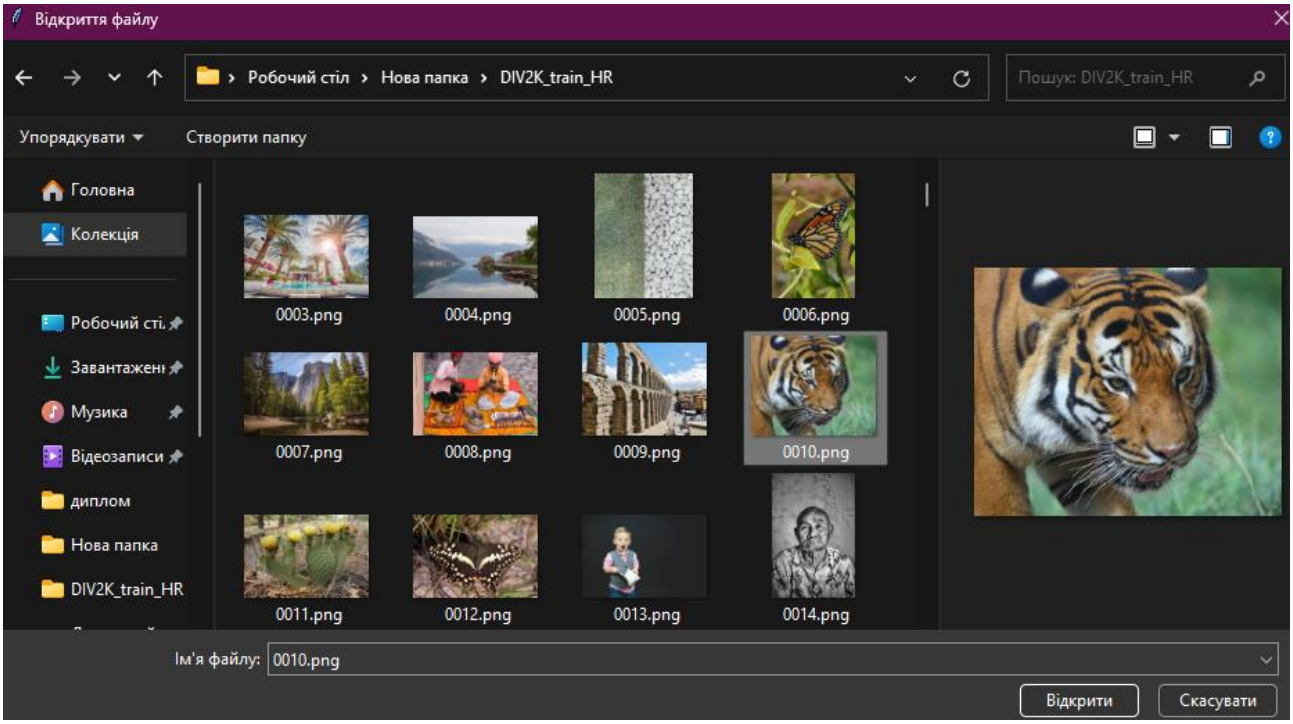

Рисунок 3.35 – Вибір зображення для стиснення

Самого початку потрібно відкрити зображення. Потрібно натиснути на кнопку Open Image File та вибрати зображення, як зображено на малюнку вище.

