

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
РОЗПІЗНАВАННЯ ВЕЛИКОФОРМАТНИХ ЗОБРАЖЕНЬ НА ОСНОВІ
ДИНАМІЧНОЇ ОРГАНІЗАЦІЇ ДАНИХ
RESEARCH AND SOFTWARE IMPLEMENTATION OF THE METHODS FOR
LARGE-SCALE IMAGE RECOGNITION BASED ON DYNAMIC DATA
MANAGEMENT

Виконав: студент 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Нестеренко М.А.

(прізвище та ініціали)

Керівник Каплієнко Т.І.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Нестеренко Маргариті Анатоліївні
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів розпізнавання великоформатних зображень на основі динамічної організації даних. Research and Software Implementation of the Methods for Large-Scale Image Recognition Based on Dynamic Data Management

керівник проєкту (роботи) Каплієнко Тетяна Ігорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “05” листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) 7 грудня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Розробка програми. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	Каплієнко Т.І., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 4 ” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	1.1.1 Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7–8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Розробка організаційно-економічної частини	10 тиждень	Розділ 6
8	Розробка заходів з охорони праці та цивільної безпеки.	11 тиждень	Розділ 7
9	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
10	Нормоконтроль та рецензування.	14–15 тижні	
11	Захист роботи.	16 тиждень	

Студент(ка)

(підпис)

Нестеренко М.А.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Каплієнко Т.І.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 113 с., 5 табл., 23 рис., 4 дод., 22 джерел.

РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ВЕЛИКОФОРМАТНІ ЗОБРАЖЕННЯ, ПАРАЛЕЛЬНА ОБРОБКА, ВИСОКОПРОДУКТИВНІ ОБЧИСЛЕННЯ, ДИНАМІЧНЕ ЗБЕРЕЖЕННЯ ТА ОБРОБКА

Об'єкт дослідження – процес розпізнавання великоформатних зображень на основі динамічної організації даних.

Предмет дослідження – методи розпізнавання великоформатних зображень на основі динамічної організації даних.

Мета роботи – дослідити, проаналізувати та реалізувати методи розпізнавання великоформатних зображень на основі паралельних обчислень для динамічної організації даних.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування Python, фреймворк машинного навчання TensorFlow, середовище розробки Anaconda, персональний комп'ютер з процесором Intel 12-core Xeon 4214R з 12 ядрами та 24 потоками для виконання високопродуктивних обчислень (тактова частота процесору: 3,5 ГГц) під управлінням операційної системи Microsoft Windows 10.

Результати. Створено програмний застосунок, що реалізує метод розпізнавання зображень з використанням паралельної обробки великоформатних зображень.

Висновки. Розроблено програмний застосунок для розпізнавання зображень.

Галузь використання – системи комп'ютерного зору.

ABSTRACT

Explanatory note to the qualifying work of the master: 113 pages, 23 figures, 5 tables, 4 appendixes, 22 sources.

IMAGE RECOGNITION, LARGE FORMAT IMAGES, PARALLEL PROCESSING, HIGH-PERFORMANCE COMPUTING, DYNAMIC SAVING AND PROCESSING

The object of research is the process of recognizing large – format images based on dynamic data organization.

The subject of the research is Methods for recognizing large – format images based on dynamic data organization.

The aim of the work is to investigate, analyze and implement methods for recognizing large – format images based on parallel calculations for dynamic data organization.

Materials, methods and technical means: structural and Object-Oriented programming, Python programming language, TensorFlow machine learning framework, Anaconda development environment, personal computer with Intel 12-core Xeon 4214r processor with 12 cores and 24 threads for performing high-performance calculations (processor clock speed: 3.5 GHz) running the Microsoft Windows 10 operating system.

Results. A software application has been created that implements an image recognition method using parallel processing of large-format images.

Conclusions. A software application for image recognition has been developed.

The field of use is computer vision systems.

ЗМІСТ

	С.
Вступ.....	9
1 Аналіз проблемної області та постановка завдань дослідження	12
1.1 Візуальний пошук та розпізнавання об'єктів за зображеннями	12
1.2 Методи РЗ	15
1.3 Використання ВЗ при РЗ	21
1.4 Підходи до розпаралелювання обробки зображень	32
1.5 Постановка завдання	36
1.6 Висновки за розділом 1	36
2 Матеріали та методи.....	38
2.1 Структура розподіленого зображення	38
2.2 Метод динамічного розподілу обчислювального навантаження ..	40
2.3 Подальше РЗ з використанням нейромоделей.....	42
2.4 Висновки за розділом 2	51
3 Розробка архітектури програми	54
3.1 Опис основних вимог до програми	54
3.2 Вибір мови програмування	54
3.3 Фреймворки та бібліотеки машинного навчання	56
3.4 Вибір середовища розробки.....	58
3.5 Проектування розробки.....	61
3.6 Вимоги до технічних засобів	61
3.7 Висновки за розділом 3	62
4 Основні рішення щодо реалізації компонентів системи	63
4.1 Розробка архітектури сервіс-орієнтованої розподіленої системи обробки ВЗ	63
4.2 Розпізнавання з CNN	66
4.3 Проектування моделі.....	68
5 Експлуатація, тестування та експериментальне дослідження програми..	74

5.1 Опис стратегії тестування ПЗ	74
5.2 Тестування та аналіз результатів.....	75
5.3 Висновки за результатами тестування.....	78
Висновки	80
Перелік джерел посилення	82
Додаток А Технічне завдання	85
Додаток Б Опис програми	90
Додаток В Текст програми	93
Додаток Д Слайди презентації.....	106

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ВЗ –	великоформатні зображення;
НМ –	нейронні мережі;
ОС –	операційна система;
ОЧС –	обчислювальних систем;
ПЗ –	програмний застосунок;
ПК –	персональний комп'ютер;
РЗ –	розпізнавання зображень;
CNN –	згортова нейронна мережа;
LSTM-RNN –	long short-term memory recurrent neural network;
RGB –	Red-Green-Blue;
RNN –	рекурентні нейронні мережі.

