Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Кібербезпеки
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
(позначення документа)

Аналіз методів кодування зображень на основі машинного навчання
(тема)

Виконав:
здобувач 3 року навчання,
групи ТРІМІу-23-1
Григорій КОЗАКОВ
(власне ім'я, прізвище)

Спеціальність 172 – Електронні комунікації та радіотехніка
(код і повна назва спеціальності)

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

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник ст. викл. Олексій ФЕДОРОВ.
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ Микола МОСКАЛЕЦЬ
(підпис) (власне ім'я, призвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Кібербезпеки

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 172 – Електронні комунікації та радіотехніка
(код і повна назва)

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

Освітня програма "Інформаційно-мережна інженерія "

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувачеві Козаков Григорій Михайлович
(прізвище, ім'я, по батькові)

1. Тема роботи:
Аналіз методів кодування зображень на основі машинного навчання
затверджена наказом по університету від " 20 " 05 2026 р. № 415Ст
2. Термін подання студентом роботи 29.06.2026 р.
3. Вихідні дані до проекту (роботи)
1. Провести аналіз сучасних штучних нейронних мереж.
2. Дослідити процес стиснення зображень.
3. Дослідити методи машинного зору.
4. Дослідити основні методи стиснення зображень із використанням ШНМ.
5. Обрати розроблений фреймворк ШНМ розроблений для стиснення зображень.
6. Провести практичний аналіз методу стиснення зображень за допомогою обраної моделі.
7. Створити структурований аналітичний висновок із доцільності використання ШНМ для стиснення зображень.
4. Перелік питань, що потрібно опрацювати в роботі
ВСТУП.
1 ТЕОРЕТИЧНІ ОСНОВИ, ПРИНЦИП РОБОТИ, МЕТОДИ НАВЧАННЯ ТА ВИДИ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ.
2 ТЕОРЕТИЧНІ ОСНОВИ СТИСНЕННЯ ЗОБРАЖЕНЬ ТА МЕТОДІВ МАШИННОГО НАВЧАННЯ.
3 ОСНОВНІ ПІДХОДИ ДО СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ.
4 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕХАНІЗМУ СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ.
ВИСНОВКИ.
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.
ДОДАТКИ.

5. Перелік графічного матеріалу із зазначенням обов'язкових креслеників, схем, плакатів, комп'ютерних ілюстрацій:

слайди презентації в форматі MS Power Point

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз сучасних штучних нейронних мереж	23.05.26–27.05.26	виконано
2.	Аналіз процесу стиснення зображень	28.05.26–30.05.26	виконано
3.	Аналіз методів стиснення зображень ШНМ	31.05.26–08.06.26	виконано
4.	Реалізація методу стиснення зображень ШНМ	09.06.26–16.06.26	виконано
5.	Створення презентації до пояснювальної записки	17.06.26–19.06.26	виконано
6.	Перевірка записки керівником	20.06.26–21.06.26	виконано
7.	Перевірка на академічний плагіат	22.06.26–23.06.26	виконано
8.	Перевірка завідувачем кафедри, рецензування	24.06.26–29.06.26	виконано
9.			
10.			
11.			
12.			

Дата видачі завдання *20.05.2026 р.*

Студент _____ Григорій КОЗАКОВ
(підпис) (посада, власне ім'я, прізвище)

Керівник роботи _____ ст. викл. Олексій ФЕДОРОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра містить: 62 сторінки, 3 рисунків, 3 таблицы, 31 джерело, 2 додатки.

Об’єкт дослідження – механізм стиснення цифрових графічних даних.

Предмет досліджень – методи та алгоритми стиснення зображень на основі штучних нейронних мереж.

Мета роботи – дослідження ефективності методу стиснення зображень із використанням ШНМ-кодеків.

Методи досліджень: аналіз сучасної науково-технічної літературі, класифікація архітектур штучних нейронних мереж, експериментальне оцінювання якості відновлених зображень за метриками PSNR та BPP.

В цій кваліфікаційній роботі було проведено аналіз сучасних методів стиснення цифрових зображень, классичного стандарту JPEG і нейромережевого підходу, що базується на глибинному навчанні. Розглянуто архітектури автокодерів на базі штучних нейронних мереж, які використовуються для стиснення зображень. Проведено аналіз процесу кодування і декодування зображення із використанням нейромережі.

В експериментальній частині було проведено тестове стиснення зображень із набору kodak із залученням штучної нейронної мережі моделі bmshj2018-hyperprior. В ході тестів було оцінено якість реконструйованих зображень у порівнянні із класичним стандартом JPEG за допомогою метрик PSNR та BPP. Отримані експериментальні результати підтверджують доцільність інтеграції ШНМ-кодеків у сучасну систему передачі й збереження цифрових зображень.

СТИСНЕННЯ ЗОБРАЖЕНЬ, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ГЛИБОКЕ НАЧАННЯ, АВТОКОДЕР, PSNR, BPP

ABSTRACT

The explanatory note to the qualification work contains: 62 pages, 35 figures, 3 tables, 31 sources, 2 appendices.

The object of the study is the mechanism of compression of digital graphic data.

The subject of the research is methods and algorithms for image compression based on artificial neural networks.

The purpose of the work is to study the feasibility and effectiveness of the neural network method of image compression in comparison with the classical JPEG standard.

Research methods: analysis of modern scientific and technical literature, classification of artificial neural network architectures, experimental assessment of the quality of restored images using PSNR and BPP metrics.

In this qualification work, an analysis of modern methods of digital image compression, the classical JPEG standard and a neural network approach based on deep learning were conducted. The architectures of autoencoders based on artificial neural networks used for image compression are considered. The process of encoding and decoding an image using a neural network is analyzed.

In the experimental part, test compression of images from the kodak set was carried out using the artificial neural network of the bmshj2018-hyperprior model. During the tests, the quality of reconstructed images was assessed in comparison with the classic JPEG standard using the PSNR and BPP metrics. The experimental results obtained confirm the feasibility of integrating ANN codecs into a modern system for transmitting and storing digital images.

IMAGE COMPRESSION, ARTIFICIAL NEURAL NETWORKS, DEEP LEARNING, AUTOENCODER, PSNR, BPP

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 9
ВСТУП 11
1 ТЕОРЕТИЧНІ ОСНОВИ, ПРИНЦИП РОБОТИ, МЕТОДИ НАВЧАННЯ ТА ВИДИ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ 12
1.1 Теоретичні основи штучних нейронних мережі 12
1.2 Принцип роботи штучної нейронної мережі 14
1.3 Методи навчання штучних нейронних мереж 15
1.4 Види глибоких штучних нейронних мереж 16
1.4.1 Рекурентні нейронні мережі (RNN) 16
1.4.2 Згорткові нейронні мережі (CNN) 16
1.4.3 Генеративно-змагальні мережі (GAN) 17
2 ТЕОРЕТИЧНІ ОСНОВИ СТИСНЕННЯ ЗОБРАЖЕНЬ ТА МЕТОДІВ МАШИННОГО НАВЧАННЯ 18
2.1 Проблематика стиснення зображень 18
2.2 Перехід до машинного навчання 19
2.3 Узагальнена концепція стисення зображень за допомогорю машинного навчання 20
2.3.1 Нелінійні перетворення 20
2.3.2 Квантування 21
2.3.3 Ентропійне кодування 21
2.4 Класифікація параметрів які важливо розуміти при побудові системи обробки зображень за допомогою машинного навчання 23
2.4.1 Тип нейронної мережі 23
2.4.2 Розмір одиниці кодування 23
2.4.3 Механізми просторової кореляції 23
2.4.4 Стратегія керування бітрейтом 24
2.5 Принцип роботи методів машинного навчання для стиснення зображень . 24

3 ОСНОВНІ ПІДХОДИ ДО СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ 26
3.1 Багатошарове кодування зображень на основі перцептрона 26
3.2 Стиснення зображень із використанням випадкової нейронної мережі (Random Neural Network) 27
3.3 Стиснення зображення із використанням згорткової нейронної мережі 28
3.3.1 Основне теоретичне уявлення про CNN 28
3.3.2 Наскрізне стиснення зображень на основі CNN 29
3.3.3 Навчання згорткових мереж для стиснення зображень з вагою за змістом 30
3.4 Стиснення зображення із використанням рекурентної нейронної мережі . 31
3.4.1 Основне теоретичне уявлення про RNN 31
3.4.2 Стиснення зображень зі змінною швидкістю за допомогою рекурентної нейронної мережі 35
3.4.3 Стиснення зображень з повною роздільною здатністю за допомогою рекурентних нейронних мереж 35
3.5 Стиснення зображення із використанням генеративно-змагальних мереж 37
3.5.1 Основне теоретичне уявлення про GAN 37
3.5.2 Адаптивне стиснення зображень у реальному часі 38
3.5.3 Генеративно-змагальні мережі для стиснення зображень з екстремально низькою роздільною здатністю 39
3.6 Систематизований огляд практичних рішень стиснення зображень на основі штучних нейронних мереж 40
4 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕХАНІЗМУ СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ 43
4.1 Характеристика архітектури bmshj2018-hyperprior, яка була обрана для проведення експериментів по стисненню зображень 43
4.2 Характеристика тестового набору зображень 44
4.3 Обґрунтування вибору програмного забезпечення та моделі ШНМ 46
4.4 Технічна реалізація обґрунтованих підходів 47
4.5 Аналіз результатів роботи ШНМ в порівнянні із стандартом JPEG 55
ВИСНОВКИ 61

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 62
ДОДАТОК А СЛАЙДИ ПРЕЗЕНТАЦІЇ 65
ДОДАТОК Б ТЕКСТ ПРОГРАМИ 74

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ANN (Artificial Neural Network) – Штучна нейронна мережа.

MLP (Multi-Layer Perceptrons) – Багатошарові перцептрони.

AE (Autoencoder) – Автокодер.

CNN (Convolutional Neural Network) – Згорткова нейронна мережа.

3D-CNN – Тривимірна згорткова нейронна мережа.

RNN (Recurrent Neural Network) – Рекурентна нейронна мережа.

GAN (Generative Adversarial Network) – Генеративно-змагальна мережа.

LSTM (Long Short-Term Memory) – Довга короткочасна пам'ять.

AE (Arithmetic Encoder) – Арифметичний кодер (у контексті ентропійного кодування на схемі).

AD (Arithmetic Decoder) – Арифметичний декодер.

Q (Quantization) – Квантування.

GDN (Generalized Normalization Transformation) – Узагальнене нормалізаційне перетворення.

IGDN (Inverse Generalized Normalization Transformation) – Обернене узагальнене нормалізаційне перетворення.

ReLU (Rectified Linear Unit) – Функція активації «випрямлена лінійна одиниця».

JPEG (Joint Photographic Experts Group) – Стандарт стиснення зображень (назва комітету розробників).

RD / RDO (Rate-Distortion Optimization) – Оптимізація залежності швидкості (бітрейту) від спотворення.

PSNR (Peak Signal-to-Noise Ratio) – Пікове співвідношення сигналу до шуму (метрика якості).

DCT (Discrete Cosine Transform) – Дискретне косинусне перетворення.

DPCM (Differential Pulse Code Modulation) – Диференціальна імпульсно-кодова модуляція.

CUDA (Compute Unified Device Architecture) – Архітектура паралельних обчислень єдиного пристрою.

AVX2 (Advanced Vector Extensions 2) – Розширені векторні розширення.

CPU (Central Processing Unit) – Центральний процесор.

BPP (Bit Per Pixel) – Біт на піксель.

PSNR (Peak Signal-to-noise Ratio) – Пікове відношення сигналу до шуму.

ВСТУП

Алгоритми кодування зображень створюють компактні цифрове представлення зображення, використовуючи просторову надмірність та перцептивну (зорову) нерелевантність, спираючись на особливості зорової системи людини. Останнім часом методи, що базуються на нейронних мережах, привернули підвищену увагу та стали популярною сферою досліджень і розробок. Цей інтерес зумовлений кількома факторами: нещодавнім прогресом у обчислювальній потужності, наявністю великих наборів даних, а також алгоритмічними та архітектурними досягненнями як приклад, згорткові шари.

Сьогодні нейронні мережі є передовими у вирішенні багатьох завдань комп'ютерного зору –тих, що потребують високорівневого розуміння семантики контенту (класифікація зображень, сегментація об'єктів, виявлення значущих областей), так і низькорівневих завдань обробки зображень (усунення шуму, реставрація фрагментів та забезпечення надвисокої роздільної здатності). Ці успіхи призвели до зростання інтересу до застосування глибоких нейронних мереж у кодуванні зображень, внаслідок чого нещодавно було запропоновано багато нових рішень.

Особливий інтерес представляє розробка нейронних мереж для завдань стиснення зображень. Замінюючи традиційні методи, такі як JPEG та JPEG 2000, передові моделі проектування автоматично знаходять ефективні способи представлення візуальної інформації в компактному форматі. Використання автокодерів та інших архітектур нейронних мереж дозволяє форматувати латентний простір, при якому зображення подається зі значно зменшеною розмірністю даних.

У цій кваліфікаційній роботі розглядаються сучасні методи стиснення зображень на основі штучних нейронних мереж, аналізуються принципи їх роботи.

1 ТЕОРЕТИЧНІ ОСНОВИ, ПРИНЦИП РОБОТИ, МЕТОДИ НАВЧАННЯ ТА ВИДИ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1 Теоретичні основи штучних нейронних мережі

Базовим структурним елементом будь-якої нейронної мережі є штучний нейрон, який моделює роботу біологічного нейрона. Математично штучний нейрон складається з трьох основних компонентів: множини вхідних сигналів, відповідних їм вагових коефіцієнтів, суматора та функції активації. На рис. 1.1 зображена структура біологічного нейрона. Вхідний сигнал надходить до клітини, потім надходить до пресинаптичної області через синаптичні з'єднання (нейрити).

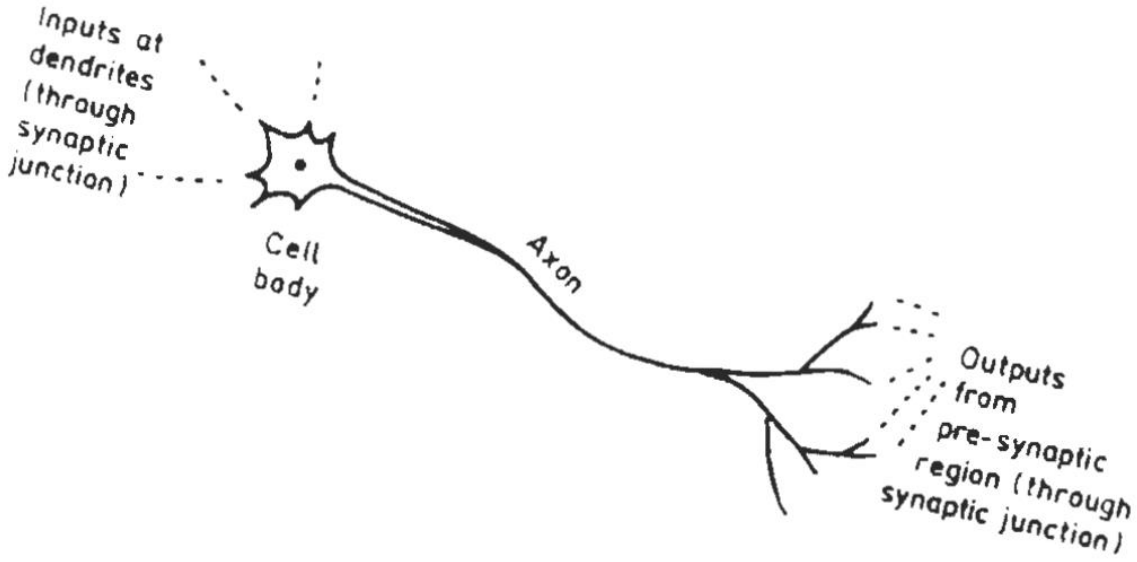


Рисунок 1.1 - Структура вхідних та вихідних сигналів біологічного нейрона [1]

На рисунку 1.2 зображена математична модель біологічного нейрона, перцептрона. Нехай нейрон має M входів, які позначаються вектором $x = [x_1, x_2, ..., x_M]^T$. Кожному входу відповідає ваговий коефіцієнт w_i, який визначає силу чи значущість даного входу для нейрона [1].