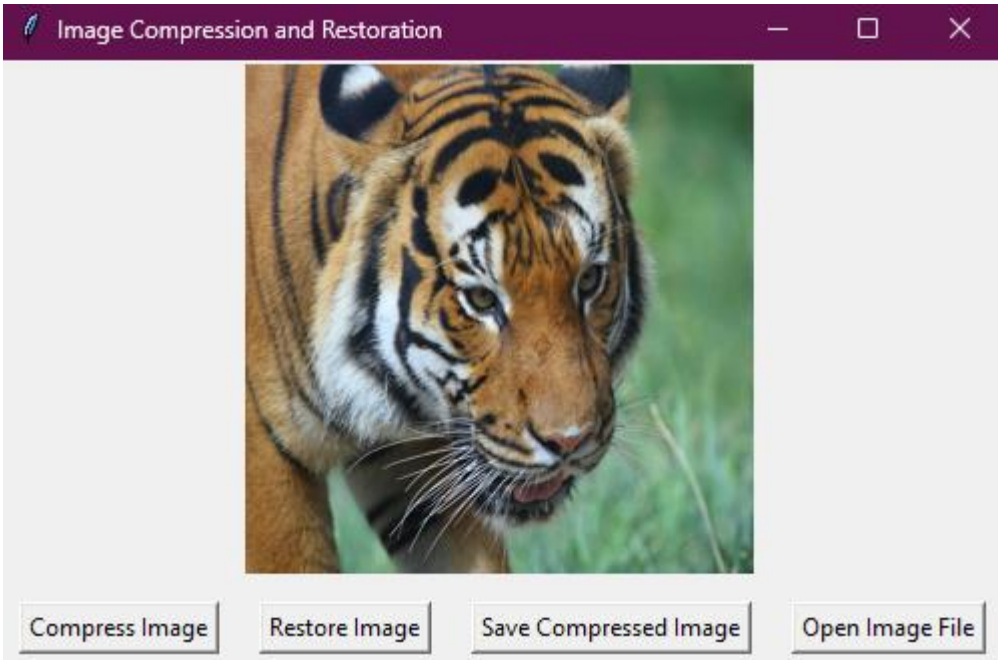

Рисунок 3.36 – Відображення оригінального зображення в програмі

На даному малюнку можемо побачити як оригінальне зображення відображається в самій програмі. Потрібно зауважити що це поле було відділено спеціально для графічного відображення процесу стиснення.

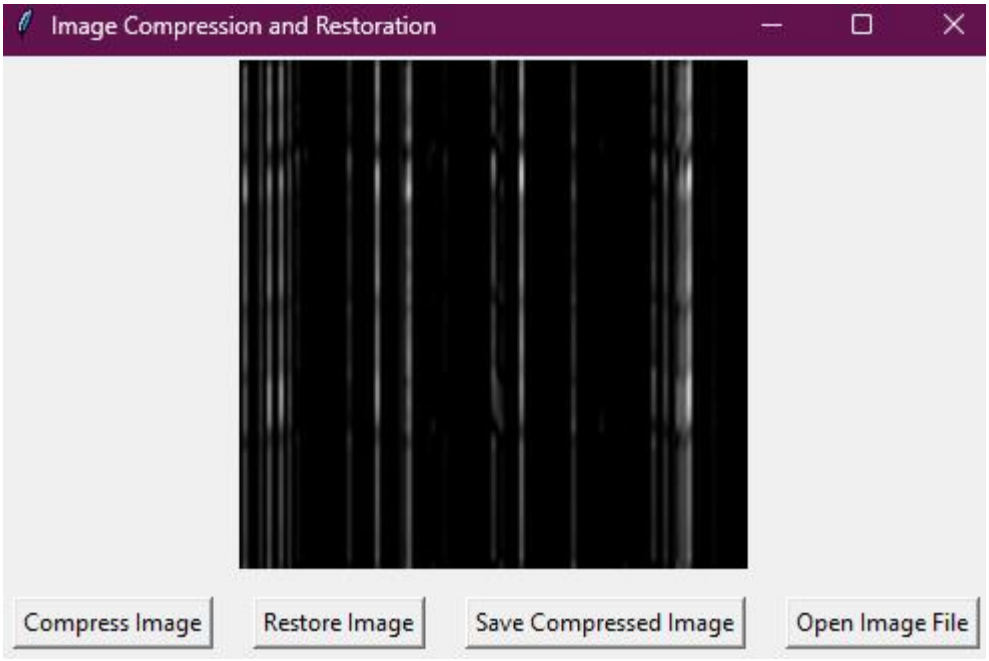

Рисунок 3.37 – Відображення стиснутого зображення в програмі

На даному малюнку можна побачити відображення зображення після стиснення для того щоб стиснути зображення потрібно натиснути на кнопку Compress image і одразу малюнок оновиться і результат замінить оригінальне зображення.

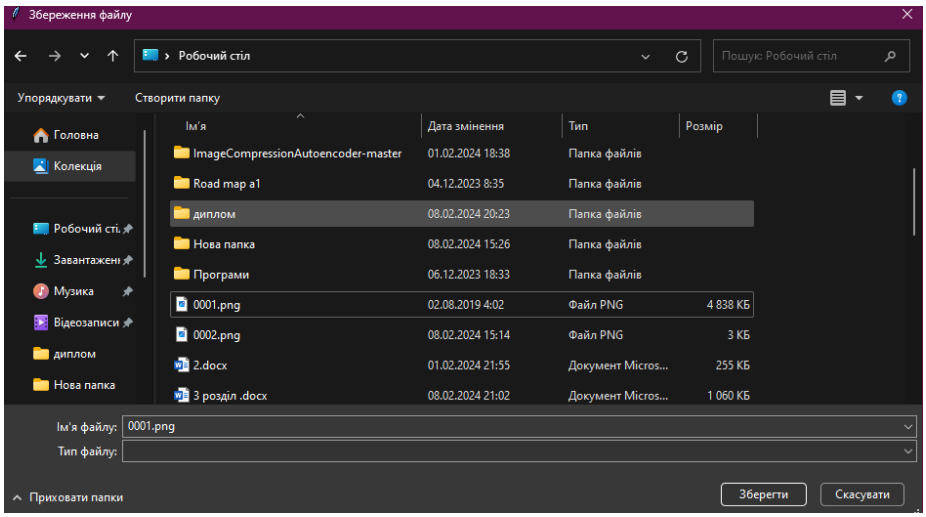

Рисунок 3.38 – Збереження стиснутого зображення

Після цього була натиснута кнопка Save compressed Image яка зберігає стиснуте фото на диск на випадок якщо в майбутньому потрібно буде його відновити.