ВСТУП

Однією з найважливіших проблем використання обчислювальної техніки є «відображення задач обчислювальної математики на архітектуру обчислювальних систем». Ця проблема була позначена багатьма науковцями, як фундаментальний науковий напрямок, коротко званий «проблемою відображення» [1].

В даний час найбільш актуальним є вирішення проблеми відображення обчислювальних задач на паралельну архітектуру обчислювальних систем (ОЧС), оскільки основним напрямком підвищення ефективності використання обчислювальних засобів є використання паралельних методів організації обчислень [1].

Основним підходом до вирішення проблеми відображення є аналіз обчислювальної задачі, що виявляє паралелізм і можливості використання розподілених даних, [2] що виконується на основі математично еквівалентних перетворень моделі інформаційної структури методу розв'язання досліджуваної задачі, або, більш узагальнено, моделі інформаційної технології розв'язання задачі [3].

У роботах, пов'язаних з паралельною обробкою зображень, в тому числі розпізнавання зображень (РЗ), в основному розглядаються питання побудови паралельних алгоритмів обробки зображень, адаптації послідовних алгоритмів стосовно до паралельних архітектур ОЧС [1].

Однак питання створення моделей інформаційних технологій обробки зображень, завдання побудови математично еквівалентних перетворень цих моделей і вирішення на цій основі проблеми відображення задач обробки зображень на архітектурі паралельних ОЧС до теперішнього часу практично не досліджувалися [1].

Природний при обробці великоформатних зображень (ВЗ) паралелізм, заснований на декомпозиції даних, дозволяє досить легко адаптувати

послідовні реалізації широкого спектру завдань обробки для їх виконання на багатоядерних процесорах [3]. Однак зі збільшенням розмірів зображень час їх обробки все більше визначається часом, що витрачається на операції введення/виводу. В таких умовах при обробці ВЗ з використанням відносно простих в обчислювальному відношенні завдань ефективність використання багатоядерних процесорів різко падає. Існуючі тенденції розвитку обчислювальної техніки ще більше загострюють проблему: все збільшується розрив між продуктивністю процесорів і швидкодією пристроїв постійного зберігання даних істотно знижує показники загальної ефективності ОЧС при вирішенні прикладних завдань [2].

Обґрунтоване вирішення проблеми відображення задач обробки зображень на архітектуру розподілених обчислювальних систем з різними типами паралелізму має спиратися на моделювання і дослідження різних варіантів організації зберігання і обробки зображень стосовно цільової архітектури апаратно-програмних засобів [3]. Наприклад, централізоване розміщення ВЗ на спеціалізованих сховищах даних є вузьким місцем паралельних систем обробки зображень, з іншого боку, при організації розподіленого зберігання необхідно вирішити проблеми, пов'язані з вибором найкращого декомпозиції (розбиття на фрагменти) зображення в умовах відсутності апріорної інформації про параметри запускаються в системі завдань обробки, збалансованістю завантаження комп'ютерів, що беруть участь в обробці при заздалегідь виконаної декомпозиції даних, відмовостійкістю розподіленого зберігання фрагментів зображень, забезпеченням інтерактивності системи при візуалізації розподілених зображень [2].

У дипломній кваліфікаційній роботі магістра на основі розроблених моделей технологій обробки зображень і методів їх перетворення, що виявляють паралелізм, запропоновані методи динамічної організації зберігання і обробки даних, які знімають ці проблеми, запропонована

архітектура і принципи реалізації розподіленої системи зберігання і обробки зображень.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Перший розділ буде присвячено детальному аналізу проблемної області пов'язаної з обробкою ВЗ, в тому числі при вирішенні задач розпізнавання. Особливу увагу буде приділено аналізу актуальності задач розпізнавання та використання ВЗ в задачах розпізнавання. Під час роботи буде проведено дослідження основних термінів таких, як розпізнавання зображень, ВЗ та паралельна обробка.

1.1 Візуальний пошук та розпізнавання об'єктів за зображеннями

Візуальний пошук – сильний тренд на ринку споживчих товарів, суть якого в тому, що користувачеві досить сфотографувати предмет або навести на нього камеру, і програма за частки секунд знайде цей товар в інтернет-магазині, повідомить його характеристики і ціну [1].

Тобто, раніше можна було надіслати фото пристрою або його серійний номер обслуговуючому персоналу магазину та вже за цією інформацією клієнту підбирали варіанти [2]. У такого підходу є очевидна складність – людський фактор або досвід персоналу. Наприклад, тільки менеджер з великим досвідом роботи може розпізнати погризений собакою пульт з пошкодженими кнопками або запчастину для авто [1]. Тільки фахівець зі стажем може по фото визначити комаху, яка вкусила хатню тварину. Нещодавно найнятий співробітник з таким завданням не впорається – його необхідно тривалий час навчати [2].