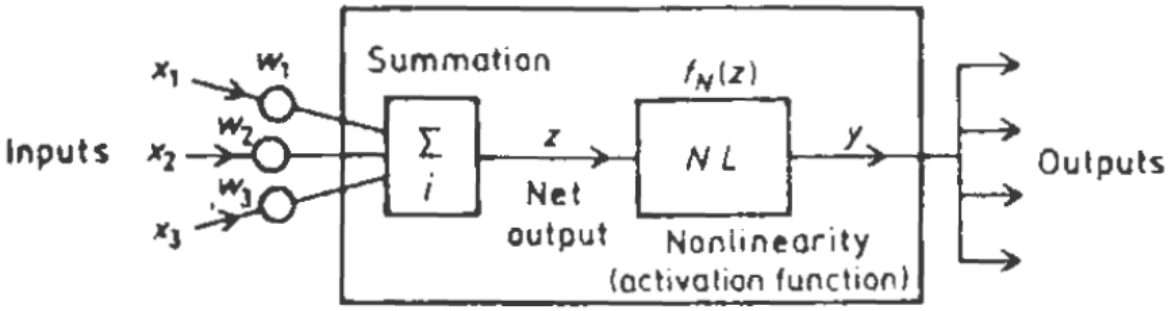


Рисунок 1.2 - Схематична структура вводу/виводу перцептрона [1]

Сукупність ваг формує вектор ваг w = $[w_1, w_2, ..., w_M]^T$. Процес обчислення всередині нейрона починається з визначення зваженої суми вхідних сигналів, який називається активаційним потенціалом z.

$$f_n(z) = \sum_{i=1}^{M} w_i x_i \tag{1.1}$$

Після цього отримане значення z передається у функцію активації *f(z)*, яка формує остаточний вихідний сигнал нейрона *y = f(z)*. До активаційних функцій можна віднести.

1. Гіперболічний тангенс

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{1.2}$$

Значення мають діапазон від -1 до 1, що робить їх центрованими відповідно до нуля [1]. Ця наближеність до нуля виступає у ролі нормалізації, що запобігає зсуву, нейрони оновлюють свої ваги рівномірно без застивань у одній площині значень.

2. Сигмоїдальна функція

$$f(z) = \frac{1}{1+e^{-z}}, \tag{1.3}$$

$$f'(z) = f(z)\big(1 - f(z)\big). \tag{1.4}$$

Вихід цієї функції завжди варіюється між 0 та 1, через це функція має плавний перехід на всій своїй протяжності. Такий діапазон значень дає нейромережі змогу робити чіткі обчислювальні операції [1].

3. Основною для багатошарових нейронних мереж є випрямлена лінійна одиниця (ReLU), яка є стандартом для глибокого навчання завдяки своїй простоті [1]

$$f(z) = \max(0, z). \tag{1.5}$$

На відміну від описаних вище функцій, ця функція робить лише операцію порівняння, що спрощує її обробку. Похідна цієї функції завжди буде *x>0*, і дорівнює *1*, що дозволяє градієнтам ефективно проходити крізь глибокі шари нейромережі.

1.2 Принцип роботи штучної нейронної мережі

Процес прямого поширення сигналу (Forward Propagation) полягає у послідовному обчисленні вихідних значень кожного шару на основі значень попереднього. Нехай шар l містить n_l нейронів, а попередній шар нейронів,

$$l - 1 - n_{l-1}, \tag{1.6}$$

тоді матриця вагових коефіцієнтів $W^l \in R^{n_l * n_{l-1}}$, вектор зміщень $b^l \in R^{n_l}$, вектор вихідних сигналів попереднього шару $z^{l-1} \in R^{n_l - 1}$, а вектор активаційних потенціалів $u^l \in R^{n_l}$. На кожному прихованому або вихідному шарі обчислюється зважена сума [2]:

$$u^l = W^l * z^{l-1} + b^l, \tag{1.7}$$

де W^l – матриця вагових коефіцієнтів шару l, b^l – вектор зміщень цього шару, a z^{l-1} – вектор вихідних сигналів попереднього шару [2]. Для першого прихованого шару $z^0 = x$ (вхідний вектор). Після обчислення зваженої суми до кожного елемента вектора u^l застосовується функція активації [2]

$$z_i^l = f\left(u_i^l\right),\ i = 1\ ,2\ ,3\ , \dots . \tag{1.8}$$

Отриманий вектор z^l є вихідним сигналом поточного шару та використовується як вхідний вектор для наступного шару мережі [2]. Цей процес послідовно повторюється для всіх прихованих шарів, доки не буде обчислено вектор вихідних значень вихідного шару z^L , який являє собою кінцевий результат роботи нейронної мережі [2].

1.3 Методи навчання штучних нейронних мереж

За методами навчання нейромереж можна виділити:

1. Навчання з учителем (Supervised Learning). Мережі надається навчальна вибірка, яка містить як вхідні дані, так і відповідні їм правильні відповіді (цільові значення або мітки) [2]. Мережа порівнює свої передбачення з еталоном і коригує параметри для зменшення розбіжності.

2. Навчання без учителя (Unsupervised Learning). Навчальна вибірка містить лише вхідні дані без будь-яких еталонних відповідей або міток[2]. Завдання мережі полягає в самостійному виявленні внутрішньої структури даних, кластеризації або зниженні розмірності.

3. Навчання з підкріпленням (Reinforcement Learning). Базується на взаємодії нейромережі із середовищем [2]. Мережа виконує певні дії та отримує винагороду або штраф залежно від результату. Метою є максимізація сумарної винагороди протягом тривалого часу.

4. Глибоке навчання (Deep Learning). Глибоке навчання є окремим напрямом машинного навчання, який використовує багатошарові нейронні мережі для автоматичного виділення ознак із даних [3]. Глибокі нейронні мережі

можуть містити десятки або навіть сотні шарів, що дозволяє їм виявляти складні закономірності.

Здебільшого не можна вважати ці методи окремими один від одного, бо в залежності від обраної моделі вони можуть поєднуватись, або змінюватись з одного на інший.

1.4 Види глибоких штучних нейронних мереж

1.4.1 Рекурентні нейронні мережі (RNN)

Цей вид архітектури нейронної мережі, який в основному використовується для виявлення закономірностей у послідовності даних. Такі дані можуть бути рукописним текстом, машинним текстом або числовими рядами, які часто створюються в промислових умовах [4]. Цю архітектуру також можна застосувати до зображень, мовного моделювання, генерації тексту, розпізнавання мови. Основною відмінністю рекурентних нейронних мереж (RNN) від багатошарових перцептронів (MLP) є те, як вони передають інформацію через себе. У той час як мережі прямого зв'язку передають інформацію через мережу без циклів, RNN має цикли та передає інформацію назад у себе [4]. Це дозволяє розширити функціональність мереж прямого зв'язку, що також дозволяє враховувати попередні вхідні дані $X_{0:t-1}$, а не лише поточні вхідні дані X_t [4].

1.4.2 Згорткові нейронні мережі (CNN)

Згорткові нейронні мережі (CNN) аналогічні традиційним MLP, оскільки вони складаються з нейронів, які самооптимізуються шляхом навчання. Кожен нейрон все одно отримуватиме вхідні дані та виконуватиме операцію (наприклад, скалярний добуток, а потім нелінійну функцію) [5]. Від вхідних векторів інформації до кінцевого виходу оцінки класу, вся мережа все ще виражатиме одну перцептивну функцію оцінки (вагу) [5].

Єдина помітна відмінність між CNN та традиційними MLP полягає в тому, що CNN в основному використовуються в галузі розпізнавання образів на

зображеннях [5]. Це дозволяє нам кодувати специфічні для зображення особливості в архітектуру, роблячи мережу більш придатною для завдань, орієнтованих на зображення, одночасно зменшуючи параметри, необхідні для налаштування моделі.

1.4.3 Генеративно-змагальні мережі (GAN)

Являє собою 2 штучні нейронні мережі на основі багатошарових перцептронів. Суть архітектури полягає в тому, що одна мережа (генератор) створює штучні дані, намагаючись зробити їх невідмінними від реальних, а друга мережа (дискримінатор) намагається відрізнити підробку від оригіналу [6]. В ході цього суперництва обидві мережі постійно вдосконалюються».

2 ТЕОРЕТИЧНІ ОСНОВИ СТИСНЕННЯ ЗОБРАЖЕНЬ ТА МЕТОДІВ МАШИННОГО НАВЧАННЯ

2.1 Проблематика стиснення зображень

У сучасному цифровому вимірі зображення займають значну частину інформаційного трафіку. Вони активно використовуються в телекомунікаційних системах, мультимедійних сервісах, соціальних мережах, медичних системах, супутниковому зв’язку та багатьох інших галузях. Проте збільшення роздільної здатності зображень призводить до суттєвого збільшення обсягів даних, що створює проблему їх ефективного зберігання та передавання.

Сире (нестиснене) цифрове зображення містить величезну кількість надлишкової інформації. Наприклад, сусідні пікселі часто мають дуже схожі значення кольору, що означає наявність просторової кореляції. Крім того, людська зорова система не однаково чутлива до всіх деталей зображення, що відкриває можливість видалення частини інформації без суттєвої втрати якості.

Основною метою стиснення є зменшення обсягу даних при збереженні прийнятного рівня якості. При цьому існують два основних підходи: стиснення без втрат та стиснення з втратами. У першому випадку відновлене зображення повністю збігається з оригіналом, що важливо для медичних або технічних застосувань. У другому випадку допускається певна втрата інформації заради значно більшого коефіцієнта стиснення.

Класичні методи стиснення, такі як JPEG, базуються на математичних перетвореннях, квантуванні та ентропійному кодуванні. Вони ефективно використовують статистичні властивості зображень, однак мають обмеження через появу артефактів, втрат деталей при сильному стиснені та недостатньої адаптивності до складних структур зображень.

JPEG [13] є найпопулярнішим стандартом стиснення зображень, який складається з базових модулів перетворення/прогнозування, як показано на рис. 2.1. У JPEG вхідне зображення розділяється на неперекриваючі

блоки 8×8, кожен з яких перетворюється в частотну область за допомогою блочного DCT (BDCT). Для кожного перетвореного блоку коефіцієнти DCT потім піддаються стисненню в двійковий потік за допомогою квантування та ентропійного кодування.

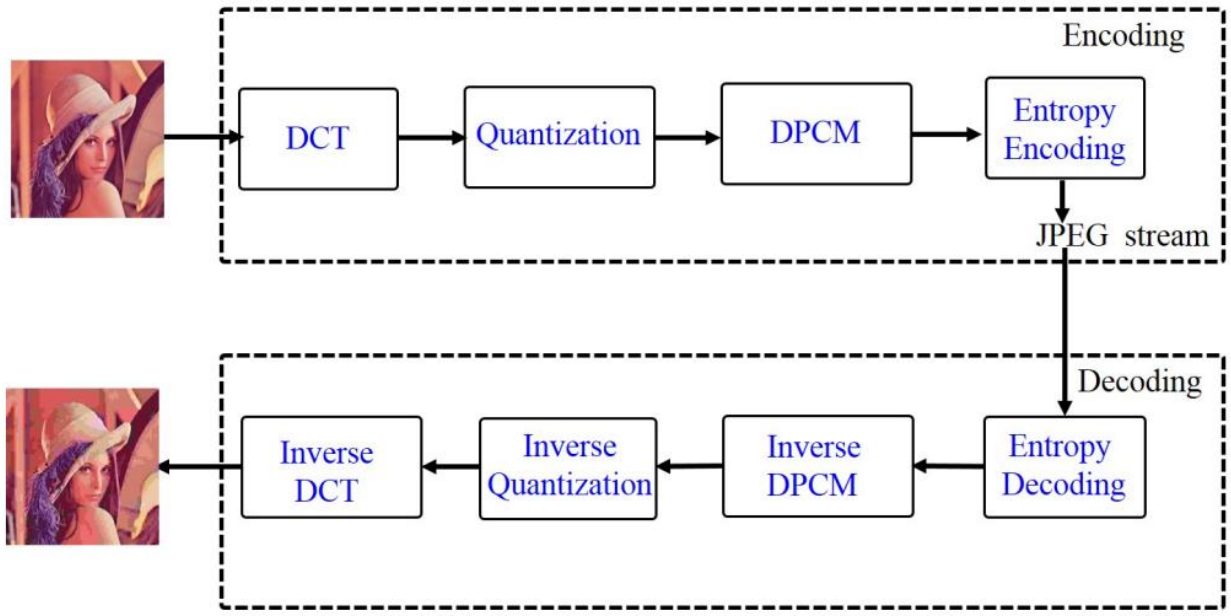


Рисунок 2.1 – Схематичне зображення процесу стиснення зображень JPEG [12]

Таким чином, проблема стиснення зображень полягає у знаходженні балансу між ступенем зменшення обсягу даних, обчислювальною складністю та якістю відновленого зображення. З розвитком технологій та збільшенням вимог до якості мультимедіа класичні підходи вже не завжди задовольняють сучасні потреби.

2.2 Перехід до машинного навчання

Обмеження традиційних методів стиснення стимулювали розвиток нових підходів, зокрема методів, заснованих на машинному навчанні. На відміну від класичних алгоритмів, які використовують заздалегідь визначені математичні моделі, методи машинного навчання здатні автоматично знаходити оптимальні способи представлення даних на основі великої кількості прикладів.

Ключова ідея полягає в тому, що нейронні мережі можуть навчатися виділяти найбільш важливі ознаки зображення та формувати компактне представлення, яке зберігає суттєву інформацію. Це дозволяє більш ефективно усувати як статистичну, так і структурну надлишковість, враховуючи складні залежності між пікселями.

Особливе місце в цій області займають автокодери, які складаються з кодера та декодера, та використовуються нейронними мережами. Кодер виконує стиснення вхідного зображення у компактний латентний простір (приховане представлення), а декодер відновлює зображення з цього представлення. Під час навчання мережа мінімізує різницю між оригінальним та відновленим зображенням, поступово навчаючись ефективному стисненню.

Перехід до машинного навчання дозволив досягти кращих результатів у порівнянні з класичними методами, особливо за високих ступенів стиснення. Такі методи демонструють зменшену кількість візуальних артефактів і краще зберігають деталі. Крім того, вони є більш гнучкими, оскільки можуть адаптуватися до різних типів зображень (природні сцени, медичні дані, текстури тощо).

Проте цей підхід має і свої недоліки: висока обчислювальна складність, необхідність великої кількості навчальних даних та значні ресурси для навчання моделей. Незважаючи на це, машинне навчання поступово стає одним із основних напрямів розвитку технологій стиснення.

2.3 Узагальнена концепція стисення зображень за допомогорю машинного навчання

2.3.1 Нелінійні перетворення

В [7] було розроблено цілісний (наскрізний) фреймворк для оптимізації та оцінки ефективності нелінійних перетворень. Система підтримує будь-які перцептивні метрики та передбачає використання скалярного квантування. Завдяки цьому фреймворку можна визначити оптимальні параметри для

узагальненого нормалізаційного перетворення (GDN) та відповідного йому оберненого процесу.

Аналіз результатів показав, що таке оптимізоване перетворення значно ефективніше зменшує кореляцію між компонентами порівняно з альтернативними методами. Крім того, дослідження оборотності цих операцій дозволило розробити алгоритми навчання, які мінімізують функцію втрат, гарантуючи максимально можливу точність відтворення зображення.

2.3.2 Квантування

Замість розробки окремої моделі для кожного рівня якості, сучасні підходи пропонують навчити одну базову модель, до якої згодом застосовуються різні стратегії, як-от адаптивні або вивчені розміри кроків квантування.

Інший напрямок фокусується на агресивному скороченні прихованого коду для стимулювання розрідженості даних [8]. Також ефективною є процедура неоднорідного квантування, оптимізована під конкретний розподіл ознак, в такому випадку мережа кодер-декодер налаштовується ітеративно разом із квантувачем для досягнення максимальної продуктивності.

Окрему увагу приділяють єдиним системам наскрізного навчання. Вони дозволяють спільно оптимізувати параметри моделі, рівні квантування та ентропію вихідного потоку символів. Такий метод базується на «м’якому» присвоєнні даних рівням квантування, які формуються безпосередньо під час навчання моделі.

2.3.3 Ентропійне кодування

Ключова ідея сучасних методів полягає у використанні статистики латентного представлення за допомогою моделі умовної ймовірності. Зокрема, застосовується архітектура автокодера з 3D-CNN для моделювання контексту, де модель ітеративно оновлюється під час навчання, щоб виявити залежності між елементами латентного коду.