Рисунок 3.39 – Відображення відновленого зображення в програмі

Після цього була натиснута кнопка Restore Image яка відновлює зображення після його стиснення. Потрібно звернути увагу так як мережа була навчена стискати зображення 256 на 256 пікселів програма перед процесом стискання,

тобто подання зображення на вхід моделі підготовлює зображення, а саме змінює розмір зображення, та змінює колір зображення на чорно білий.

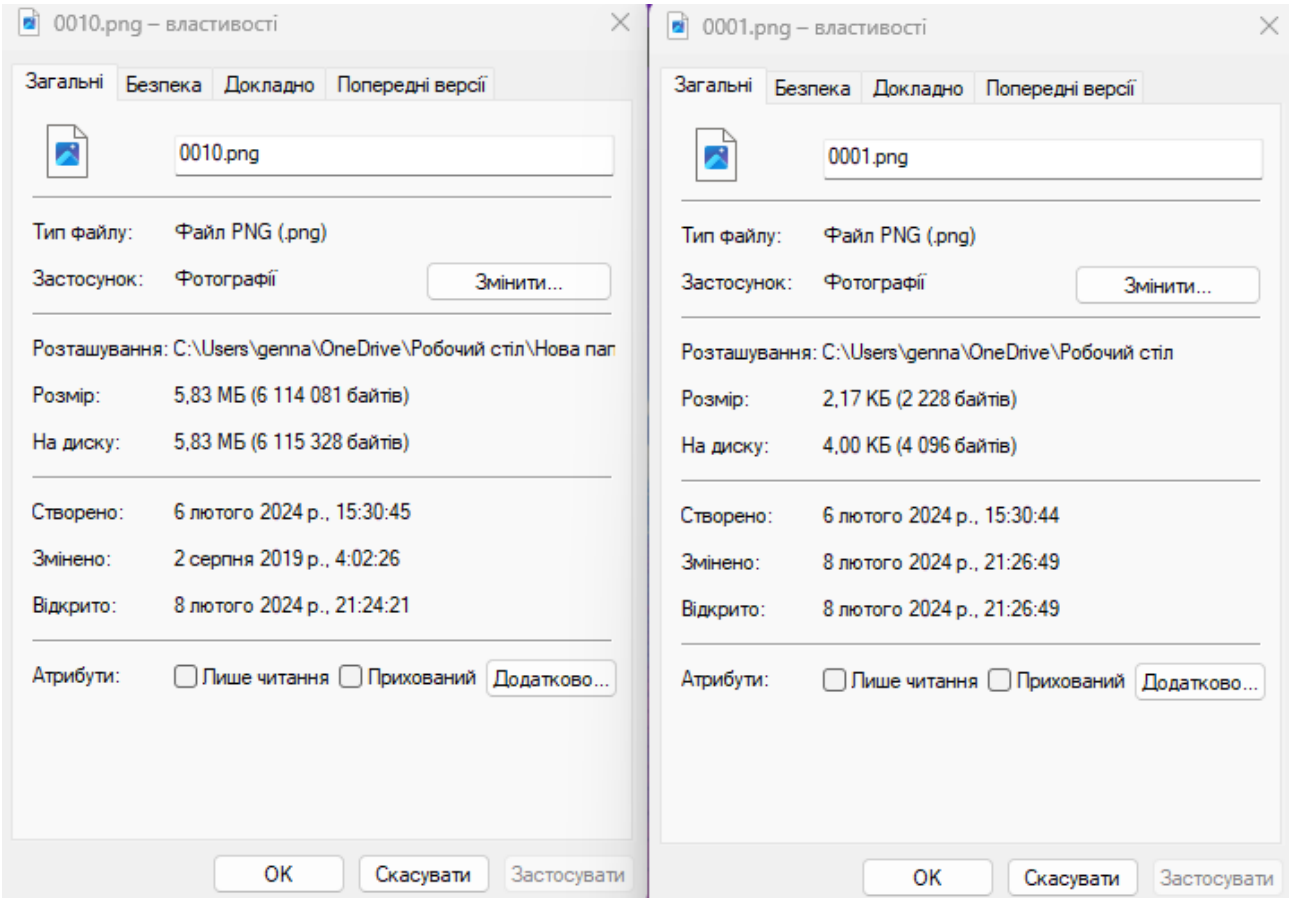

Рисунок 3.40 – Порівняння розміру зображення

На даному зображені видно як сильно змінився розмір оригінального та стиснутого зображення, Як було сказано раніше нейрона мережа була навчена стискати зображення 256 на 256 пікселів а оригінальне зображення розміром 2040 на 1644 пікселів таке зменшення розширення зображення, та навчена мережа яка стискає фото ще більше та прибирає деталі із фото вплинуло на такий малий розмір 2.17 КБ.