Саме тому все частіше використовують передові технології штучного інтелекту для вирішення таких задач [3]. Проте імплементація таких технологій в реальне життя стикається з низкою проблем, які треба досліджувати та вирішувати [4].

Сучасний світ переповнений різними технологіями та інноваціями, розпізнавання образів і зображень вже не здається нам чимось новим і інтригуючим. За такою технологією стоїть велика історія і величезна праця багатьох вчених, інженерів, конструкторів і винахідників [4].

РЗ та визначення зображень за допомогою ідентичної до мозку математичної мережі – це сучасна інформаційна технологія, яка допомагає не тільки розпізнавати, а й розуміти зображення, зроблені в реальному світі, а також перетворювати їх в цифровий варіант [3].

На сьогоднішній день розпізнавання образів дуже широка діяльність, за допомогою неї розпізнаються обличчя, відбитки пальців, автомобільні номери, штрихкоди, проводиться сортування сміття, листів, посилок, різних поштових відправлень, проводиться геологічна і топографічна розвідка місцевості [4].

Говорячи математично: розпізнавання образів – науковий напрямок, пов'язаний з розробкою принципів і побудовою систем, призначених для визначення приналежності даного об'єкта до одного із заздалегідь виділених класів об'єктів [3].

Ця сфера дослідження включає в себе багато процесів і функцій, наприклад розширену базу даних, цифрове навчання, аналізування даних і безпосередньо розпізнавання зображень і представлених образів [4]. При несистематизованому і ненаправленому пошуку схожих об'єктів з безлічі об'єктів, їх можна перераховувати нескінченно довго і не прийти до завершення з кінцевою ймовірністю [3].

В окремих випадках об'єкти характеризуються такими ідентифікаційними параметрами (ознаками), як форма, колір, положення, рухливість, за відмітними особливостями, їх комбінації і т.д. виходячи з цього їх можна класифікувати [5]. Тому ставиться завдання виділити в вступному відеопотоку об'єкти певного класу [6].

Розпізнавання зображень – одна з найбільш основних функцій сучасних телефонів, комп'ютерів і систем відео спостереження,

універсальність якої може забезпечити величезний набір функцій. Наприклад, є функції особистого користування та комерційного, такі як [6]:

- виробництво (оцінка якості виготовлених деталей, якість трубопроводів, безконтактний аналіз складу сумішей і рідин) [5];
- космічний простір (оцінка стабільності атмосфери Землі за даними багаторічного спостереженням, топографічна оцінка правдивості планети та аналіз складу різних небесних тіл: астероїди, комети, метеори тощо) [1];
- медицина та охорона здоров'я (діагностика онкологічних новоутворень, аналіз електрокардіограм, оцінка якості складу в фармакології, розпізнавання осіб та інших біометричних даних (відбитки пальців, сітківки ока, форми носа, губ, вух, очей, форми обличчя, волосся, тон шкіри, форми рук, походи і т. д.) [3];
- логістика (оцінка якості маршруту слідування і оптимізації пересувань технічних пристроїв, вироблення упередження перешкод і системи безпілотної навігації: морської, сухопутної, повітряної) [6];
- управління складськими ресурсами (автономне розміщення складських ресурсів);
- доставка доставки (системи автономної доставки продуктів);
- інспекція (контроль якості документів, кабелів і ліній електропередач, а також мостів, тунелів і та інших підвісних пристроїв);
- сервіс (системи аналізу діагностики автомобілів і верстатів);
- сільське господарство (контроль якості дозрівання фруктів, положення укладання ґрунту, стан плодово-ягідних рослин на предмет захворювань, оцінка якості виробничих продуктів - масло, молоко, сир, соки; оцінка справжності лісових масивів по даним зрізів деревини) [5];
- безпека – аналізування та програмування контенту для певного користувача, розпізнавання букв і штрих-кодів, розпізнавання автомобілів номерів, номерів на контейнерах і поїздках і вагонах [4].

1.2 Методи РЗ

Незважаючи на всі існуючі технології, комп'ютери не здатні аналізувати такий же величезний обсяг інформації, як людський мозок. Вчені намагаються наблизити дану технологію до людини. Серед найбільш відомих методів розпізнавання зображення значаться: пошук за шаблоном, пошук за аналітичною формою, класифікація за положенням, кольором [1].

Методи розпізнавання образів поділяються на [3]:

- метод перебору – проводиться порівняння з базою даних, де для кожного виду об'єктів представлені всілякі модифікації відображення;
- глибокий аналіз характеристик образу – це може бути визначення різних геометричних характеристик: звуковий зразок, в цьому випадку, піддається частотному, амплітудному аналізу та іншим видам аналізу;
- статичні класифікації – описова характеристика при розкладанні досліджуваного предмета на складові і дослідження кожного з них [4];
- штучні нейронні мережі (НМ) – даний метод вимагає великої кількості прикладів завдання розпізнавання при навчанні, або спеціальної структури нейронної мережі, що враховує специфіку даного завдання [5].

Методика віднесення елемента до якого-небудь класу називається вирішальним правилом, а метрика-спосіб визначення відстані між елементами універсального безлічі. Чим менше ця відстань, тим більше схожими є об'єкти [4].