Для підвищення точності використовуються просторово локальні та адаптивні моделі, які спільно працюють у кодері та декодері [9]. Завдяки передачі

певної додаткової інформації такі моделі адаптуються до характеристик кожного конкретного зображення.

Подальший розвиток технології призвів до появи контекстно-адаптивних структур, здатних оперувати різними типами контекстів одночасно [9]. Вибір та комбінування декількох типів контексту дозволяє точніше визначити ймовірність розподілу елементів коду, що забезпечує надзвичайно ефективне використання просторової кореляції.

На рисунку 2.2 наведене концептуальне зображення моделі, яку використовують при стисненні зображень.

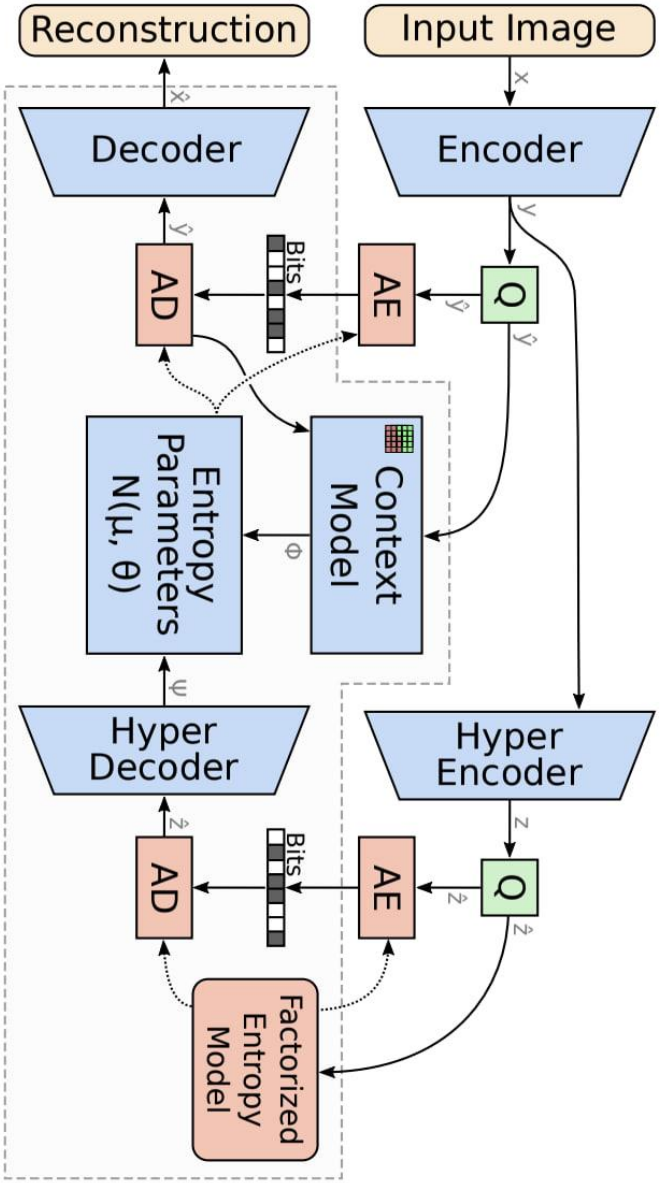


Рисунок 2.2 – Ілюстрація архітектури моделі із нелінійним перетворенням яка виконує операцію квантування Q та ентропійне кодування за допомогою блоків AE та AD [10]

2.4 Класифікація параметрів які важливо розуміти при побудові системи обробки зображень за допомогою машинного навчання

2.4.1 Тип нейронної мережі

При виборі типу штучної нейронної мережі найбільш релевантним рішенням є автокодери, які здатні витягувати компактне представлення зображення та можуть навчатися нелінійним перетворенням кодера-декодера, здатним ефективно декорелювати візуальну інформацію [11]. Більше того, важливий не тільки високорівневий опис нейронної мережі, але й те, який варіант автокодеру використовується (варіаційний автокодер) та тип шарів, що використовуються як у кодері, так і в декодері (згортковий, рекурентний). Окрім того, стратегія, якої дотримується штучна нейронна мережа під час процесу навчання, може також допомогти охарактеризувати рішення для процесу стиснення зображень; це може бути проста функція втрат (PNSR) або більш складні стратегії з двома нейронними мережами, що застосовуються ітеративно, наприклад, у генеративно-змагальних мережах (GAN).

2.4.2 Розмір одиниці кодування

Параметр розміру одиниці кодування є досить важливим параметром для кодування зображень і має критичний вплив на релевантність та продуктивність рішення для кодування [11]. Перш за все, оскільки рішення для кодування не використовують кореляцію між одиницями кодування, забезпечується певний ступінь просторового довільного доступу [11]. Окрім того, архітектура нейронної мережі зазвичай адаптована для певного розміру одиниці кодування, що означає, що зображення з роздільною здатністю, яка не кратна цьому розміру, потребує адаптації, наприклад, шляхом доповнення або певного типу адаптації роздільної здатності, що може вплинути на ефективність стиснення.

2.4.3 Механізми просторової кореляції

Ці механізми будуються на використанні просторової кореляції для створення більш компактного представлення зображення [11]. У літературі існує

кілька підходів, які можна залучати для використання просторової кореляції в піксельному домені або в деякому іншому просторі ознак, таких як використання гіперапріорних змінних, які відповідають деякій додатковій інформації, що передається від кодера до декодера для отримання точніших моделей ентропійного кодування [11]. Іншим релевантним інструментом у цьому вимірі є оцінка моделей ймовірності ентропійного кодування з використанням інформації, що декодується раніше.

2.4.4 Стратегія керування бітрейтом

Цей параметр характеризує метод кодування залежно від прийнятого типу керування бітрейтом, а саме, чи модель навчається для кожної точки RD (Rate-Distortion) з використанням заданого кроку квантування (фіксованого), або використовує гнучкий метод зміни кроку квантування [11]. Це важливий фактор, оскільки, якщо моделі потрібно вивчити цільову якість, яку, потім треба застосувати під час кодування зображень, кількість пропонованих якостей декодування може бути занадто малою порівняно з більш традиційними рішеннями для кодування зображень [11]. Це також може вплинути на вимоги до пам’яті кодера та декодера (якому, можливо, доведеться зберігати велику вивчену модель для кожної якості декодування).

2.5 Принцип роботи методів машинного навчання для стиснення зображень

Принцип розробки методів машинного навчання полягає у стисненні зображень із знаходженням найбільш ефективного способу представлення їх у компактній формі без явного задання правил стиснення. На відміну від класичних алгоритмів, де всі етапи (перетворення, квантування, кодування) жорстко визначені, у машинному навчанні модель самостійно навчається виконувати ці операції, оптимізуючи їх під конкретні дані.

Основним принципом є навчання на великій вибірці зображень. Модель отримує на вхід оригінальне зображення та намагається відтворити його після стиснення. У процесі навчання вона мінімізує різницю між початковим і

відновленим зображенням. Таким чином, система поступово «розуміє», які частини зображення є важливими, а які можна спростити або відкинути.

Найбільш типовою архітектурою для таких задач є автокодер. Його робота базується на трьох ключових етапах:

1. Кодування (encoding) – зображення перетворюється у компактне представлення (латентний простір). На цьому етапі відбувається фактичне стиснення.
2. Зберігання або передача – використовується вже не саме зображення, а його стиснене представлення.
3. Декодування (decoding) – з латентного представлення відновлюється зображення, максимально наближене до оригіналу.

Ключовою ідеєю є те, що нейронна мережа не просто виконує стиснення зображень, а виділяє їхню структуру: контури, текстури, повторювані елементи та інші закономірності. Це дозволяє досягати кращого стиснення, ніж у класичних методах, особливо для складних зображень.

Важливу роль відіграє функція втрат, яка визначає якісне відновлення. Це може бути піксельна різниця (наприклад, середньоквадратична помилка), або більш складні метрики, що враховують сприйняття людиною (структурна схожість). Таким чином, модель оптимізується не просто на точне відтворення, а на візуальну якість.

Ще одна важлива особливість – адаптивність. Якщо класичні алгоритми однакові для всіх типів зображень, то модель машинного навчання може бути навчена під конкретний тип даних (медичні знімки або фотографії природи), що значно підвищує ефективність.

У загальному вигляді суть алгоритма можна звести до трьох ідей:

1. Навчання замість ручного проєктування алгоритму.
2. Перетворення зображення у компактний латентний простір.
3. Оптимізація якості відновлення через функцію втрат.

У підсумку, методи машинного навчання розглядають стиснення не як набір фіксованих операцій, а як задачу оптимізації, де система сама знаходить найкращий компроміс між розміром даних і якістю зображення.

3 ОСНОВНІ ПІДХОДИ ДО СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

3.1 Багатошарове кодування зображень на основі перцептрона

MLP [14] складається з вхідного шару нейронів (або вузлів, одиниць), кількох прихованих шарів нейронів та останнього шару вихідних нейронів. Вихідний сигнал h_i кожного нейрона,

$$h_i = f(\sum_{j=1}^{N} w_{ij} x_j + b_i), \quad (3.1)$$

де *f()* - функція активації, b_i позначає зміщення лінійного перетворення, а w_{ij} вказує на регульований параметр, вагу, який представляє зв'язок між шарами. Теоретичний аналіз показав, що MLP, побудований з більш ніж одним прихованим шаром, може апроксимувати будь-яку неперервну обчислювальну функцію з довільною точністю [15]. Ця властивість надає підґрунття для таких сценаріїв, як зменшення розмірності та стиснення даних. Ініціатива використання MLP для стиснення зображень полягає в розробці унітарних перетворень для всіх просторових даних.

Адаптивність цього алгоритма визначається ручним встановленням різної кількості прихованих нейронів, а не використанням мереж з більшою кількістю шарів та складними зв'язками, що може обмежувати потужність MLP з точки зору продуктивності стиснення [16]. Для вирішення цієї проблеми було досліджено алгоритм прогнозного кодування зображень на основі MLP [17], використовуючи інформацію про просторовий контекст. Як показано на рис. 3.1, модель MLP використовує три вхідні вузли, 30 прихованих вузлів та один вихідний вузол для предиктора, а сама модель навчається за допомогою алгоритму зворотного поширення [18] для мінімізації середньоквадратичних помилок між вихідними та прогнозованими сигналами.

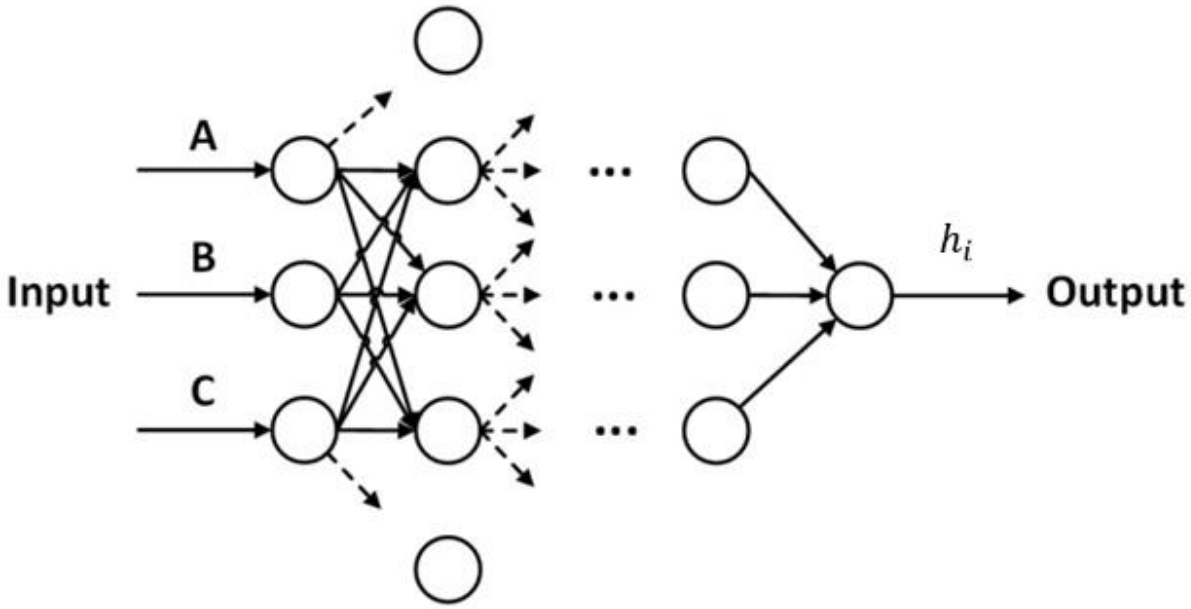


Рисунок 3.1 – Схематичне зображення прогнозного кодування на основі MLP для стиснення зображень [17]

3.2 Стиснення зображень із використанням випадкової нейронної мережі (Random Neural Network)

Цей клас нейронної мережі [19] був представлений у 1989 році. Випадкова нейронна мережа працює інакше, сигнали у випадковій нейронній мережі передаються у вигляді піків одиничної амплітуди. Зв'язок між цими нейронами моделюється як процес Пуассона, де позитивні сигнали представляють збуджуючі сигнали, а негативні сигнали представляють гальмування. Деякі теоретичні результати були представлені для аналізу поведінки випадкової нейронної мережі в [19]. Для оновлення параметрів використовується метод навчання типу «зворотного поширення», що вимагає розв'язання n лінійних та n нелінійних рівнянь кожного разу з новою парою вхід-вихід. Деякі дослідники розглядали комбінацію випадкової нейронної мережі та стиснення зображень і представили деякі значущі результати. Геленбе та ін. вперше застосували випадкову нейронну мережу в задачі стиснення зображень [20]. Архітектура

використовує випадкову нейронну мережу кодера/декодера прямого зв'язку з одним проміжним шаром. Зокрема, перший шар приймає зображення на вхід, останній шар видає реконструйоване зображення, а проміжний шар створює стиснені біти.

3.3 Стиснення зображення із використанням згорткової нейронної мережі

3.3.1 Основне теоретичне уявлення про CNN

Нещодавно CNN значно перевершував традиційні алгоритми у завданнях комп'ютерного зору високого рівня, таких як класифікація зображень, виявлення об'єктів [21]. Навіть для багатьох завдань комп'ютерного зору низького рівня моедль CNN досягає дуже вражаючої продуктивності, наприклад, суперроздільної здатності та зменшення артефактів стиснення. CNN використовує операцію згортки для характеристики кореляції між сусідніми пікселями, а каскадні операції згортки добре відповідають ієрархічним статистичним властивостям природних зображень. Крім того, локальні рецептивні поля та спільні ваги, введені операціями згортки, також зменшують параметри CNN, що підлягають навчанню, що значно знижує ризик проблеми переналаштування.

Однак важко безпосередньо інтегрувати модель CNN у наскрізне стиснення зображень. Загалом, навчання CNN залежить від алгоритму зворотного поширення та стохастичного градієнтного спуску, які вимагають майже скрізь диференційованості функції втрат відносно до параметрів, що формується під час навчання, наприклад, ваги згортки та зміщення. Через модуль квантування у стисненні зображень, він створює нульові градієнти майже скрізь, що зупиняє оновлення параметрів у CNN. Крім того, класичну оптимізацію швидкості та спотворень важко застосувати до структури стиснення на основі CNN.

3.3.2 Наскрізне стиснення зображень на основі CNN

Балле та ін. вперше представили оптимізовану наскрізну структуру CNN для стиснення зображень за припущення скалярного квантування у 2016 році [22], [7]. На рисунку 3.2 наведена схема яка зображає описану архітектуру. Вхідне зображення (input image) подається одночасно до мережі семантичного аналізу (semantic analysis network) та мережі стиснення (image compression network). Блок семантичного аналізу визначає важливі області зображення та формує рівень розподілу бітів для стиснення. Далі кодер (encoder) перетворює зображення у компактне представлення, бынаризатор (binarizer) переводить дані у бітовий потік, а декодер (decoder) відновлює зображення після стиснення. Завдяки семантичному аналізу система виділяє більше ресурсів для важливих частин зображення, що дозволяє покращити якість відновлення при високому рівні стиснення.