Висновки до розділу 3

У цьому розділі було проведено дослідження чотирьох моделей глибокого навчання для завдання стискання зображень. Два з цих моделей були автоенкодерами, а інші дві - згортковими нейронними мережами.

Перша автоенкодер модель використовувала класичну архітектуру з енкодером, який перетворює вхідні зображення на компактний вектор у складі прихованого шару, і декодер, який відновлює вихідні зображення з цього вектора яка була навчена для стиснення та відновлення зображення 28 на 28 пікселів а друга модель була навчена стискати зображення 256 на 256 пікселів.

Дві згорткові нейронні мережі також були вивчені. Перша модель використовувала конволюційні шари для здійснення стиснення зображень, а потім відновлювала їх за допомогою транспонованих конволюційних шарів. Друга згорткова нейронна мережа використовувала архітектуру енкодер-декодер з блоками згорткових та транспонованих згорткових шарів.

Крім того, було розглянуто програмну реалізацію стиснення зображень, яка виявилася ефективною в порівнянні з іншими підходами. Ця реалізація дозволяла ефективно стискати зображення з мінімальною втратою якості.

Загалом, у розділі 3 проведено порівняльний аналіз різних моделей глибокого навчання для завдання стискання зображень, включаючи автоенкодери та згорткові нейронні мережі, а також розглянуто програмну реалізацію, що відображала високу ефективність у зазначеній задачі.

ВИСНОВКИ

В ході магістерської роботи було досліджено методи стиснення даних з використанням нейронних мереж. В результаті дослідження було зроблено наступні висновки:

Нейронні мережі є потужним інструментом для стиснення даних. Вони можуть навчатися на великих наборах даних і виявляти закономірності, які неможливо виявити за допомогою традиційних методів. Це дозволяє їм досягати більш високого ступеня стиснення без втрати якості даних.

Існують різні типи нейронних мереж, які можна використовувати для стиснення даних. Найбільш поширеними типами є автоенкодери та згорткові нейронні мережі.

Автоенкодери добре підходять для стиснення даних, які мають чітку структуру, наприклад, зображення. Згорткові нейронні мережі добре підходять для стиснення даних, які мають складну структуру, наприклад, відео.

Ефективність нейронних мереж для стиснення даних залежить від багатьох факторів, таких як тип нейронної мережі, архітектура нейронної мережі, параметри нейронної мережі, набір даних, на якому навчається нейронна мережа, та метод стиснення даних.

Нейронні мережі є перспективним напрямком розвитку методів стиснення даних. Їх використання дозволяє досягти більш високого ступеня стиснення без втрати якості даних, що може бути корисним для багатьох застосувань.

В ході магістерської роботи були розглянуті як традиційні методи стиснення даних без втрат і з втратами, так і нові методи, засновані на штучному інтелекті. Були детально вивчені алгоритми Хафмана, JPEG-LS, JPEG, фрактального стиснення, а також методи, засновані на CNN та автоенкодерах. Були розроблені та протестовані автоенкодери та згорткові нейронні мережі для стиснення зображень різних розмірів. Було показано, що нейронні мережі дозволяють досягти більш високого ступеня стиснення без втрати якості даних.

В результаті виконання магістерської роботи було досягнуто наступних результатів:

– розроблено нові методи стиснення даних, засновані на нейронних мережах;

– показано, що нейронні мережі дозволяють досягти більш високого ступеня стиснення без втрати якості даних, ніж традиційні алгоритми стиснення даних:

– розроблено програмне забезпечення для реалізації розроблених нейронних мереж.

Отже під час виконання роботи було досягнено мети якої є дослідження методів ефективного стиснення даних з використанням нейронних мереж.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ziv, Jacob; Lempel, Abraham (May 1977). "A Universal Algorithm for Sequential Data Compression". IEEE Transactions on Information Theory. 23 (3): 337–343

2. Terry Welch, "A Technique for High Performance Data Compression", IEEE Computer, vol. 17, No. 6, pp. 8-19, Jun. 1984

3. Software for JPEG XT. URL: https://jpeg.org/jpegxt/software.html (дата звернення 15.12.2023)

4. "JBIG: A Standard for Progressive Bi-level Image Compression", by R. L. Baker, D. S. Taubman, and M. Marcellin, in Journal of Electronic Imaging, vol. 2, no. 3, July 1993, pp. 183-191

5. ITU-T Recommendation T.82:1993 (JBIG) URL: https://www.itu.int/rec/T-REC-T.82-199303-I/en (дата звернення 15.12.2023)

6. Дізнатися про JPEG-файли URL: https://www.adobe.com/ua/creativecloud/file-types/image/raster/jpeg-file.html (дата звернення 15.12.2023)

7. Lenz, W. (1879). Theorie der magnetischen Kreise. Annalen der Physik und Chemie. 223-240

8. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and retrieval. Psychological Review. 386-408

9. Minsky, M., & Papert, S. (1969). Perceptrons: An introduction to computational geometry. Cambridge, MA: MIT Press. 31-32.