Для того, щоб навчити мережу розпізнавати що-небудь, їй пропонуються зображення зі спеціальними мітками, які показують до якого типу можна віднести ту або іншу картинку. Задається певний напрямок значень, угруповання ознак допомагає нейронної мережі однозначно сказати, з якими зображеннями вона має справу [2].

Також, необхідно видавати дані, які не будуть суперечити один одному, інакше мережа буде визначати кілька образів. Створення нейронної мережі для розпізнавання зображень включає в себе кілька етапів [6]:

- збірка величезної бази даних;
- визначення типології образів;
- вибір характеристик і процесу навчання;
- саме навчання;
- перевірка навчання;
- коригування при необхідності;
- вербалізація.

Таким чином НМ може проектувати способи розпізнавання образів і масштабувати моделі до необхідних величин об'єднуючи в собі величезний потік даних [5].

1.2.1 Нейромережеві методи РЗ

Нейромережеві методи – це методи, що базуються на застосуванні різних типів НМ. Основні напрямки застосування різних НМ для розпізнавання образів і зображень [5]:

- застосування для вилучення ключових характеристик або ознак заданих образів;
- класифікація самих образів або вже витягнутих з них характеристик (в першому випадку витяг ключових характеристик відбувається неявно всередині мережі);
- рішення оптимізаційних задач.

Архітектура штучних НМ має деяку схожість з природними нейронними мережами. НМ, призначені для вирішення різних завдань, можуть істотно відрізнятися алгоритмами функціонування, але їх головні властивості наступні.

НМ складається з елементів, які називаються формальними нейронами, які самі по собі дуже прості і пов'язані з іншими нейронами. Кожен нейрон перетворює набір сигналів, що надходять до нього на вхід у вихідний сигнал. Саме зв'язки між нейронами, що кодуються вагами,

відіграють ключову роль. Одна з переваг НМ (а так само недолік при реалізації їх на послідовній архітектурі) це те, що всі елементи можуть функціонувати паралельно, тим самим істотно підвищуючи ефективність вирішення завдання, особливо в обробці зображень [4]. Крім того, що НМ дозволяють ефективно вирішувати багато завдань, вони надають потужні гнучкі та універсальні механізми навчання, що є їх головною перевагою перед іншими методами (імовірнісні методи, лінійні роздільники, вирішальні дерева тощо) [6]. Навчання позбавляє від необхідності вибирати ключові ознаки, їх значимість і відносини між ознаками. Але тим не менше вибір вихідного уявлення вхідних даних (вектор в n -мірному просторі, частотні характеристики, вейвлети, тощо), істотно впливає на якість рішення і є окремою темою. НМ мають гарну узагальнюючу здатність (краще ніж у вирішальних дерев), тобто можуть успішно поширювати досвід, отриманий на кінцевому навчальному наборі, на всі безліч образів [5].

Розглянемо найпопулярніші нейромережеві методи РЗ, поділяючи їх за топологією нейромоделей класифікаторів [6].

1.2.2 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) – спеціальна топологія НМ, запропонована в 1988 році і призначена для розпізнавання зображень. Архітектура натхненна деякими особливостями кори головного мозку, в якій були відкриті прості клітини і складні клітини [6]. Прості клітини реагують при сприйнятті прямих ліній під різними кутами, реакція складних клітин пов'язана з певним набором простих клітин. У CNN використовується три види шарів: згортки, пулінга (також званий шаром підвибірки або субдіскретизації) і повнозв'язний шар. Структура мережі – односпрямована, багат шарова, для навчання, як правило, використовується метод зворотного поширення помилки, функція активації нейронів визначається дослідником. Назву ця архітектура отримала завдяки використанню операції згортки, яка

полягає в поелементному множенні кожного фрагмента зображення на ядро згортки і подальшого запису результату у відповідну позицію вихідного зображення [1, 5].

Оператор згортки становить основу згорткового шару мережі. Шар складається з набору ядер і обчислює згортку вихідного зображення з попереднього шару за допомогою цього набору, на кожній ітерації додаючи відповідне ядру зміщення [4]. Результат даної операції – додавання і масштабування вхідних пікселів, ядра можна отримати з навчального набору методом градієнтного спуску, аналогічно обчисленню ваг в повнозв'язних мережах, які так само можуть виконувати ці операції, але зажадають набагато більшого часу і даних для навчання [3]. Однак, в порівнянні з повнозв'язними мережами, CNN використовують більшу кількість гіперпараметрів – параметрів, що задаються до початку навчання, таких як: глибина (кількість ядер і коефіцієнтів зміщення в шарі), висота і ширина кожного ядра, крок (зміщення ядра на кожному кроці при обчисленні наступного пікселя). Шар пулінга отримує на вихід окремі фрагменти зображення (як правило, 2x2) і об'єднує їх в одне значення [5].

Існують різні способи агрегації, зазвичай, вибирається найбільше значення з отриманого фрагмента. Незважаючи на те, що архітектура CNN була розроблена в кінці минулого століття, апаратна складова, яка дозволила ефективно застосовувати їх і їм подібні архітектури з'явилася порівняно недавно, і з того моменту згорткові мережі є найпоширенішим методом в задачі розпізнавання зображень [6].

1.2.3 Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) – вид НМ, в яких зв'язки утворюють спрямовану послідовність, що дозволяє обробляти серії подій за часом і послідовні просторові ланцюжки [4]. Їх основна відмінність від традиційних архітектур — логіка роботи мережі, в рекурентних мережах

нейрон взаємодіє сам з собою. На вхід, як правило, подається деяка послідовність. Послідовність поелементно передається одним і тим же нейронам, який свій сигнал знову передає собі разом з наступним елементом послідовності поки вона не буде оброблена повністю [6].