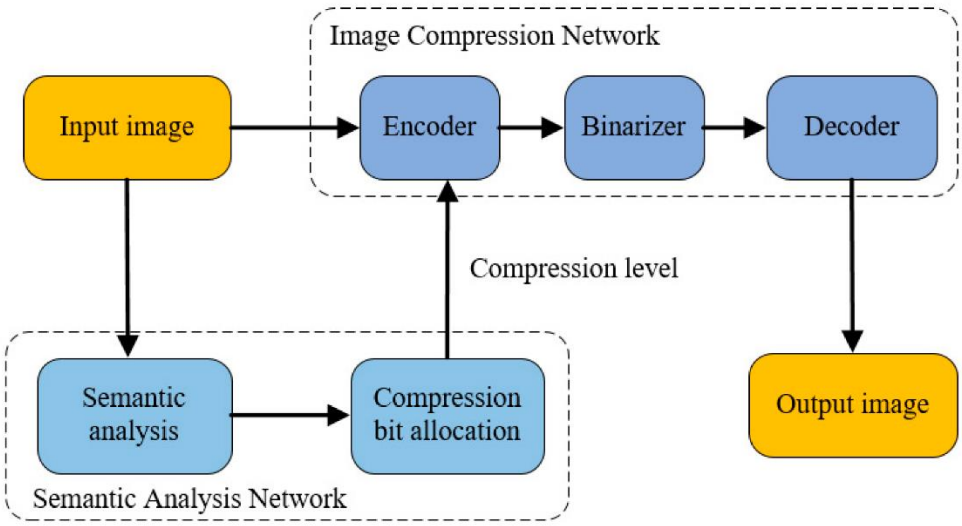


Рисунок 3.2 – Схематичне зображення архітектури наскрізного стиснення зображень на основі CNN [23]

В аналітичному перетворенні використовується афінна згортка, яка складається з двох послідовних операцій: лінійної згортки та додавання зсуву. Для кожного шару нейронної мережі обчислення вихідної ознаки здійснюється за формулою:

$$v_i^{(k)}(m,n) = \sum \left(w_{k,ij} * x_j^{(k)} \right) + b_{k,i}, \qquad (3.2)$$

де $x_j^{(k)}$ – j-й вхідний канал k-го каскаду в просторовій точці (m, n), $*$ позначає 2D операцію згортки, а $w_{k,ij}$ – параметр згортки. $b_{k,i}$ – параметр зміщення, а оператор * позначає двовимірну операцію згортки у просторовій точці (m,n).

Афінна складова ($b_{k,i}$) дозволяє здійснювати зсув значень ознак після згортки, що підвищує виразність моделі та забезпечує більшу гнучкість при навчанні. Таким чином, кожен вихідний канал формується як зважена сума згорток усіх вхідних каналів із подальшим додаванням зміщення.

3.3.3 Навчання згорткових мереж для стиснення зображень з вагою за змістом

Лі запропонував рішення для кодування зображень з вагою за змістом на основі згорткових нейронних мереж [24]. Основна ідея полягає у виконанні просторово-адаптивного розподілу бітрейту за допомогою карти важливості, яка вказує на важливі (виразні) з точки зору сприйняття людиною частини зображення. Крім того, для квантування використовується простий бінаризатор, карта важливості також застосовується для отримання точної оцінки бітрейтів під час навчання.

На рисунку 3.3 наведена схема яка зображає описану архітектуру. Основний кодер (encoder) виділяє глибокі просторові ознаки зображення, які блок бінаризації (binarizer) перетворює на повний тривимірний куб двійкових даних (binary feature map). Водночас мережа значущості (importance map network) визначає найбільш інформативні об'єкти на знімку й генерує карту важливості (importance map), яка після квантування (qiantization) перетворюється на просторову 3D-маску (mask). Далі відбувається поелементне множення куба ознак на цю маску (element-wise multiplication), що дозволяє відфільтрувати інформацію, дані про другорядний фон фактично відсікаються, а деталі ключових об'єктів залишаються недоторканими. Сформовані таким чином оптимізовані бінарні коди передаються на декодер (decoder), який розпаковує

зменшений потік даних та синтезує фінальне зображення, забезпечуючи високу чіткість у важливих зонах при мінімальному розмірі підсумкового файлу.

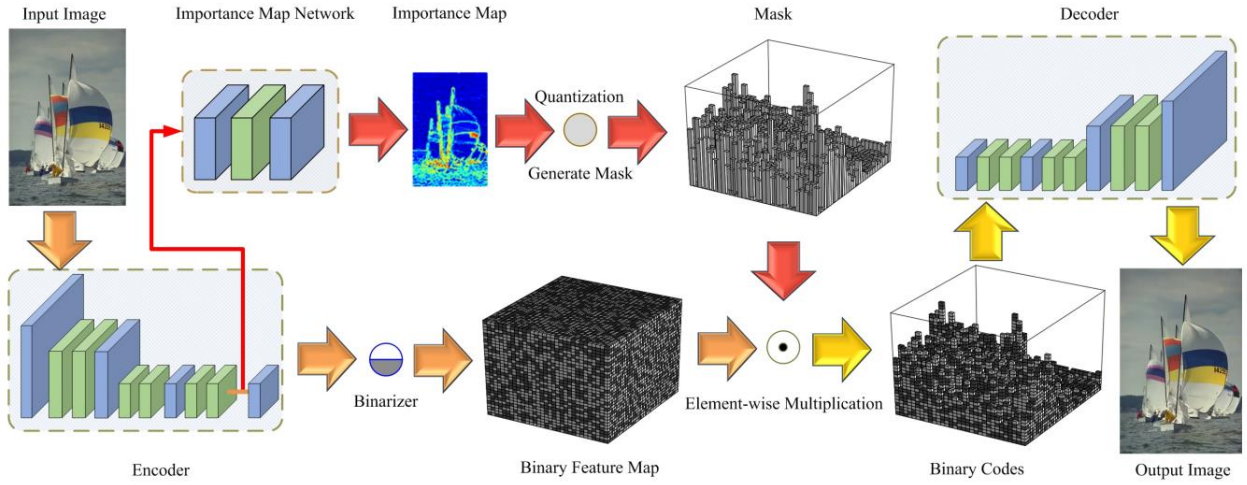


Рисунок 3.3 – Ілюстрація архітектури CNN для стиснення зображень з урахуванням ваги розрахованої за змістом зображення [24]

3.4 Стиснення зображення із використанням рекурентної нейронної мережі

3.4.1 Основне теоретичне уявлення про RNN

На відміну від архітектури CNN, згаданої вище, RNN – це клас нейронної мережі з пам'яттю для зберігання нещодавньої поведінки. Зокрема, блоки пам'яті в RNN мають з'єднання з собою, які передають перетворену інформацію з виконання в минулому. Блоки довгострокової пам'яті (LSTM) [25] були розроблені для належного вирішення проблеми зникнення градієнта. Завдяки підтримці постійного потоку помилки (градієнта) RNN навчатися протягом набагато більшої кількості кроків. Для досягнення цього LSTM зберігають більше інформації поза традиційним потоком нейронної мережі в структурах, які називаються гейтованими комірками [25].

Щоб LSTM працював, ми використовуємо вихідний шлюз O_t для зчитування записів комірки, вхідний шлюз I_t для зчитування даних у комірку та шлюз забуття F_t для скидання вмісту комірки [4]. На рисунку 3.4 зображена схема процесу описана формулами (3.3) – (3.5)

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_0), \tag{3.3}$$

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i), \tag{3.4}$$

$$F_t = \sigma\left(X_t W_{xf} + H_{t-1} W_{hf} + b_f\right). \tag{3.5}$$

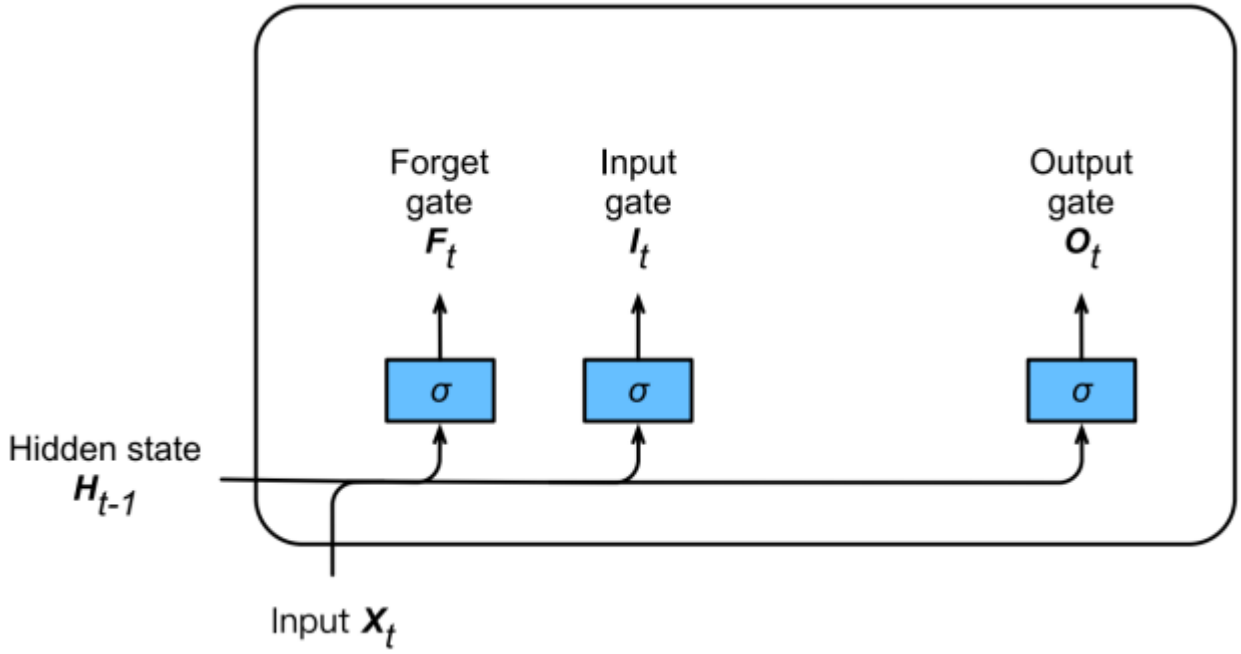


Рисунок 3.4 – Схематичне представлення першого етапа LSTM описаного формулами (3.3) – (3.5) [4]

У показаних рівняннях W_{xi}, W_{xf}, $W_{xo} \in R^{d\times h}$ та W_{hi}, W_{hf}, $W_{ho} \in R^{h\times h}$ використовуються як вагові матриці, де d – розмірність вхідного вектора, h – розмірність прихованого стану (кількість прихованих нейронів). При цьому b_i, b_f, $b_o \in R^{1\times h}$ – це їхні відповідні зміщення [4]. Крім того, вони використовують сигмоїдну активаційну функцію σ для перетворення виходу який приймає значення з інтервалу *(0, 1)*, кожне з яких призводить до вектора з елементами, що належать інтервалу *(0, 1)*. Далі, обчилюэться стан комірки-кандидат $\widehat{C_t} \in R^{n\times h}$($n$ позначає розмір міні-пакету даних), яка має подібне обчислення, як і раніше згадані вентилі, але замість цього використовує функцію активації tanh() для отримання виходу в інтервалі *(−1, 1)* [4]. Крім того, вона знову має власні ваги $W_{xc} \in R^{d\times h}$, $W_{hc} \in R^{h\times h}$ та зміщення $b_c \in R^{1\times h}$. На рисунку 3.5 зображена схема процесу, описана формулою (3.6)

$$\widehat{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c). \quad (3.6)$$

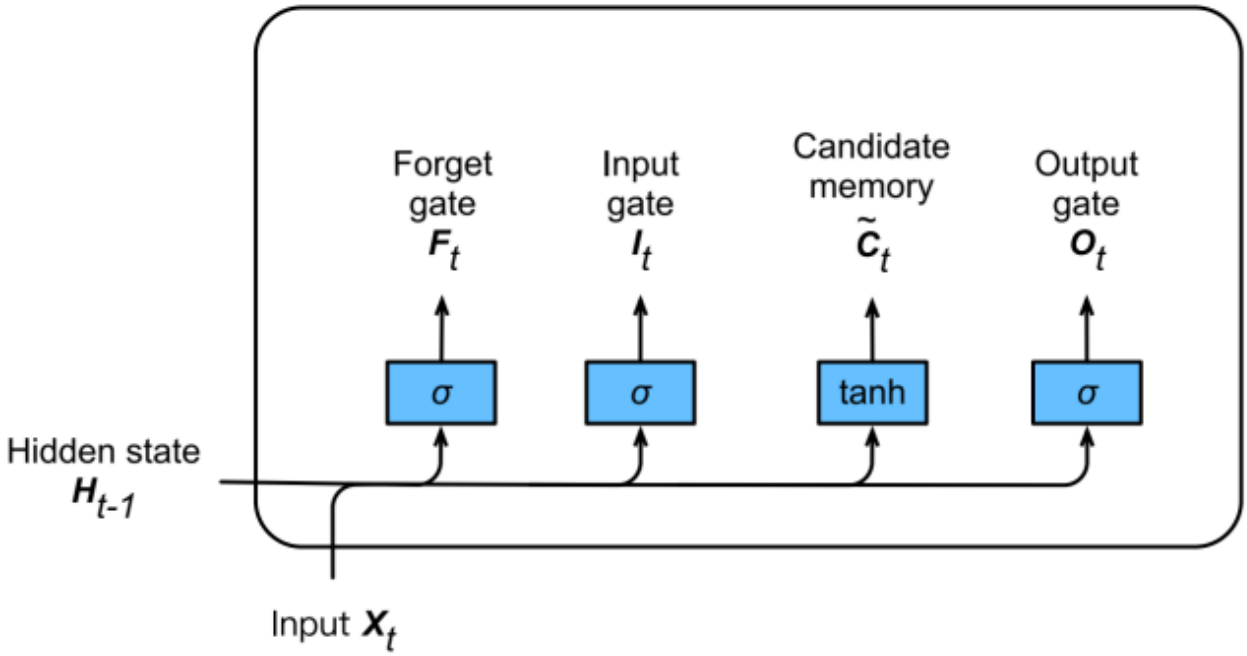


Рисунок 3.5 – Схематичне представлення другого етапа LSTM, описаного формулою (3.6) [4]

Щоб поєднати деякі речі, ми вводимо старий вміст пам'яті $C_{t-1} \in R^{n \times h}$, який разом із введеними вентилями контролює, скільки старого вмісту пам'яті ми хочемо зберегти, щоб дістатися до нового вмісту пам'яті C_t. На рисунку 3.6 зображена схема процесу описана формулой (3.7)

$$C_t = f_t * C_{t-1} + I_t * \widehat{C}_t. \quad (3.7)$$

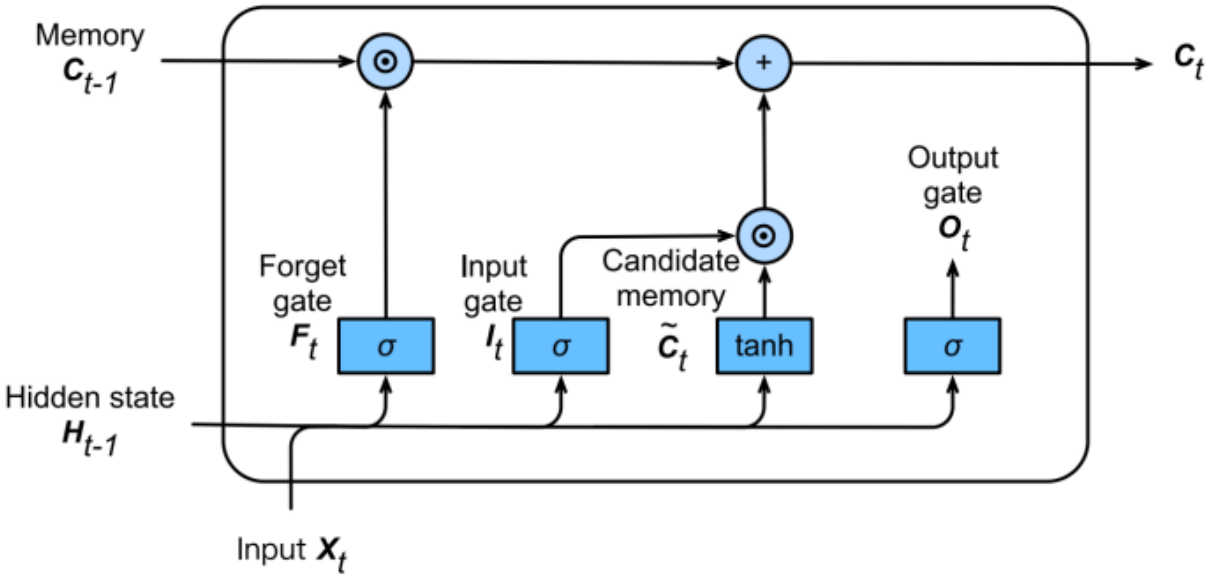


Рисунок 3.6 –Схематичне представлення третього етапа LSTM описаного формулою (3.7) [4]