10. Амарі, С. (1967). Learning patterns and pattern sequences by self-organizing systems. IEEE Transactions on Computers, 46-57

11. Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43(1), 59-69

12. Fukushima, K. (1980). Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks, 3(2), 263-280.

13. Fukushima, K. (1969). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 19(3), 193-202.

14. Lecun, Y. Bengio, Y. Hinton, G. (2015). Deep learning. Nature. 436-444

15. Schmidhuber, Jürgen 1992. Learning to control fast-weight memories: an alternative to recurrent nets.. Neural Computation 131–139

16. Vaswani, Ashish 2017. "Attention is all you need." Advances in neural information processing systems. 504-512

17. GAN 2.0: NVIDIA's Hyperrealistic Face Generator. https://syncedreview.com/2018/12/14/gan-2-0-nvidias-hyperrealistic-face-generator (дата звернення 15.12.2023)

18. Kramer, Mark A. (1991). Nonlinear principal component analysis using autoassociative neural networks . 233–243

19. Сало, І. А. Стиснення даних: теорія та практика, навчальний посібник : Ліра-К, 2016. 304 с.

20. Сталл, Р. Алгоритми стиснення даних Р. Сталл. – СПб.: БХВ-Петербург, 2010. – 688 с.

21. Комаров, А. В. Алгоритми стиснення даних: навч. посіб. / А. В. Комаров. – М.: Фізматліт, 2014. – 240 с.

22. Huffman, D. A. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9), 1099-1101, 1952.

23. Nelson, M. R. Data compression with the Burrows-Wheeler transform. ACM Transactions on Information Systems, 14(1), 1-15, 1996.

24. Sayood, K. Introduction to data compression. Morgan Kaufmann, 2005.

25. Ziv, J., & Lempel, A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23(3), 337-342, 1977.

26. Cover, T. M., & Thomas, J. A. Elements of information theory. John Wiley & Sons, 2006.

27. Shannon, C. E. A mathematical theory of communication. The Bell System Technical Journal, 27(3), 379-423, 1948.

28. Huffman, D. A. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9), 1099-1101, 1952.

29. Said, A., & Pearlman, W. A., A new, fast, and efficient image codec based on set partitioning in hierarchical trees (SPIHT). IEEE Transactions on Circuits and Systems for Video Technology, 2000. 10(3), 443-456.

30. Wu, X., & Memon, N. D. Context-based, adaptive, lossless image coding. IEEE Transactions on Communications, 1997. 45(4), 437-444.

31. Chen, W., & Smith, J. R. . JPEG-LS: Lossless image compression using adaptive prediction. IEEE Transactions on Image Processing, 2002. 11(2), 176-188.

32. Wallace, G. K. The JPEG still picture compression standard. IEEE Transactions on Consumer Electronics, 1991, 38(1), 18-34.

33. Watson, A. B. DCT quantization matrices visually optimized for individual images. Human Vision, Visual Processing, and Digital Display III, 1913, 202-216.

34. Skodras, A., & Tourapis, A. M. JPEG 2000: The new standard for image compression. IEEE Signal Processing Magazine, 21(2), 36-58.

35. Martucci, S. A. JPEG2000: A concise overview. IEEE Signal Processing Magazine, 18(5), 80-90.

36. Christopoulos, C., & Skodras, A. The JPEG 2000 still image coding system: An overview. IEEE Transactions on Consumer Electronics, 46(4), 1103-1127.

37. Horowitz, M., & Hill, F. The JPEG 2000 standard: A technical overview. IEEE Signal Processing Magazine, 19(6), 46-60.

38. Wallace, G. K. The JPEG still picture compression standard. IEEE Transactions on Consumer Electronics, 1991, 38(1), 18-34.

39. Watson, A. B. DCT quantization matrices visually optimized for individual images. Human Vision, Visual Processing, and Digital Display III, 1913, 202-216.

40. Said, A., & Pearlman, W. A. A new, fast, and efficient image codec based on set partitioning in hierarchical trees (SPIHT). IEEE Transactions on Circuits and Systems for Video Technology, 2000, 10(3), 443-456.

41. Wu, X., & Memon, N. D. Context-based, adaptive, lossless image coding. IEEE Transactions on Communications, 1997, 45(4), 437-444.

42. Chen, W., & Smith, J. R. JPEG-LS: Lossless image compression using adaptive prediction. IEEE Transactions on Image Processing, 2002, 11(2), 176-188.

43. Skodras, A., & Tourapis, A. M. JPEG 2000: The new standard for image compression. IEEE Signal Processing Magazine, 2004, 21(2), 36-58.