Для моделювання пам'яті в нейронній мережі подібним образів вводимо новий вимір в опис процесу її роботи – час. Нехай мережа обробляє послідовність даних, наприклад, деякий текст. Кожен елемент послідовності надходить на нейрон в новий умовний момент часу, до цього моменту в ньому вже є накопичений з початку роботи досвід. Основна відмінність типів рекурентних нейронів один від одного криється в обробці пам'яті всередині них [5].

Класичний підхід – додавання векторів сигналу і пам'яті і обчислення активації від суми, наприклад, гіперболічним тангенсом. Отримуємо звичайну мережу з одним прихованим шаром. Але пам'ять, реалізована подібним чином, виходить досить короткою. Оскільки інформація в пам'яті кожен раз змішується з інформацією в новому сигналі, через 5-7 ітерацій інформація повністю перезаписується. При роботі з короткими послідовностями це не критично, але при обробці великих обсягів даних, то збережені в пам'яті закономірності будуть застосовуватися тільки на невеликих обсягах даних, а масштабні шаблони взагалі не підлягатимуть виявленню. Це дуже умовний опис досить фундаментальної проблеми нейронних мереж, яка називається проблемою зникаючого градієнта [6].

Щоб впоратися з цією проблемою, була придумана модифікація Long Short-Term Memory Recurrent Neural Network (LSTM-RNN). Така мережа управляється за допомогою рекурентних вентилів забування. Поширення помилки назад за часом відбувається через умовно нескінченну кількість віртуальних шарів, що дозволяє під час навчання зберігати в пам'яті набагато більшу кількість часових інтервалів, завдяки чому, навіть великі паузи між важливими подіями враховуються, значить компоненти з різною частотою

виникнення можуть змішуватися. Топології подібних мереж можуть змінюватися залежно від особливостей процесу [6].

Безпосередньо рекурентні мережі для завдання розпізнавання зображень використовувалися нечасто, але в 2015 році дослідники з Microsoft з серйозним відривом виграли змагання в цій дисципліні, використовуючи нову нейромережу, архітектуру якої вони назвали залишковою (Residual Neural Network). Але при подальшому розгляді з'ясувалося, що вона є, практично, повним еквівалентом LSTM-RNN [6].

1.2.4 Порівняння нейромережевих методів РЗ

Виконаємо порівняння нейромережевих методів РЗ. Для порівняння було обрано 3 основні топологічні підходи: CNN, RNN та багатошарова НМ. Результати порівняння представимо в табличному вигляді – табл. 1.1.

Таблиця 1.1 – Результати порівняння нейромережевих методів РЗ

	Багатошарова НМ	CNN	RNN
Швидкість класифікації	Середній рівень	Високий рівень	Низький рівень
Надійність результатів	Середній рівень	Високий рівень	Високий рівень
Вимоги до вхідних даних	Середній рівень	Середній рівень	Низький рівень
Швидкість навчання	Середній рівень	Високий рівень	Низький рівень
Методи навчання	Середній рівень	Середній рівень	Низький рівень
Інтерпретовність моделі	Середній рівень	Високий рівень	Низький рівень

З результатів порівняння можна зробити наступні висновки. Загалом нейромережеві методи РЗ відрізняються високими вимогами до часу навчання моделі (в порівнянні з класичними методами). Проте, точність їх роботи вище. Однак, при порівнянні в середині власне нейромережевих

методів – результати CNN найточніші. Так само CNN не має особливих вимог до вхідних даних, адже для навчання використовуються звичайні методи, як і в багат шарових НМ, на відміну від RNN, де через специфічні методи навчання виникають особливі вимоги до вхідних даних. Також CNN відрізняє високий рівень інтерпретовності моделі в той час, як модель на базі RNN та багат шарових НМ зазвичай є значно структурно складнішими та менш зрозумілими.

1.3 Використання ВЗ при РЗ

На даний момент зростає інтерес до систем паралельної або розподіленої обробки зображень. В першу чергу це пов'язано з тим, що з'явилася потреба в обробці ВЗ [7]. Спостерігається стійка тенденція до збільшення розмірів формованих зображень. Зображення земної поверхні великого розміру стають все більш доступними, існує кілька супутників, які надають інформацію в режимі реального часу у вільному доступі. Сучасні електронні мікроскопи забезпечують формування наборів даних високої роздільної здатності об'ємом в кілька гігабайт [8].

Збільшення розміру зображення породжує проблеми при їх обробці, зберіганні і передачі даних. В першу чергу це пов'язано з тим, що постійне нарощування обчислювальної потужності персональних комп'ютерів сформувало стійкий вибір інтерактивного способу проведення досліджень в області обробки зображень: підбір параметрів обробки, момент завершення ітераційних процедур здійснюється користувачем візуально. Разом з тим обсяг оброблюваних даних можна порівняти з об'ємом оперативної пам'яті сучасних високопродуктивних робочих станцій або перевищує його, тоді як широко поширені універсальні програмні системи обробки зображень рекомендують, щоб подвоєний обсяг найбільшого оброблюваної зображення займав не більше 75% наявної оперативної пам'яті [9]. При недотриманні цієї умови час виконання навіть найпростіших операцій над великоформатним

зображенням стає неприйнятно великим для користувачів інтерактивних систем [10].