Останнім кроком є введення обчислення для прихованих станів $H_t \in R^{n\times h}$ у структуру. Функції tanh() гарантує, що кожен елемент $H_t \in (-1, 1)$ На рисунку 3.7 зображена схема процесу описана формулой (3.8)

$$H_t = O_t * \tanh(C_t). \tag{3.8}$$

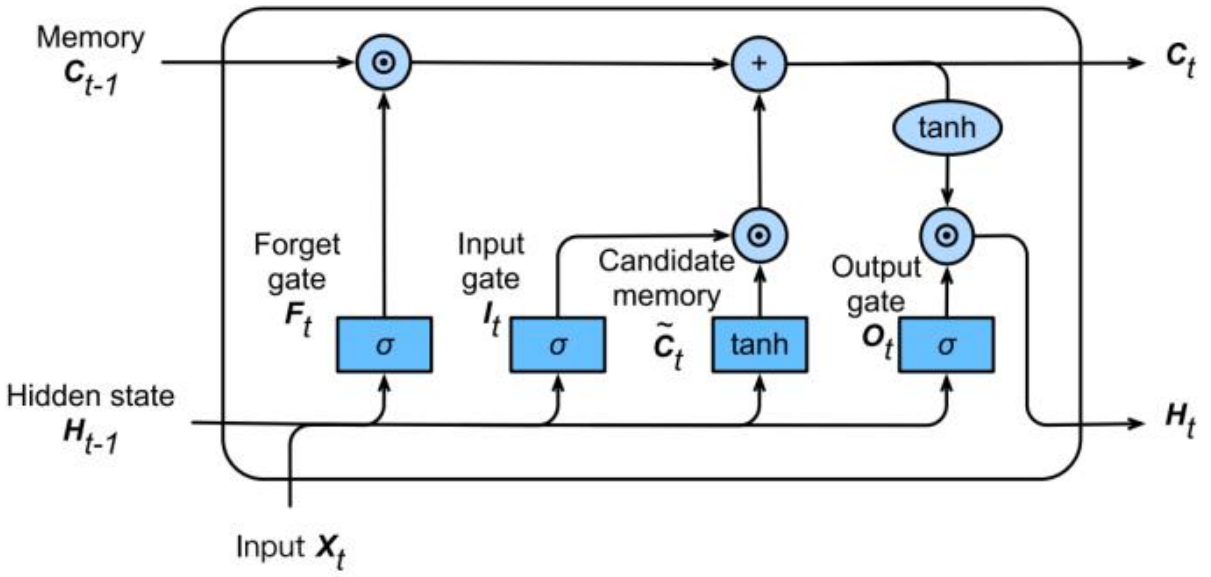


Рисунок 3.7 – Схематичне представлення фінального етапа LSTM, описаного формулою (3.8) [4]

3.4.2 Стиснення зображень зі змінною швидкістю за допомогою рекурентної нейронної мережі

Тодерічі у ранніх роботах з кодування зображень на основі навчання запропонував глибокі повністю зв’язані кодери з довгою короткочасною пам’яттю (LSTM) та згорткові/деконволюційні кодери залишків для стиснення мініатюрних зображень [26]. Розмірність латентного представлення значно зменшена порівняно з вхідним зображенням, а квантування виконується за допомогою методу прямої бінаризації. На рисунку 3.8 наведена схема яка зображає описну архітектуру, на кожен єтап кодування/декодування припадає 3 шари нейромережі по 512 каналів, в блоці «4 bits» відбувається квантування. Перша частина алгоритму виконує аналіз, кодування та розрахунок різниці із орігіналом. Друга частина виконує тіж самі дії, але рахує передбачення помилки реконструкції зображення.

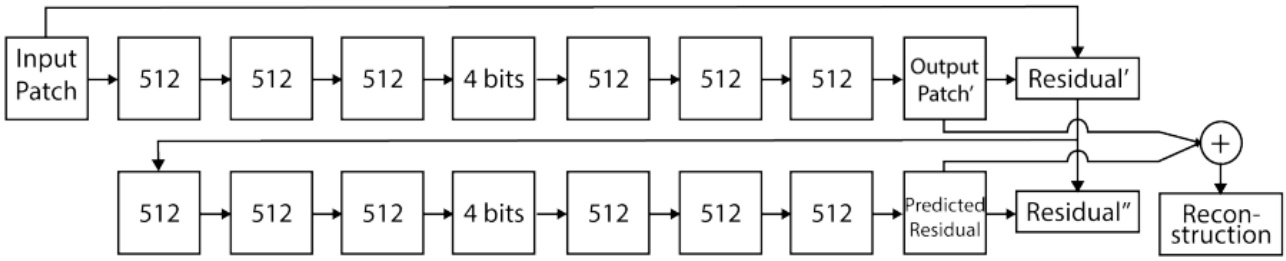


Рисунок 3.8 – Автокодер надлишку в якому зображена двоітераційна ахітектура [26]

3.4.3 Стиснення зображень з повною роздільною здатністю за допомогою рекурентних нейронних мереж

Запропоновані методи кодування зображень з втратами з повною роздільною здатністю, засновані на рекурентних нейронних мережах (RNN) - це було одне з перших рішень з конкурентоспроможною продуктивністю в широкому діапазоні бітрейтів для зображень довільних розмірів [27]. Рекурентний характер запропонованого рішення дозволяє поширювати

інформацію між ітераціями як кодера, так і декодера, поступово досягаючи кращих компромісів між цільовим бітрейтом та якістю.

На рисунку 3.9 наведена схема, яка зображає описану архітектуру. Вхідне зображення H×W×3 подається до блоку Encoder, який виконує аналіз і стиснення даних. Блок E-Conv є згортковим шаром, що виділяє основні ознаки зображення та зменшує його розмір. Далі блоки E-RNN#1 – E-RNN#3, які є рекурентними згортковими мережами, обробляють інформацію та формують компактне представлення зображення. Після цього формується Encoded (floats) – стиснений набір числових даних.

Далі дані передаються до блоку Binarizer. У ньому шар B-Conv готує інформацію до бінаризації, в наслідок якої формується бітовий код, який використовується для зберігання або передачі зображення.

Потім бітовий код надходить у Decoder, який виконує реконструкцію зображення. Блоки D-RNN#1 – D-RNN#4 поступово відновлюють структуру та деталі зображення. Операції Depth to Space збільшують роздільну здатність і повертають зображення до початкового розміру. Наприкінці шар D-Conv#2 формує фінальне відновлене зображення I_t розміру H×W×3.

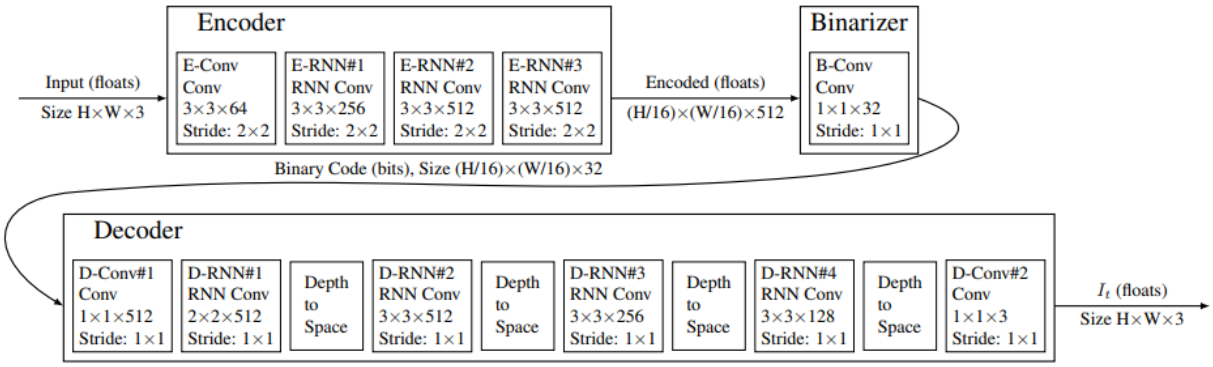


Рисунок 3.9 – Одна ітерація автокодера із архітектурою RNN [27]

3.5 Стиснення зображення із використанням генеративно-змагальних мереж

3.5.1 Основне теоретичне уявлення про GAN

Архітектура генеративно-змагальних мереж (GAN) базується на грі двох моделей, які зазвичай представлені багатошаровими перцептронами, генератора (G) та дискримінатора (D).

Щоб модель могла вивчити розподіл генератора p_g на основі реальних даних x, спочатку задається апріорний розподіл вхідних випадкових змінних (шуму) $p_z(z)$ [6]. Після цього визначається відображення у простір даних як G(z, θ_g), де G — диференційована функція, параметризована вагами θ_g. Другий багатошаровий перцептрон, дискримінатор $D(x, \theta_d)$, приймає на вхід вектор даних і повертає єдине скалярне значення. Параметр x представляє ймовірність того, що об'єкт x походить з реального розподілу навчальних даних p_{data}, а не з розподілу моделі p_g.

Під час навчання модель D максимізує ймовірність призначення правильної мітки як реальним прикладам, так і згенерованим зразкам від G. У той же час G навчається мінімізувати ймовірність того, що D розпізнає його підробку. Таким чином, D та G конкурують у мінімаксній грі для двох гравців із функцією вартості $V(G, D)$.

$$\min_G \max_D V(D,G) = E_{p_{data}(x)}[log\, D(x)] + E_{p_z(z)}\left[log\left(1 - D\big(G(z)\big)\right)\right], \quad (3.9)$$

де E позначає математичне сподівання випадкової величини, взятої за відповідним розподілом.

На практиці рівняння (3.9) може не забезпечити достатнього градієнта для успішного навчання G [6]. На ранніх етапах, коли функція G ще не навчена, дискримінатор D може відхиляти вибірки з високою впевненістю, оскільки вони явно відрізняються від навчальних даних. У цьому випадку значення log(*1 - D*(*G*(*z*))) насичується (градієнт прямує до нуля) [6]. Тому замість мінімізації функції *log*(*1 - D*(*G*(*z*))) краще навчити генератор G максимізувати

вираз $log\ D(G(z))$ [6]. Ця модифікована цільова функція приводить до тієї ж самої фіксованої точки динаміки моделей G та D, але забезпечує набагато сильніші градієнти на ранніх етапах навчання.

3.5.2 Адаптивне стиснення зображень у реальному часі

Ріппель та ін. запропонували архітектуру кодування зображень на основі генеративно-змагальних мереж (GAN) [28]. Найбільш вагомий внесок цієї роботи полягає у структурі навчання багатомасштабних змагальних мереж, яка поєднує втрати змагальних мереж із втратами реконструкції. Конструкція автокодера включає пірамідальний аналіз, а також адаптивне арифметичне кодування з використанням контекстних моделей для покращення показників обробки зображення.

На рисунку 3.10 наведена схема яка зображає описану архітектуру. Процес починається з вхідного зображення (Target), з якого блок Feature extraction виділяє ключові ознаки. Отримані дані проходять дискретизацію в блоці Quantization і піддаються стисненню за допомогою Coding у компактний бінарний потік (Bitstream). Паралельно блок Adaptive codelength regularization оцінює та мінімізує розмір цього потоку, відповідаючи за ступінь стиснення даних. Під час зворотного процесу блоки Decoding та Synthesis from features розпаковують біти й відновлюють з ознак фінальне зображення (Reconstruction).

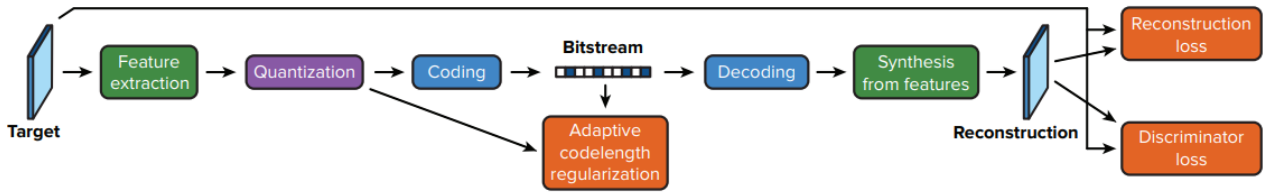


Рисунок 3.10 – Ілюстрація архітектури автокодера із адаптивним стисненням у реальному часі [28]

3.5.3 Генеративно-змагальні мережі для стиснення зображень з екстремально низькою роздільною здатністю

Агустссон запропонував рішення для глибокого кодування зображень на основі генеративно-змагальної мережі, призначене для дуже низьких бітрейтів (зокрема, нижче 0,1 біта на піксель), яке, попри це, дозволяє отримувати візуально приємні зображення з високою роздільною здатністю [29]. За таких бітрейтів важко зберегти багато візуальних елементів у зображенні, тому це рішення дозволяє генерувати штучні елементи, щоб зробити зображення приємнішим, використовуючи наявну додаткову інформацію (тобто семантичну інформацію про зображення, що кодується). На рисунку 3.11 наведена схема яка зображає описану архітектуру.

Принцип роботи цієї моделі полягає в наступному:

1. Вхідне зображення x, що походить з реального розподілу даних p_{data}, проходить через кодер E, який перетворює візуальні дані на компактне неперервне латентне представлення $z_e = E(x)$.

2. Отримане представлення обробляється квантучем q для формування дискретного стиснутого коду $\widehat{w} = q(Z_e)$, який оптимізує швидкість передачі даних (бітрейт).

3. До отриманого коду $\widehat{w}$ опціонально додається випадковий шум v, взятий з апріорного розподілу шуму $p_v(v)$, для формування фінального латентного вектора $z = \widehat{w} + v$.

4. Генератор G використовує латентний вектор z як вхідні дані для відображення назад у простір зображень. Він генерує відновлене зображення $\hat{x} = G(z)$, яке має виглядає максимально реалістично завдяки синтезу текстур.

5. Дискримінатор D приймає на вхід зображення і повертає скаляр, який представляє ймовірність того, що об'єкт є реальним зображенням x, а не згенерованим штучним зразком $\hat{x}$. Це стимулює генератор G покращувати якість відновлення, щоб обдурити дискримінатор

$$\min_G \max_D V(D,G) = E_{p_{data}(x)}[log\, D(x)] + E_{p_z(z)}\left[log\left(1 - D\big(G(z)\big)\right)\right] + L_{comp} \quad (3.10)$$

L_{comp} - додаткові втрати автокодера (метрика різниці між x та $\hat{x}$, а також обмеження на ентропію квантованого коду $\hat{w}$.

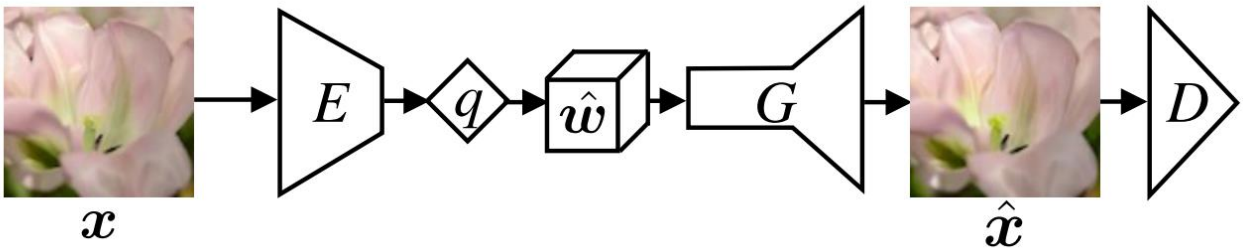


Рисунок 3.11 – Ілюстрація архітектури автокодера із генеративно-змагальною мережею для стиснення зображень з екстремально низькою роздільною здатністю [29]

3.6 Систематизований огляд практичних рішень стиснення зображень на основі штучних нейронних мереж

У таблиці 3.1 систематизовані характеристики сучасних рішень на основі на основі моделей глибинного навчання штучних нейронних мереж, які згруповано за типами мереж, застосованими інструментами просторової кореляції, стратегіями оптимізації бітрейту та підсумковою якістю відновленого зображення.