44. Martucci, S. A. JPEG2000: A concise overview. IEEE Signal Processing Magazine, 2000. 18(5), 80-90.

45. Christopoulos, C., & Skodras, A. The JPEG 2000 still image coding system: An overview. IEEE Transactions on Consumer Electronics, 2000, 46(4), 1103-1127.

46. Horowitz, M., & Hill, F. The JPEG 2000 standard: A technical overview. IEEE Signal Processing Magazine, 2002, 19(6), 46-60.

47. Shannon, C. E. A mathematical theory of communication. The Bell System Technical Journal, 1948, 27(3), 379-423.

48. Rice, R. F. Some practical universal noiseless coding techniques. JPL Publication, 1979, 79-22.

49. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11), 2278-2324.

50. Krizhevsky, A., Sutskever, I., & Hinton, G. E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012, 25, 1097-1105.

51. Simonyan, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv preprint 2014, arXiv:1409.1556.

52. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, 770-778.

53. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K. Q. Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, 2261-2269.

54. Khan, A., Sohail, A., Zahoora, U., Qureshi, A. S. A survey of the recent advances in convolutional neural networks (CNNs) applications in computer vision. Artificial Intelligence Review, 2020, 53(8), 5499-5531.

55. Wang, L., Li, Y. Convolutional neural network based on image classification. Journal of Physics: Conference Series, 2019, 1237(1), 012027.

56. Ojala, T., Pietikäinen, M. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 24(7), 971-987.

57. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, 770-778.

58. Krizhevsky, A., Sutskever, I., Hinton, G. E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012, 25, 1097-1105.

59. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11), 2278-2324.

60. Agrawal, M., Girshick, R., Malik, J. Long short-term memory networks for visual recognition. 2014, arXiv preprint arXiv:1409.1556.

61. Donahue, J., Hendricks, J. R., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. 2015, Long-term recurrent convolutional networks for visual recognition and description. arXiv preprint arXiv:1411.4389.

62. Karpathy, A., Fei-Fei, L. Deep visual-semantic alignments for generating image descriptions. 2015, arXiv preprint arXiv:1411.4533.

63. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Rusu, A. A., ... & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. arXiv preprint arXiv:1502.03044.

64. Vinyals, O., Toshev, A., Bengio, S., Erhan, D. Show and tell: A neural image caption generator. 2015, arXiv preprint arXiv:1411.4555.