При проведенні реальних досліджень зберігання досить представницького набору ВЗ може займати значну частину обсягу даних постійного зберігання, що може істотно знизити ефективність роботи всього комп'ютера. Якщо робота виконується групою дослідників, то неминуче дублювання всього набору або частини зображень. При обробці створюється деяка кількість проміжних зображень, обсяг яких може перевищувати вихідний в кілька разів, загострюючи проблему з вільним місцем на диску комп'ютера [8].

Централізоване розміщення ВЗ на спеціалізованих сховищах даних пред'являє підвищені вимоги до пропускнуєї здатності мережі, причому, як правило, один запит на візуалізацію зображення породжує істотний графік, оскільки запитується і передається весь обсяг даних, незважаючи на те, що реально відображатися при цьому може істотно менша частина. Обробка зображення на комп'ютері користувача збільшує цей трафік як мінімум удвічі. Обробка зображення безпосередньо на сервері зберігання даних передбачає істотне збільшення обчислювальної потужності сервера, а значить, і вартість зберігання даних на ньому [8].

Обробка зображень з використанням паралельних або розподілених систем вирішує багато з розглянутих вище проблем. Однак більшість дослідників вирішують завдання організації паралельних систем обробки зображень у відриві від того, як і де зберігаються дані зображень [11]. Разом з тим ефективність використання розподіленої системи при обробці великоформатних зображень багато в чому визначається способом організації доступу програм обробки до даних зображень і зручністю візуалізації результатів обробки користувачами [12].

Саме тому у роботі слід на початку виконати аналіз найбільш поширених варіантів організації паралельної обробки зображень і їх вузьких місць, після чого запропонувати новий підхід до організації зберігання даних

в багатопроцесорних системах різної архітектури, що базуються на концепції розподілених зображень [10].

1.3.1 Проблеми реалізації концепції розподіленого зображення

Основним показником ефективності системи обробки зображень є час реакції на запит користувача. Цей час складається з часу вибірки даних з пристрою зберігання, часу обробки зображення (якщо це необхідно для виконання запиту), часу передачі даних результуючого зображення на комп'ютер користувача для його візуалізації або наступної обробки.

Вузьким місцем паралельних систем обробки зображень є попередній етап розсилки даних вихідних зображень по комп'ютерах розподіленої системи і завершальний етап збору оброблених фрагментів в єдине зображення [9].

Спроби розпаралелити або істотно зменшити ці необхідні, але непродуктивні етапи розподіленої обробки зображень призводять до ідеї розподіленого зображення.

У цьому випадку дані оброблюваних зображень зберігаються безпосередньо на комп'ютерах, що виконують паралельну обробку. Кожна функція паралельної програми обробляє той фрагмент зображення, який розташований на комп'ютері, де виконується дана задача, результат обробки зберігається тут же, як частина нового розподіленого зображення, отриманого в результаті роботи всіх завдань, що беруть участь в розподіленій обробці.

Незважаючи на очевидність даної ідеї вона не набула поширення, тому що дослідники не змогли запропонувати задовільних рішень виникаючих при цьому проблем:

- як виконати декомпозицію (розбиття), близьку до оптимальної для апіорі невідомої наступної задачі обробки;

- як вирішити проблеми збалансованості завантаження комп'ютерів, що беруть участь в обробці при заздалегідь виконаній декомпозиції даних;
- як забезпечити достатній рівень відмовостійкості розподіленого зберігання фрагментів зображень;
- як забезпечити віртуальну цілісність розбитого зображення;
- як забезпечити прийнятний рівень інтерактивності системи при візуалізації розподілених зображень.

З одного боку при створенні розподіленого зображення заздалегідь виконується декомпозиція даних. З іншого боку при цьому апіорі невідомо як буде надалі оброблятися зображення, на які фрагменти оптимально поділити зображення з точки зору збалансованості завантаження комп'ютерів, що беруть участь в обробці. Причому, якщо програми працюють не в режимі монопольного використання комп'ютера, то передбачити реальне завантаження конкретного комп'ютера не представляється можливим, і необхідно здійснювати динамічне балансування в умовах вже існуючого розбиття даних (заздалегідь виконаної декомпозиції даних) або виконувати перерозподіл даних по комп'ютерах в процесі обробки, знижуючи ефективність обробки в цілому. У разі динамічного балансування при частих істотних змінах навантаження в процесі обробки такий перерозподіл може звести нанівець весь виграш від попереднього розміщення даних по комп'ютерах обробки.

У роботі [11] наявність запропонованого авторами методу збалансованого завантаження комп'ютерів кластера для варіанту централізованого менеджера розподілу даних переважило можливий виграш від розпаралелювання введення/виведення даних у разі заздалегідь розподілених по комп'ютерах зображень, тому що при такому варіанті розподілу даних зазначений алгоритм балансування завантаження не працює.

Далі розглянемо вибір способу декомпозиції, який може бути використаний при розпаралелюванні більшості операцій обробки зображень.

1.3.2 Формалізація опису процесу обробки зображень

Виділяють три види операцій (рис. 1.1), актуальних для більшості застосунків в області обробки зображень, включаючи створення систем технічного зору [12].