Таблиця 3.1 – Рішення для кодування зображень на основі навчання, організовані відповідно до вищезазначених моделей штучних нейронних мереж (AE – Автокодер, GAN – Генеративно-змагальна мережа) [30]

Реалізація	Тип нейромережі	Розмір одиниці кодування	Інструменти просторової кореляції	Стратегія контролю бітрейта	Якість обробленого зображення
1	1) MLP AE 2) Рекурентний AE 3) Згортковий AE	32 x 32	Залишкове кодування з двоітераційною архітектурою	Фіксована (одна модель на розмір)	Від низької до високої якості

Продовження таблиці 3.1

2	Рекурентна АЕ зі згортковими шарами	32 x 32	Рекурентні шари з пам'яттю між ітераціями та режимом ентропійного кодування	Змінний бітрейт/якість з використанням однієї моделі	Від низької до високої якості
3	Згортковий АЕ	128 x 128	Пропущене з'єднання	Три моделі для низького, середнього та високого бітрейту	Від низької до високої якості
4	Згортковий АЕ з нелінійними перетвореннями	256 x 256	Нелінійні перетворення та RDO-оптимізація	Одна модель для кожної точки співвідношення бітрейт/якість	Від низької до середньої якості
5	Варіаційний АЕ	256 x 256	Гіперпріор для обчислення моделі ентропійного кодування	Чотири моделі для різних розмірів	Від низької до високої якості
6	Згортковий АЕ з пірамідальним розкладанням та GAN навчанням	128 x 128	Адаптивна регуляція довжини коду	?	Від низької до високої якості
7	Згортковий АЕ з GAN навчанням	Усе зображення	Кодування області інтересу, використовує семантичні дані	Одна модель для кожної точки бітрейту	Низька якість
8	Згортковий + JPEG	8 x 8	Виявлення області інтересу з подальшим кодуванням JPEG	Єдина(спільна) модель	Від низької до високої якості
9	Згортковий АЕ	128 x 128	Розподіл бітрейту з урахуванням контенту та визначення важливості	?	Від низької до високої якості

Продовження таблиці 3.1

10	Згортковий АЕ	256 x 256	Оптимізація розміру кроку квантування	Єдина(спільна) модель	Від низької до високої якості
11	Згортковий АЕ	128 x 128	Спрощення латентного коду	Одна модель для кожної точки бітрейту	Від низької до високої якості
12	Згортковий АЕ	256 x 256	Неоднорідне квантування	Одна модель для кожної точки бітрейту	Від низької до середньої якості

Аналізуючи наведені данні із таблиці 3.1 дані можна зробтити деякі висновки стосовно деяких окремих моделей, або об’єднаних моделей штучних нейронних мереж.

Для підвищення ефективності стиснення у реалізаціях застосовують різні інструменти просторової кореляції та оптимізації, такі як, залишкове кодування, пропущені з'єднання, нелінійні перетворення, RDO-оптимізація, концепція гіперпріора для моделювання ентропійного кодування, а також семантичний аналіз для виділення областей інтересу (region of Interest).

Практичні реалізації демонструють різні стратегії керування щільністю даних, такі, як, використання фіксованих параметрів у межах однієї моделі, застосування змінного бітрейту, масштабування через набір із кількох окремих моделей для різних рівнів компресії, а також динамічне регулювання довжини коду.

Розмір вхідних блоків (одиниць кодування) у розглянутих моделях варіюється від мінімальних значень 8 x 8 пікселів (при гібридних підходах із JPEG) та 32 x 32 пікселів у ранніх моделях, до великих блоків розміром 128 x 128 та 256 x 256 пікселів, або ж кодування зображення в цілому.

Таким чином, представлена таблиця фіксує поточний стан розвитку технологій нейромережевої компресії та слугує структурованою базою прикладних архітектурних рішень, що використовуються для кодування візуальної інформації.

4 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕХАНІЗМУ СТИСНЕННЯ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Характеристика архітектури bmshj2018-hyperprior, яка була обрана для проведення експериментів по стисненню зображень

Головною відмінністю моделі bmshj2018-hyperprior від класичної архітектури CNN є наявність гіпераприорного шару, який виступає у ролі додаткової інформації для забезпечення більш якісного результату обробки зображень. У звичайних методах стиснення структура цієї побічної інформації розробляється вручну. На противагу цьому, ця модель вивчає латентне представлення ентропійної моделі, так само, як базова модель стиснення вивчає представлення зображення [32]. Bmshj2018-hyperprior мінімізує загальну очікувану довжину коду, навчаючись балансувати кількість побічної інформації з очікуваним покращенням ентропійної моделі [32]. На рисунку 4.1 наведена схема, що зображає цю модель.

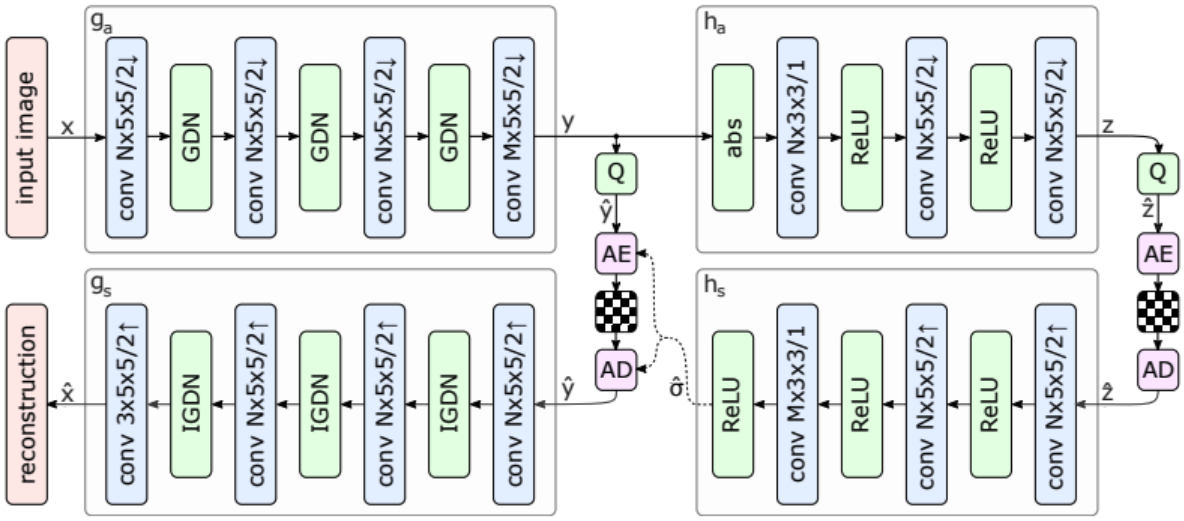


Рисунок 4.1 – Схематичне представлення моделі bmshj2018-hyperprior

Ліва сторона показує архітектуру автокодера зображення, права сторона відповідає автокодеру, що реалізує розрахунок значення гіперапріорного

параметра. Модель з факторизованою апріорною функцією (h_a та h_s) використовує ідентичну архітектуру для аналізу та синтезу, перетворень g_a та g_s. Q представляє квантування [32], а AE, AD представляють арифметичний кодер та арифметичний декодер відповідно. Параметри згортки позначені як: кількість фільтрів × висота підтримки ядра × ширина підтримки ядра / крок дискретизації (при кодуванні вниз (↓), а при декодуванні вгору (↑)) [32]. N = 128 та M = 192 для 5 нижчих значень, а N = 192 та M = 320 для 3 вищих значень [32]. Формула 4.1 зображає роботу гіперпріора. Нелінійна нормалізація виконується з використанням методу узагальненої дивізивної нормалізації (GDN), який спрямований на зменшення обсягу надлишкової інформації в картах ознак. Процес нормалізації враховує взаємозв'язки між каналами ознак у межах одного шару нейронної мережі, що дозволяє знизити їхню статистичну залежність та підвищити ефективність подальшого представлення даних.

4.2 Характеристика тестового набору зображень

Для проведення експериментального дослідження якості стиснення зображень із використанням ШНМ було обрано стандартний набір зображень для цього типу досліджень «Kodak image dataset», який має 24 зображення формату .PNG.

Для проведення цього експерименту було обрано зображення kodim02, kodim23 та kodim24. Зображення kodim02 є переважно монотонним, тому було обране для досліджень того, як нейромережа зможе провести стиснення інформації без втрати таких деталей, як щелини між дошками. Зображення kodim23 має велике різноманіття кольорів із переходами одного виду кольору до іншого. Зображення kodim24 має дуже деталізоване зображення фасаду будинку, тому, це зображення було обране для проведення аналізу як алгоритм нейромережі зможе провести стиснення інформації та її відтворення без втрати цих деталей. На рисунках 4.1 – 4.3 наведені ці 3 зображення.



Рисунок 4.1 – Тестове зображення kodim02, на якому зображені ворота [31]



Рисунок 4.2 – Тестове зображення kodim23, на якому зображені кольорові папуги [31]



Рисунок 4.3 – Тестове зображення kodim24, на якому зображен деталізований фасад будинку [31]

4.3 Обґрунтування вибору програмного забезпечення та моделі ШНМ

Під час вибору прогрманого забезпечення та моделі ШНМ варто дуже детально оцінити технічні характеристики обладнаня, яке може бути задіяне під час проведення експерименту. В моєму персональному комп'ютері встановлені відеокарта RX 7800 XT і процессор Ryzen 5 5600x. Через те, що більшість досліджень у галузі ШНМ проводятся із використанням відеокарт лінійки RTX від Nvidia, через наявність технології CUDA, мені треба було використовувати мій процесор. У Ryzen 5 5600x є 6 ядр із 12 потоками, кешем третього рівня у розмірі 32 мб. В цьому процесорі є апаратні блоки із інструкціями AVX2, які дозволяють використовувати процесор для проведення матричних обчислень шарів кодера ШНМ. Процесор Ryzen 5 5600x має гарні характеристики, але його потужності не вистачить на те, щоб за короткий час навчити з нуля ШНМ проводити стиснення зображень, на це може піти від 1 до 2 тижнів, тому був

обрано варіант із використанням ШНМ, яка вже навчена проводити стиснення через наявність сформованих ваг оцінки інформації. Обрана модель ШНМ класифікується, як bmshj2018-hyperprior, вона є класичною згортковою нейромережею із згортковими шарами. Мова технічної реалізації є – Python, який э стандартом для роботи із ШНМ.

4.4 Технічна реалізація обґрунтованих підходів

Першим кроком реалізації буде створення ізольованого обчислювального середовища (ІОС) на базі інтерпретатора Python 3.10, для цього був встановлений програмний пакет Miniconda. На рисунку 4.4 наведен процес першого кроку.

```
(base) C:\Windows\system32>conda create -n diplom_cpu python=3.10 -y
2 channel Terms of Service accepted
Retrieving notices: done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
    current version: 26.3.2
    latest version: 26.5.2

Please update conda by running

    $ conda update -n base -c defaults conda


## Package Plan ##

  environment location: C:\Users\Gregorius\miniconda3\envs\diplom_cpu

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2026.5.14  |       haa95532_0         107 KB
    libexpat-2.8.1             |       hd7fb8db_0         122 KB
    libzlib-1.3.2              |       h1c6eee0_0          63 KB
    openssl-3.5.7              |       hbb43b14_0         8.9 MB
    packaging-26.0             |  py310haa95532_0         166 KB
    pip-26.1.1                 |     pyhc872135_1         1.1 MB
    python-3.10.20             |       hb00fc5c_1        15.3 MB
    setuptools-82.0.1          |  py310haa95532_0         1.2 MB
    sqlite-3.53.2              |       hee5a0db_0         944 KB
    tzdata-2026b               |       he532380_0         118 KB
    vc-14.3                    |       h2df5915_12          16 KB
    vc14_runtime-14.44.35208   |       h4927774_12         664 KB
    vs2015_runtime-14.44.35208 |       ha6b5a95_12          16 KB
    wheel-0.46.3               |  py310haa95532_0          70 KB
    zlib-1.3.2                 |       h1c6eee0_0         828 KB
    ------------------------------------------------------------
                                           Total:        29.7 MB

The following NEW packages will be INSTALLED:

  bzip2              pkgs/main/win-64::bzip2-1.0.8-h2bbff1b_6
  ca-certificates    pkgs/main/win-64::ca-certificates-2026.5.14-haa95532_0
  libexpat           pkgs/main/win-64::libexpat-2.8.1-hd7fb8db_0
  libffi             pkgs/main/win-64::libffi-3.4.8-h2b21627_2
  libzlib            pkgs/main/win-64::libzlib-1.3.2-h1c6eee0_0
  openssl            pkgs/main/win-64::openssl-3.5.7-hbb43b14_0
  packaging          pkgs/main/win-64::packaging-26.0-py310haa95532_0
  pip                pkgs/main/noarch::pip-26.1.1-pyhc872135_1
  python             pkgs/main/win-64::python-3.10.20-hb00fc5c_1
  setuptools         pkgs/main/win-64::setuptools-82.0.1-py310haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.53.2-hee5a0db_0
  tk                 pkgs/main/win-64::tk-8.6.15-hf199647_0
  tzdata             pkgs/main/noarch::tzdata-2026b-he532380_0
  ucrt               pkgs/main/win-64::ucrt-10.0.22621.0-haa95532_0
  vc                 pkgs/main/win-64::vc-14.3-h2df5915_12
  vc14_runtime       pkgs/main/win-64::vc14_runtime-14.44.35208-h4927774_12
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.44.35208-ha6b5a95_12
  wheel              pkgs/main/win-64::wheel-0.46.3-py310haa95532_0
  xz                 pkgs/main/win-64::xz-5.8.2-h53af0af_0
  zlib               pkgs/main/win-64::zlib-1.3.2-h1c6eee0_0


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate diplom_cpu
#
# To deactivate an active environment, use
#
#     $ conda deactivate


Channel "defaults" has the following notices:
  [info] -- Tue Jun  9 00:00:00 2026
  PyTorch 2.12 with CUDA support is now available to install with your current channel (Anaconda Main). Learn more: https://anaconda.org/main/pytorch?utm_source=channel_notices
```

Рисунок 4.4 – Процес створення ІОС на базі інтерпретатора Python 3.10

Другим короком було зроблено налаштування IOC, для цього до нього було встановлено PyTorch (фреймворк глибокого навчання), який сконфігурований для виконання тензорних операцій на базі CPU. Після встановлення цього фреймворку було встановлено CompressAI (бібліотеку наскрізноого стиснення зобржень), яка містить обрану архітектуру bmshj2018-hyperprior. На рисунках 4.5 – 4.8 наведен процес встановлення фреймворку та бібліотеки.

```
(diplom_cpu) C:\Windows\system32>conda install pytorch torchvision cpuonly -c pytorch -y
3 channel Terms of Service accepted
Channels:
 - pytorch
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
    current version: 26.3.2
    latest version: 26.5.2

Please update conda by running

    $ conda update -n base -c defaults conda



## Package Plan ##

  environment location: C:\Users\Gregorius\miniconda3\envs\diplom_cpu

  added / updated specs:
    - cpuonly
    - pytorch
    - torchvision


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    blas-1.0                   |              mkl           6 KB
    brotlicffi-1.2.0.0         |  py310h885b0b7_0         346 KB
    certifi-2026.5.20          |  py310haa95532_0         131 KB
    cffi-2.0.0                 |  py310h02ab6af_1         241 KB
    charset-normalizer-3.4.4   |  py310haa95532_0         106 KB
    cpuonly-2.0                |                0           2 KB  pytorch
    filelock-3.20.3            |  py310haa95532_0          34 KB
    freetype-2.14.1            |       hfbffc0b_0         514 KB
    giflib-5.2.2               |       h7edc060_0         105 KB
    gmp-6.3.0                  |       h537511b_0         330 KB
    gmpy2-2.2.2                |  py310h8598115_0         211 KB
    idna-3.18                  |  py310haa95532_0         237 KB
    intel-openmp-2023.1.0      |   h59b6b97_46320         2.7 MB
    jinja2-3.1.6               |  py310haa95532_0         279 KB
    jpeg-9f                    |       ha349fce_0         302 KB
    lcms2-2.16                 |       hb4a4139_0         566 KB
    lerc-3.0                   |       hd77b12b_0         120 KB
    libdeflate-1.17            |       h2bbff1b_1         153 KB
    libjpeg-turbo-2.0.0        |       h196d8e1_0         618 KB
    libpng-1.6.56              |       h2854ad3_0         239 KB
    libtiff-4.5.1              |       hd77b12b_0         1.1 MB
    libuv-1.52.0               |       hd0d7782_0         298 KB
    libwebp-1.3.2              |       hbc33d0d_0          73 KB
    libwebp-base-1.3.2         |       h3d04722_1         303 KB
    markupsafe-3.0.2           |  py310h827c3e9_0          36 KB
    mkl-2023.1.0               |   h6b88ed4_46358       155.9 MB
    mkl-service-2.4.0          |  py310h827c3e9_2          66 KB
    mkl_fft-1.3.11             |  py310h827c3e9_0         168 KB
    mkl_random-1.2.8           |  py310hc64d2fc_0         257 KB
    mpc-1.3.1                  |       h827c3e9_0          85 KB
    mpfr-4.2.1                 |       h56c3642_0         266 KB
    mpmath-1.3.0               |  py310haa95532_0         834 KB
    networkx-3.4.2             |  py310haa95532_0         2.5 MB
    numpy-2.0.1                |  py310h055cbcc_1          11 KB
    numpy-base-2.0.1           |  py310h65a83cf_1         9.1 MB
    openjpeg-2.5.2             |       hae555c5_0         268 KB
    pillow-11.1.0              |  py310h096bfcc_0         770 KB
    pycparser-3.0              |  py310haa95532_0         113 KB
    pysocks-1.7.1              |  py310haa95532_1          29 KB
    pytorch-2.5.1              |     py3.10_cpu_0       147.4 MB  pytorch
    pytorch-mutex-1.0          |             cpu             3 KB  pytorch
    pyyaml-6.0.3               |  py310hb9a58be_0         203 KB
    requests-2.34.2            |  py310haa95532_0         157 KB
    sympy-1.14.0               |  py310haa95532_1        10.6 MB
    tbb-2021.8.0               |       h59b6b97_0         149 KB
    torchvision-0.20.1         |        py310_cpu         6.5 MB  pytorch
    typing_extensions-4.15.0   |  py310haa95532_0          79 KB
    urllib3-2.7.0              |  py310haa95532_0         298 KB
    win_inet_pton-1.1.0        |  py310haa95532_1           9 KB
    yaml-0.2.5                 |       he774522_0          62 KB
    ------------------------------------------------------------
                                           Total:       344.7 MB
```