ДОДАТОК А
Код програмної реалізації

autoencoder_28_28.py

```
from keras.layers import Input, Dense, Flatten, Reshape
from keras.models import Model
from keras.datasets import mnist
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from keras.callbacks import History
from skimage.metrics import peak_signal_noise_ratio as psnr

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
x_train = x_train.astype('float32') / 255.
x_test  = x_test .astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test  = np.reshape(x_test,  (len(x_test),  28, 28, 1))

# Function to plot images
def plot_digits(*args):
    args = [x.squeeze() for x in args]
    n = min([x.shape[0] for x in args])

    plt.figure(figsize=(2*n, 2*len(args)))
    for j in range(n):
        for i in range(len(args)):
            ax = plt.subplot(len(args), n, i*n + j + 1)
            plt.imshow(args[i][j])
            plt.gray()
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)

    plt.show()

# Function to create deep dense autoencoder
def create_deep_dense_ae():
    # Dimension of the encoded representation
    encoding_dim = 49

    # Encoder
    input_img = Input(shape=(28, 28, 1))
    flat_img = Flatten()(input_img)
    x = Dense(encoding_dim*3, activation='relu')(flat_img)
    x = Dense(encoding_dim*2, activation='relu')(x)
```

```
    encoded = Dense(encoding_dim, activation='linear')(x)

    # Decoder
    input_encoded = Input(shape=(encoding_dim,))
    x = Dense(encoding_dim*2, activation='relu')(input_encoded)
    x = Dense(encoding_dim*3, activation='relu')(x)
    flat_decoded = Dense(28*28, activation='sigmoid')(x)
    decoded = Reshape((28, 28, 1))(flat_decoded)

    # Models
    encoder = Model(input_img, encoded, name="encoder")
    decoder = Model(input_encoded, decoded, name="decoder")
    autoencoder = Model(input_img, decoder(encoder(input_img)), name="autoencoder")
    return encoder, decoder, autoencoder

# Create the deep dense autoencoder
d_encoder, d_decoder, d_autoencoder = create_deep_dense_ae()
d_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

history = History()

# Train the autoencoder
d_autoencoder.fit(x_train, x_train,
                  epochs=10,
                  batch_size=256,
                  shuffle=True,
                  validation_data=(x_test, x_test),
                  callbacks=[history])

n = 10

imgs = x_test[:n]
encoded_imgs = d_encoder.predict(imgs, batch_size=n)
encoded_imgs[0]

decoded_imgs = d_decoder.predict(encoded_imgs, batch_size=n)

plot_digits(imgs, decoded_imgs)

# Compute and display metrics for each image
mse_values = []
mae_values = []
psnr_values = []

for i in range(n):
    mse = np.mean((imgs[i] - decoded_imgs[i]) ** 2)
    mae = np.mean(np.abs(imgs[i] - decoded_imgs[i]))
    psnr_value = psnr(imgs[i], decoded_imgs[i], data_range=imgs[i].max() - imgs[i].min())
```

```
    mse_values.append(mse)
    mae_values.append(mae)
    psnr_values.append(psnr_value)

    print(f"Image {i + 1}: MSE = {mse}, MAE = {mae}, PSNR = {psnr_value}")

# Plot MSE
plt.figure(figsize=(10, 5))
plt.plot(range(n), mse_values, marker='o', color='r')
plt.xlabel('Image Index')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot MAE
plt.figure(figsize=(10, 5))
plt.plot(range(n), mae_values, marker='o', color='g')
plt.xlabel('Image Index')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('MAE for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot PSNR
plt.figure(figsize=(10, 5))
plt.plot(range(n), psnr_values, marker='o', color='b')
plt.xlabel('Image Index')
plt.ylabel('PSNR (dB)')
plt.title('PSNR for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

cnn_256_256.py

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
import matplotlib.pyplot as plt
from keras.callbacks import History
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.metrics import structural_similarity as ssim

# Check if GPU is available and set memory growth
physical_devices = tf.config.list_physical_devices('GPU')
print(len(physical_devices))
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.2989, 0.5870, 0.1140])

# Define the directory containing the images
data_dir = 'data2'

# Get the list of image files
image_files = [file for file in os.listdir(data_dir) if file.endswith('.jpg')]

# Load and format images with grayscale conversion
images = [rgb2gray(img_to_array(load_img(os.path.join(data_dir, file),
target_size=(256, 256)))) for file in image_files]
images = np.array(images)

# Check the number of images
if len(images) < 1:
    raise ValueError("There are no images in the dataset.")

# Normalize pixels to the range [0, 1]
images = images.astype('float32') / 255.0

# Split into training and testing sets
x_train, x_test = train_test_split(images, test_size=0.3, random_state=42)

# Reshape images to match the expected input shape
x_train = np.reshape(x_train, (len(x_train), 256, 256, 1))
x_test = np.reshape(x_test, (len(x_test), 256, 256, 1))

def create_deep_conv():
```

```
    input_img = Input(shape=(256, 256, 1))
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    coded = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x_decoded = Conv2D(64, (3, 3), activation='relu', padding='same')(coded)
    x_decoded = UpSampling2D((2, 2))(x_decoded)
    x_decoded = Conv2D(32, (3, 3), activation='relu', padding='same')(x_decoded)
    x_decoded = UpSampling2D((2, 2))(x_decoded)
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x_decoded)
    coder = Model(input_img, coded, name="coder")
    decoder = Model(coded, decoded, name="decoder")
    CNN = Model(input_img, decoder(coder(input_img)), name="CNN")
    return coder, decoder, CNN

# Create the deep convolutional autoencoder models
c_coder, c_decoder, CNN = create_deep_conv()
CNN.compile(optimizer='adam', loss='binary_crossentropy')

# Display summary of the CNN model
CNN.summary()

# Train the CNN model
history = History()
CNN.fit(x_train, x_train,
                epochs=10,
                batch_size=10,
                shuffle=True,
                validation_data=(x_test, x_test),
                callbacks=[history])

n = 10

# Display the reconstructed images
imgs = x_test[:n]
coded_imgs = c_coder.predict(imgs, batch_size=n)
decoded_imgs = c_decoder.predict(coded_imgs, batch_size=n)

plt.figure(figsize=(20, 4))
for i in range(n):
    # Original images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(imgs[i].reshape(256, 256), cmap='gray')  # Adjusted to the image size
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```
# Display compressed images
for i in range(n):
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(coded_imgs[i].reshape(64, 64, 128)[..., 0], cmap='gray')  # Adjusted
to the encoder output size
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

# Display reconstructed images
for i in range(n):
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(decoded_imgs[i].reshape(256, 256), cmap='gray')  # Adjusted to the
image size
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

# Calculate and plot metrics for each image
mse_values = []
mae_values = []
psnr_values = []

for i in range(n):
    mse = np.mean((imgs[i] - decoded_imgs[i]) ** 2)
    mae = np.mean(np.abs(imgs[i] - decoded_imgs[i]))
    psnr_value = psnr(imgs[i], decoded_imgs[i], data_range=imgs[i].max() -
imgs[i].min())

    mse_values.append(mse)
    mae_values.append(mae)
    psnr_values.append(psnr_value)

    print(f"Image {i + 1}: MSE = {mse}, MAE = {mae}, PSNR = {psnr_value}")

# Plot MSE
plt.figure(figsize=(10, 5))
plt.plot(range(n), mse_values, marker='o', color='r')
plt.xlabel('Image Index')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot MAE
```

```
plt.figure(figsize=(10, 5))
plt.plot(range(n), mae_values, marker='o', color='g')
plt.xlabel('Image Index')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('MAE for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot PSNR
plt.figure(figsize=(10, 5))
plt.plot(range(n), psnr_values, marker='o', color='b')
plt.xlabel('Image Index')
plt.ylabel('PSNR (dB)')
plt.title('PSNR for Reconstructed Images')
plt.xticks(range(n))
plt.grid(axis='x')
plt.show()

# Plot training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Save the models
c_coder.save('c_coder_model.h5')
c_decoder.save('c_decoder_model.h5')
CNN.save('CNN_model.h5')
```