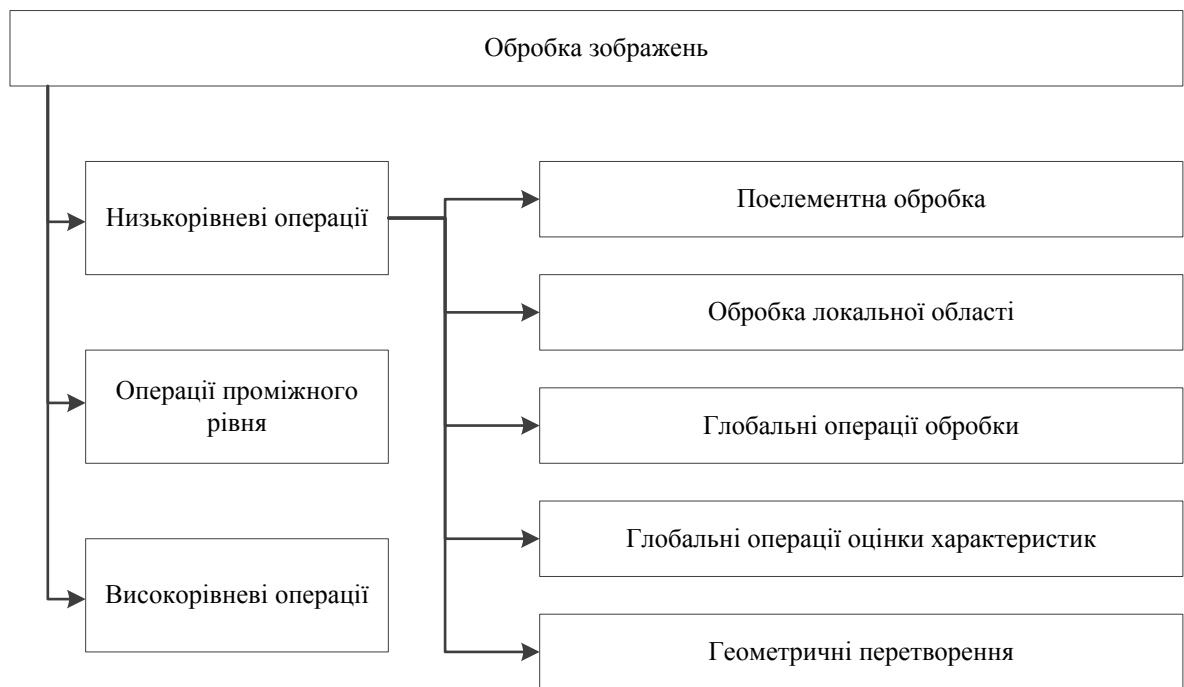


Рисунок 1.1 – Типологія операцій обробки зображень

Перший вид – низькорівневі операції [7]. Ці операції виконують перетворення над усіма точками зображень і формують або змінене зображення, або якусь векторну структуру або навіть одне значення. Зазвичай обчислення мають локальну природу і визначаються як набір операцій, необхідних для формування відповідно пікселя зображення, елемента вектора або результуючого значення. Прикладами таких операцій є операції фільтрації, підвищення контрасту, підкреслення контурів, геометричні перетворення зображень, формування гістограм, отримання різного роду статистичних параметрів і т. п.

Другий вид – операції проміжного рівня [11]. Сюди відносять різноманітні операції сегментації зображень, які формують якісь структурні описи зображень. Наприклад, списки описів меж об'єктів на зображенні.

Третій вид – високорівневі операції [12]. На цьому рівні виконується семантична обробка різного роду структурних описів, отриманих на проміжковому рівні, формується якесь рішення. Як приклади можна вказати різного роду операції розпізнавання образів, семантичної інтерпретації сцен і т. п.

Слід зазначити, що саме виконання якоїсь послідовності операцій низького рівня над набором вихідних зображень є відправною точкою для багатьох типових додатків в області обробки зображень і систем технічного зору. Основні витрати в розглянутих додатках пов'язані з низькорівневими операціями [8]. Ці операції вимагають значних обчислювальних ресурсів, ресурсів зберігання. Час реакції всього додатка в цілому визначається в більшості випадків саме часом, що витрачається на виконання низькорівневих операцій.

У загальному вигляді процес вирішення завдання обробки зображень передбачає або початкове вказівку послідовності деякого числа кроків рішення, або можливість декомпозиції методу рішення на складові частини, що реалізуються у вигляді низькорівневих операцій [13]. На кожному кроці рішення обробляються та/або виробляються елементи рішення [14].

Формально цей процес можна представити наступним чином [14].

На I -му кроці формуються за допомогою перетворення F^I необхідні елементи рішення, що подаються, в загальному випадку, деякою безліччю параметрів S^I і безліччю зображень $\{X\}^I$:

$$\{\{X\}^I, S^I\} = F^I(\{X\}, S, P, \{X\} \subset \bigcup_{i=0}^{I-1} \{X\}, S \subset \bigcup_{i=0}^{I-1} S^I), \quad (1.1)$$

де $\{X\}, S, P$ – множини вхідних зображень, вхідних параметрів і власних параметрів перетворення F^I відповідно. Будь-яке з цих наборів на певному конкретному кроці може бути порожнім;

$\{X^0\}, S^0$ – вихідні множини зображень і параметрів для даної задачі.