Рисунок 4.5 – Процес встановлення фреймворку «PyTorch»

```
The following NEW packages will be INSTALLED:

  blas                pkgs/main/win-64::blas-1.0-mkl
  brotlicffi          pkgs/main/win-64::brotlicffi-1.2.0.0-py310h885b0b7_0
  certifi             pkgs/main/win-64::certifi-2026.5.20-py310haa95532_0
  cffi                pkgs/main/win-64::cffi-2.0.0-py310h02ab6af_1
  charset-normalizer  pkgs/main/win-64::charset-normalizer-3.4.4-py310haa95532_0
  cpuonly             pytorch/noarch::cpuonly-2.0-0
  filelock            pkgs/main/win-64::filelock-3.20.3-py310haa95532_0
  freetype            pkgs/main/win-64::freetype-2.14.1-hfbffc0b_0
  giflib              pkgs/main/win-64::giflib-5.2.2-h7edc060_0
  gmp                 pkgs/main/win-64::gmp-6.3.0-h537511b_0
  gmpy2               pkgs/main/win-64::gmpy2-2.2.2-py310h8598115_0
  idna                pkgs/main/win-64::idna-3.18-py310haa95532_0
  intel-openmp        pkgs/main/win-64::intel-openmp-2023.1.0-h59b6b97_46320
  jinja2              pkgs/main/win-64::jinja2-3.1.6-py310haa95532_0
  jpeg                pkgs/main/win-64::jpeg-9f-ha349fce_0
  lcms2               pkgs/main/win-64::lcms2-2.16-hb4a4139_0
  lerc                pkgs/main/win-64::lerc-3.0-hd77b12b_0
  libdeflate          pkgs/main/win-64::libdeflate-1.17-h2bbff1b_1
  libjpeg-turbo       pkgs/main/win-64::libjpeg-turbo-2.0.0-h196d8e1_0
  libpng              pkgs/main/win-64::libpng-1.6.56-h2854ad3_0
  libtiff             pkgs/main/win-64::libtiff-4.5.1-hd77b12b_0
  libuv               pkgs/main/win-64::libuv-1.52.0-hd0d7782_0
  libwebp             pkgs/main/win-64::libwebp-1.3.2-hbc33d0d_0
  libwebp-base        pkgs/main/win-64::libwebp-base-1.3.2-h3d04722_1
  lz4-c               pkgs/main/win-64::lz4-c-1.9.4-h2bbff1b_1
  markupsafe          pkgs/main/win-64::markupsafe-3.0.2-py310h827c3e9_0
  mkl                 pkgs/main/win-64::mkl-2023.1.0-h6b88ed4_46358
  mkl-service         pkgs/main/win-64::mkl-service-2.4.0-py310h827c3e9_2
  mkl_fft             pkgs/main/win-64::mkl_fft-1.3.11-py310h827c3e9_0
  mkl_random          pkgs/main/win-64::mkl_random-1.2.8-py310hc64d2fc_0
  mpc                 pkgs/main/win-64::mpc-1.3.1-h827c3e9_0
  mpfr                pkgs/main/win-64::mpfr-4.2.1-h56c3642_0
  mpmath              pkgs/main/win-64::mpmath-1.3.0-py310haa95532_0
  networkx            pkgs/main/win-64::networkx-3.4.2-py310haa95532_0
  numpy               pkgs/main/win-64::numpy-2.0.1-py310h055cbcc_1
  numpy-base          pkgs/main/win-64::numpy-base-2.0.1-py310h65a83cf_1
  openjpeg            pkgs/main/win-64::openjpeg-2.5.2-hae555c5_0
  pillow              pkgs/main/win-64::pillow-11.1.0-py310h096bfcc_0
  pycparser           pkgs/main/win-64::pycparser-3.0-py310haa95532_0
  pysocks             pkgs/main/win-64::pysocks-1.7.1-py310haa95532_1
  pytorch             pytorch/win-64::pytorch-2.5.1-py3.10_cpu_0
  pytorch-mutex       pytorch/noarch::pytorch-mutex-1.0-cpu
  pyyaml              pkgs/main/win-64::pyyaml-6.0.3-py310hb9a58be_0
  requests            pkgs/main/win-64::requests-2.34.2-py310haa95532_0
  sympy               pkgs/main/win-64::sympy-1.14.0-py310haa95532_1
  tbb                 pkgs/main/win-64::tbb-2021.8.0-h59b6b97_0
  torchvision         pytorch/win-64::torchvision-0.20.1-py310_cpu
  typing_extensions   pkgs/main/win-64::typing_extensions-4.15.0-py310haa95532_0
  urllib3             pkgs/main/win-64::urllib3-2.7.0-py310haa95532_0
  win_inet_pton       pkgs/main/win-64::win_inet_pton-1.1.0-py310haa95532_1
  yaml                pkgs/main/win-64::yaml-0.2.5-he774522_0
  zstd                pkgs/main/win-64::zstd-1.5.7-h56299aa_0


Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Рисунок 4.6 – Процес встановлення фреймворку «PyTorch» (продовження)



Рисунок 4.7 – Процес встановлення бібліотеки CompressAI

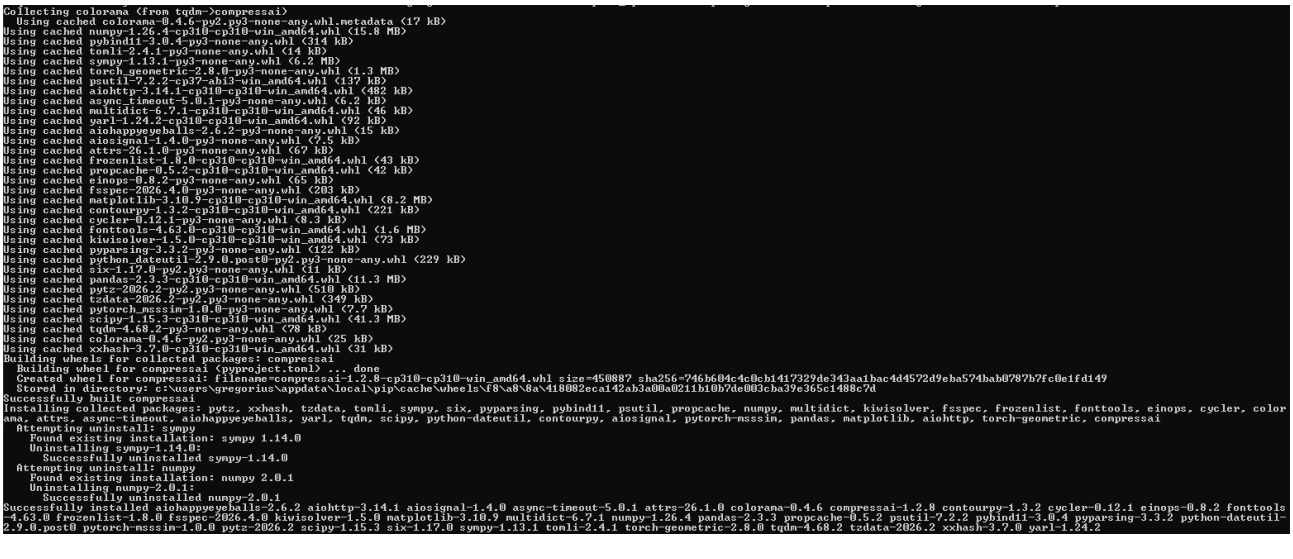


Рисунок 4.8 – Процес встановлення бібліотеки CompressAI (продовження)

Третім кроком було написання скрипту на мові Python, в якому буде описані, процесс стиснення зображення, відповідні розрахунки метрик PSNR та BPP, в додатку А, наведен текст коду цього скрипта. Параметри стиснення

вимірються (q) від 1 до 8, у ході експерименту було проведено 3 тести з q = 1, 4, 8 для кожного із 3-х зображень. Для первинного наочного аналізу на рисунках 4.9 – 4.17 наведені відтворені зображення.



Рисунок 4.9 – Зображення kodim02 із найбільшим рівнем стиснення q = 1



Рисунок 4.10 – Зображення kodim02 із середнім рівнем стиснення q = 4



Рисунок 4.11 – Зображення kodim02 із найменшим рівнем стиснення q = 8



Рисунок 4.12 – Зображення kodim23 із найбільшим рівнем стиснення q = 1



Рисунок 4.13 – Зображення kodim23 із середнім рівнем стиснення q = 4



Рисунок 4.14 – Зображення kodim23 із найменшим рівнем стиснення q = 8



Рисунок 4.15 – Зображення kodim24 із найбільшим рівнем стиснення q = 1



Рисунок 4.16 – Зображення kodim24 із середнім рівнем стиснення q = 4



Рисунок 4.17 – Зображення kodim24 із найменшим рівнем стиснення q = 8

Провівши первинний аналіз зображень можна сказати, що якщо в зображенні немає читкіх переходів одного кольору до іншого, то при стисненні деталі із різних планів можуть зливатись у однорідну текстуру, на рисунку 4.15 видно, як віконниці злились із деревом, яке видно з-за цього будинку.

4.5 Аналіз результатів роботи ШНМ в порівнянні із стандартом JPEG

Було сформовано таблиці параметрів, на які варто звертати увагу при проведенні аналізу ефективності і доцільності сиснення зображень із допомогою ШНМ у порівнянні з стандартом JPEG. У таблицях 4.1 – 4.2 наведені параметри файлів, які піддалися стисненню описаними вище методами.

Таблиця 4.1 – Параметри описаних вище зображеннь, після стиснення із використанням стандарта JPEG

Зображення	Рівень якості	Значення параметру BPP	Розмір файлу після стиснення	Значення параметру PSNR
Kodim02	q = 15%	0,34 біт/пікс.	16,3 КБ	29,85 дБ
	q = 45%	0,54 біт/пікс.	25,9 КБ	32,90 дБ
	q = 95%	1,72 біт/пікс.	82,5 КБ	41,20 дБ
Kodim23	q = 15%	0,29 біт/пікс.	13,9 КБ	31,40 дБ
	q = 45%	0,47 біт/пікс.	22,5 КБ	34,75 дБ
	q = 95%	1,51 біт/пікс.	72,4 КБ	42,80 дБ
Kodim24	q = 15%	0,51 біт/пікс.	24,4 КБ	28,10 дБ
	q = 45%	0,82 біт/пікс.	39,3 КБ	31,35 дБ
	q = 95%	2,34 біт/пікс.	112,3 КБ	39,95 дБ

Таблиця 4.2 – Параметри описаних вище зображеннь, після стиснення із використанням ШНМ моделі bmshj2018-hyperprior

Зображення	Рівень якості	Значення параметру BPP	Розмір файлу після стиснення	Значення параметру PSNR
Kodim02	q = 1	0,089 біт/пікс.	4,368 КБ	29,05 дБ
	q = 4	0,33 біт/пікс.	16,292 КБ	33,17 дБ
	q = 8	1,54 біт/пікс.	75,664 КБ	40,70 дБ
Kodim23	q = 1	0,0802 біт/пікс.	3,94 КБ	30,80 дБ
	q = 4	0,217 біт/пікс.	10,676 КБ	35,86 дБ
	q = 8	0,83 біт/пікс.	40,812 КБ	41,91 дБ
Kodim24	q = 1	0,1654 біт/пікс.	8,123 КБ	24,98 дБ
	q = 4	0,6362 біт/пікс.	31,272 КБ	30,89 дБ
	q = 8	2,04 біт/пікс.	100,276 КБ	38,58 дБ

Для зручності порівняння було побудовано графіки залежності метрики PSNR від бітової швидкості BPP. На рисунках 4.19 – 4.21 наведені графіки залежності описаних вище параметрів.

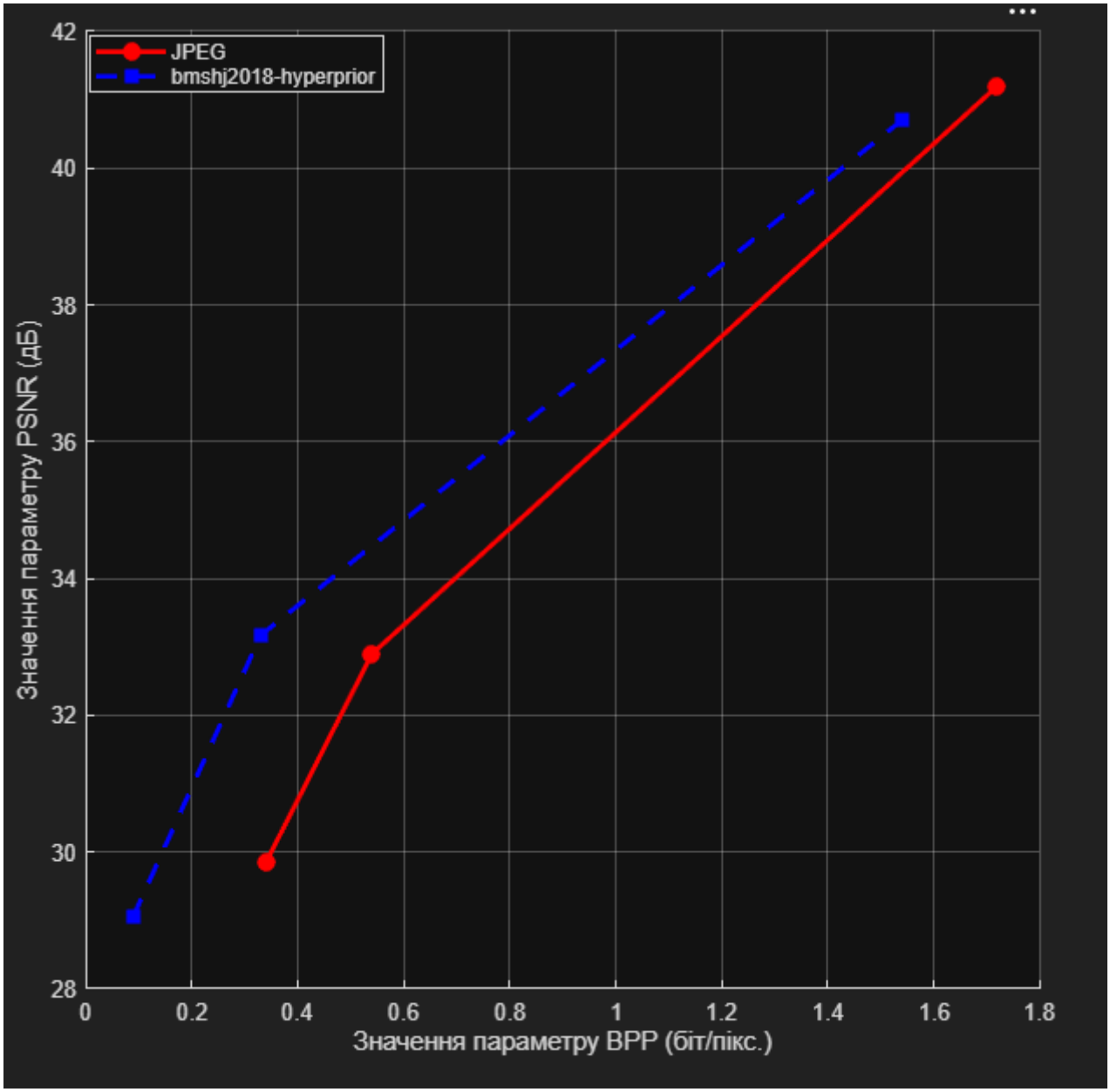


Рисунок 4.19 – Графік залежності метрики PSNR від бітової швидкості BPP для зображення kodim02

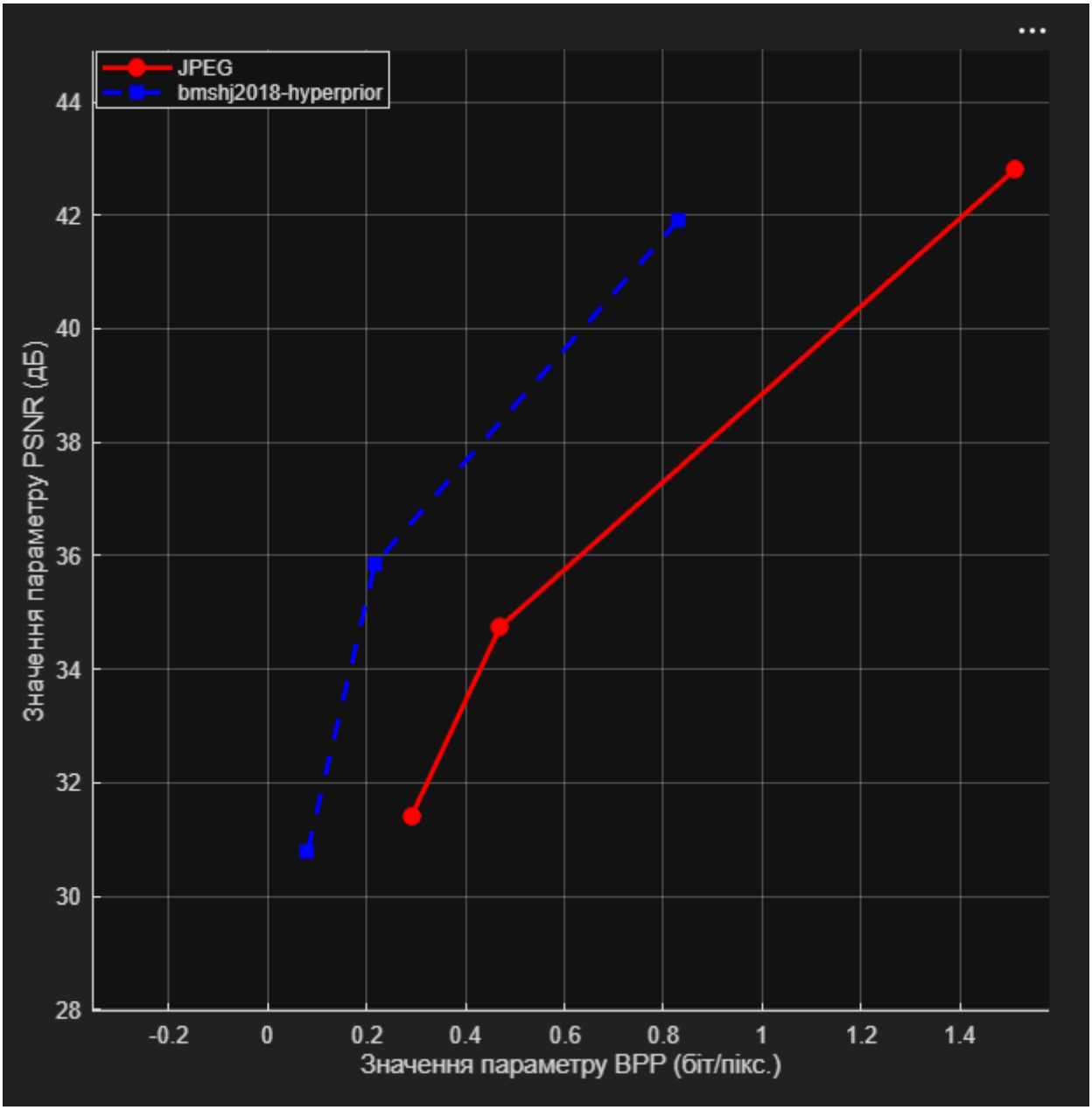


Рисунок 4.20 – Графік залежності метрики PSNR від бітової швидкості BPP для зображення kodim23

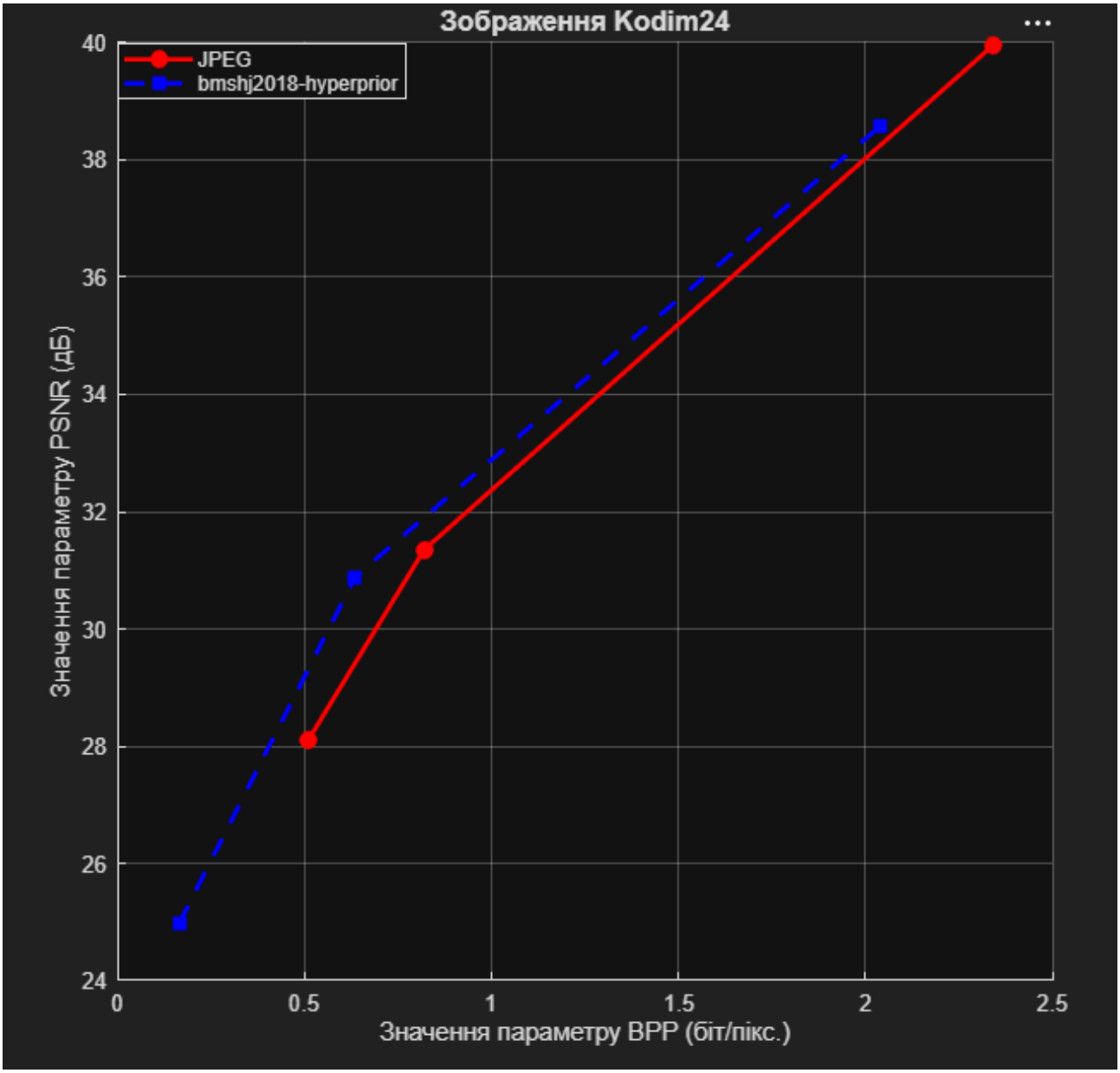


Рисунок 4.21 – Графік залежності метрики PSNR від бітової швидкості BPP для зображення kodim24