program.py

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import pickle

# Завантаження збережених моделей
c_coder = load_model('c_coder_model.h5')
c_decoder = load_model('c_decoder_model.h5')
CNN = load_model('CNN_model.h5')

coded_image_path = 'coded_image.pkl'  # Шлях до файлу для збереження стисненого зображення
```

```
# Функція для збереження стисненого зображення
def save_coded_image(coded_image):
    with open(coded_image_path, 'wb') as f:
        pickle.dump(coded_image, f)

# Функція для завантаження стисненого зображення
def load_coded_image():
    global coded_image
    try:
        with open(coded_image_path, 'rb') as f:
            coded_image = pickle.load(f)
            return True
    except FileNotFoundError:
        return False

coded_image = None
load_coded_image()

def compress_image():
    global original_image_pil, coded_image
    # Перетворення зображення в потрібний формат
    image = original_image_pil.resize((256, 256))  # Розмір вхідного зображення моделі
    image_array = np.array(image.convert('L')) / 255.0  # Перетворення в чорно-біле та
нормалізація

    # Стискання зображення
    coded_image = c_coder.predict(np.expand_dims(image_array, axis=0).reshape(1, 256, 256,
1))

    # Видалення попереднього зображення перед відображенням нового
    canvas.delete(tk.ALL)

    # Збереження стисненого зображення
    save_coded_image(coded_image)

    # Відображення стисненого зображення
    compressed_image = (coded_image.reshape(64, 64, 128) * 255).astype(np.uint8)[0]
    compressed_image = Image.fromarray(compressed_image)
    compressed_image = compressed_image.resize((256, 256), Image.ANTIALIAS)  # Розширення
зображення
    compressed_image = ImageTk.PhotoImage(compressed_image)
    canvas.create_image(0, 0, anchor='nw', image=compressed_image)
    canvas.compressed_image = compressed_image

def restore_image():
    global coded_image
    # Відновлення зображення зі стисненого зображення
```

```
    if coded_image is None:
        messagebox.showerror("Error", "No compressed image found. Please compress an image
first.")
        return

    decoded_image = c_decoder.predict(coded_image)

    # Видалення попереднього зображення перед відображенням нового
    canvas.delete(tk.ALL)

    # Перетворення зображення назад у зображення PIL та показ його на екрані
    decoded_image = (decoded_image.reshape(256, 256) * 255).astype(np.uint8)
    decoded_image = Image.fromarray(decoded_image)
    decoded_image = ImageTk.PhotoImage(decoded_image)
    canvas.create_image(0, 0, anchor='nw', image=decoded_image)
    canvas.decoded_image = decoded_image

def save_compressed_image():
    global coded_image
    # Перетворення стисненого зображення назад у зображення PIL та збереження його
    compressed_image = Image.fromarray((coded_image.reshape(64, 64, 128) *
255).astype(np.uint8)[0])
    save_path = filedialog.asksaveasfilename(defaultextension=".jpg")
    if save_path:
        compressed_image.save(save_path)

def open_file():
    global original_image, original_image_pil
    # Відкриття файлу та завантаження його як зображення
    file_path = filedialog.askopenfilename()
    if file_path:
        original_image_pil = Image.open(file_path)
        original_image_pil = original_image_pil.resize((256, 256))  # Зміна розміру для
підгонки під модель
        original_image = ImageTk.PhotoImage(original_image_pil)
        canvas.create_image(0, 0, anchor='nw', image=original_image)
        canvas.original_image = original_image

# Створення вікна
root = tk.Tk()
root.title("Image Compression and Restoration")

# Створення полотна для відображення зображень
canvas = tk.Canvas(root, width=256, height=256)
canvas.pack()

# Створення кнопок
compress_button = tk.Button(root, text="Compress Image", command=compress_image)
```

```
compress_button.pack(side=tk.LEFT, padx=10, pady=10)

restore_button = tk.Button(root, text="Restore Image", command=restore_image)
restore_button.pack(side=tk.LEFT, padx=10, pady=10)

save_button = tk.Button(root, text="Save Compressed Image", command=save_compressed_image)
save_button.pack(side=tk.LEFT, padx=10, pady=10)

open_button = tk.Button(root, text="Open Image File", command=open_file)
open_button.pack(side=tk.LEFT, padx=10, pady=10)

root.mainloop()
```