Провівши аналіз отриманих значень можна сказати, що у порівнянні з стандартом JPEG, метод стиснення зображень із залученням штучної нейронної мережі э більш ефективим.

На рисунках 4.19 – 4.20 можна побачити, що у випадку із BPP та розміром, ШНМ показує значення менші у 2 – 2,5 рази у порівнянні із стандартом JPEG. Цю закономірність можна пояснити різницею до підходу обробки зображення. JPEG розбиває зображення на сотні секторів 8x8 пікселів, що призводить до значної кількості надлишкової інформації. Тоді як ШНМ має глобальний підхід

до аналізу зображення пропускаючи його через згорткові шари. Ці шари аналізуючи зображення формують його компактне латентне представлення. Через збільшення вибірки пікселей – зменшується кількість секторів, що знижує вплив надлишкової інформації на розмір зображення, яке піддавалось стисненню.

Аналізуючи рисунок 4.21 можна побачити, що якість зображення, яка рахується в PSNR має розбіжність в 0,5 – 1,5 дБ на користь стандарту JPEG. Таку закономірність можна пояснити тим, що ШНМ проводить роботу із латентним простором аналізуючи контекст зображення. У випадку коли ШНМ не вистачає бітів для кодування, то вона плавно розмиває дрібні деталі, що чітко простежується із високим рівнем стиснення. Ці розмиття пікселів формула PSNR сприймає як критичну помилку, що призводить до штучного заниження фінального показника якості зображення.

ВИСНОВКИ

В даній кваліфікаційній роботі було проведено аналіз сучасних методів стиснення цифрових зображень із застосування штучних нейронних мереж. Було проведено дослідження основних видів архітектур нейронних мереж, що використовують технологію глибинного навчання. Додатково було описано принцип роботи цих архітектур.

Дослідження сучасних архітектур нейромереж дозволив обґрунтувати вибір згорткових нейронних мереж (CNN) через те, що вона э основних стандартом для проведення стиснення цифрових зображень. У межах роботи було розглянуто та оптимізовано CPU-середовище для розгортання програмного комплексу моделі bmshj2018-hyperprio на базі процесора AMD Ryzen 5600x. Експериментальні тести проводились із використанням базового набору зображень kodak.

Окрему увагу було приділено порівняльному аналізу інженерних метрик інформаційної щільності (BPP) та об’єктивної якості (PSNR) у порвінянні із класичним стандартом JPEG. Було встановлено, що на усіх рівнях стиснення ШНМ-кодек має схожі результати із стандартом JPEG. Варто зауважити, що при відповідних параметрах рівня якості стиснення ШНМ-кодеки мають меньші значення об'єктивної якості (PSNR) в диапазоні різниці від 0,5 до 1,5 дБ. На практиці ця похибка компенсується високою якістю збереження геометрії об’єктів під час аналізу зображення ШНМ-кодеком.

Спираючись на отримані дані, можна зробити висновок про високу ефективність ШНМ-кодеків, що дає змогу замислитись про інтеграцію їх до сучасних систем передачі даних. Проте впровадження цієї технології на поточному етапі має низку суттєвих інженерних обмежень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Graupe, D. (2013). *Principles of artificial neural networks* (3rd ed.). World Scientific Publishing.
2. Hu, Y. H., & Hwang, J.-N. (Eds.). (2002). *Handbook of neural network signal processing*. CRC Press.
3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
4. Dertat, A. (2017). *Recurrent neural networks (RNNs): A gentle introduction and overview*.
5. Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). *An introduction to convolutional neural networks*. Radiology, 288(3), 634–649.
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative adversarial nets*. In *Advances in Neural Information Processing Systems 27* (pp. 2672–2680).
7. Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., & Van Gool, L. (2018). *End-to-end optimization of nonlinear transform codes for perceptual quality*. In *Picture Coding Symposium (PCS)* (pp. 1–5).
8. Mikhael, W., Tschannen, M., Agustsson, E., & Van Gool, L. (2019). *CAE-ADMM: Implicit bitrate optimization via ADMM-based pruning in compressive autoencoders*.
9. He, D., Zheng, H., Sun, B., Wang, Y., & Qin, H. (2018). *Image-dependent local entropy models for learned image compression*.
10. Theis, L., Shi, W., Cunningham, A., & Huszár, F. (2017). *Lossy image compression with compressive autoencoders*.
11. Ascenso, J., Akayzi, P., Testolina, M., Boev, A., & Alshina, E. (2019). *Learning-based image coding: Early solutions reviewing and subjective quality evaluation*.
12. Toderici, G., Vincent, D., Johnston, N., Hwang, S. J., Minnen, D., Shor, J., & Covell, M. (2017). *Image and video compression with neural networks*.

13. Wallace, G. K. (1991). The JPEG still picture compression standard. *Communications of the ACM, 34*(4), 30–44.
14. Gardner, M. W., & Dorling, S. R. (1998). Artificial neural networks (the multilayer perceptron): A review of applications in the atmospheric sciences. *Atmospheric Environment, 32*(14–15), 2627–2636.
15. Schalkoff, R. J. (1997). *Artificial neural networks*. McGraw-Hill.
16. Dony, R. D., & Haykin, S. (1995). Neural network approaches to image compression. *Proceedings of the IEEE, 83*(2), 288–303.
17. Dianat, S., Nasrabadi, N., & Venkataraman, S. (1991). A non-linear predictor for differential pulse-code encoder (DPCM) using artificial neural networks. In *Proceedings of ICASSP 1991* (pp. 2793–2796).
18. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536.
19. Gelenbe, E. (1989). Random neural networks with negative and positive signals and product form solution. *Neural Computation, 1*(4), 502–510.
20. Gelenbe, E., & Sungur, M. (1994). Random network learning and image compression. In *Proceedings of the IEEE International Conference on Neural Networks* (Vol. 6, pp. 3996–3999).
21. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444.
22. Ballé, J., Laparra, V., & Simoncelli, E. P. (2016). End-to-end optimized image compression.
23. Ballé, J., Laparra, V., & Simoncelli, E. P. (2017). End-to-end optimized image compression. In *International Conference on Learning Representations (ICLR)*.
24. Li, M., Zuo, W., Gu, S., Zhao, D., & Zhang, D. (2018). Learning convolutional networks for content-weighted image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
25. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.
26. Toderici, G., O'Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., & Sukthankar, R. (2016). Variable rate image compression with

recurrent neural networks. In *International Conference on Learning Representations (ICLR)*.

27. Toderici, G., Vincent, D., Johnston, N., Hwang, S. J., Minnen, D., Shor, J., & Covell, M. (2017). Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
28. Rippel, O., & Bourdev, L. (2017). Real-time adaptive image compression. In *Proceedings of the International Conference on Machine Learning (ICML)*.
29. Agustsson, E., Tschannen, M., Mentzer, F., Timofte, R., & Van Gool, L. (2019). Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
30. Ascenso, J., Akyazi, P., Pereira, F., & Ebrahimi, T. (2020). *Learning-based image coding: Early solutions reviewing and subjective quality evaluation*. In P. Schelkens & T. Kozacki (Eds.), *Optics, Photonics and Digital Technologies for Imaging Applications VI* (Proceedings of SPIE, Vol. 11353, Article 113530S). SPIE.
31. Franzen, R. (1999). *Kodak Lossless True Color Image Suite*. URL: https://r0k.us/graphics/kodak/ (дата звернення: 15.06.2026)
32. Ballé, J., Minnen, D., Singh, S., Hwang, S. J., & Johnston, N. (2018). *Variational image compression with a scale hyperprior*.