Кроки рішення з відповідними елементами рішення зручно представляти у вигляді графа рішення даної задачі. Множині кроків розв'язку (операціям алгоритму F^I) ставиться у взаємно-однозначну відповідність деяка множина точок – вершин. Якщо елемент рішення однієї операції використовується при виконанні іншої операції, то відповідні вершини з'єднуються дугою, спрямованою з тієї точки, де формується даний елемент рішення. З кожної вершини виходить стільки дуг, скільки формується на даному кроці зображень і параметрів. У випадку, коли елемент рішення є аргументом для декількох операцій, які виходять, дуг може бути більше [14].

При обговоренні проблем організації процесу обчислень в системах обробки зображень немає необхідності в детальній конкретизації окремих кроків технології обробки, це завдання розробника прикладної програми, що реалізує певний етап вирішення. В даному випадку істотний тільки інформаційний тип програми обробки зображення, тобто спосіб її роботи з даними зображень [15].

Більшість дослідників виділяють наступні види низькорівневих операцій над зображеннями:

- поелементна обробка (PO – point operators),
- обробка локальної області (LNO – local neighborhood operators) або локальна обробка ковзаючим вікном;
- глобальні операції обробки (GO – global operators), як правило, засновані на двовимірному перетворенні Фур'є або іншому подібному перетворенні;
- глобальні операції редукції або операції обчислення параметрів зображення, в. т. ч. статистичних;

– геометричне перетворення.

Більш загальна класифікація підрозділяє операції на глобальні і локальні.

Методи глобальної обробки зображень [16] характеризуються тим, що для обчислення значення кожної точки результуючого зображення $Y = \{y_{ij}\}$, в принципі, використовуються значення всіх точок вихідного зображення X :

$$y_{ij} = F_{ij}(X), y_{ij} \in Y, \quad (1.2)$$

де F_{ij} – оператор перетворення для точки y_{ij} ;

i, j – тут і далі цілочисельні координати точок цифрового зображення.

Завданням глобальних операцій редукції є отримання деякої оцінки (або опису) оброблюваного зображення [17]. Під оцінкою можна розуміти, наприклад, деякі інтегральні характеристики вихідного зображення, Геометричні характеристики різних об'єктів на зображенні, віднесення вихідного зображення або його частин до заданих класів. Перетворення редукції R за множиною зображень $\{X\}$ визначає необхідні параметри з множини S [17]:

$$S = R(\{X\}). \quad (1.3)$$

Методи локальної обробки зображень формують кожну вихідну точку за результатами аналізу її околиці на вхідному зображенні. У свою чергу, вони включають в себе поелементні операції і обробку ковзаючим вікном [18].

Поелементна обробка включає в себе як унарні, так і бінарні операції. Унарні операції поелементної обробки кожну точку вихідного зображення X перетворюють у відповідну їй точку вихідного зображення Y [13, 18]:

$$y_{ij} = F(x_{ij}), \quad (1.4)$$

де $x_{ij} \in X$, $y_{ij} \in Y$.

Бінарні операції поелементної обробки формують кожну точку вихідного зображення Y як результат якоїсь операції над відповідними точками двох вихідних зображень X' і X'' [13, 18]:

$$y_{ij} = F(x'_{ij}, x''_{ij}), \quad (1.5)$$

де $x'_{ij} \in X'$, $x''_{ij} \in X''$, $y_{ij} \in Y$.

Операції локальної обробки ковзаючим вікном формують зображення Y , кожна точка якого є результатом перетворення над деякою відповідною їй околицею Q точок вихідного зображення [17]:

$$y_{ij} = F(\{x_{i+k, j+l} \mid (k, l) \in Q\}). \quad (1.6)$$

Найбільш популярним видом обробки в даному випадку є зважене підсумовування околишніх відліків, а в якості вікна обробки – квадратна околиця, симетрично розташована щодо поточного відліку [18]:

$$y_{i,j} = \sum_{k=-K}^K \sum_{l=-L}^L a_{k,l} \cdot x_{i+k, j+l}, i = \overline{0, M-1}, j = \overline{0, N-1}, \quad (1.7)$$

при $K = L = 1$ це відповідає локальній обробці ковзаючим вікном 3×3 , при $K = L = 2$ – вікном 5×5 і т. д.

Спеціальним класом операцій над локальною околицею є рекурсивна обробка локальної околиці.

В даному випадку кожна точка результуючого зображення Y залежить не тільки від точок деякої околиці Q вихідного зображення X , але і від точок околиці Q' самого результуючого зображення Y [18]:

$$y_{i,j} = F(\{x_{i+k,j+l}, y_{i+m,j+n} \mid (k,l) \in Q, (m,n) \in Q'\}). \quad (1.8)$$

Зазвичай такі операції реалізуються за допомогою наступної ітераційної процедури:

$$x_{i,j}^p = \sum_{k=-K}^K \sum_{l=-L}^L a_{k,l}^p \cdot x_{i+k,j+l}^{p-1}, i = \overline{0, M-1}, j = \overline{0, N-1}, p = \overline{0, P}. \quad (1.9)$$

Тобто до вихідного зображення x_0 послідовно застосовується P операцій обробки [18].

Геометричні перетворення складно віднести як до глобальних, так і до локальних операцій. При геометричних перетвореннях задається деяка функція перетворення координат G :

$$(k,l) = G(i,j): \begin{cases} k = G_1(i) \\ l = G_2(j) \end{cases} \quad (1.10)$$

яка ставить у відповідність деякій точці початкового зображення з координатами (i,j) точку результуючого зображення з координатами (k,l) . У загальному випадку, кожна точка результуючого зображення $Y = \{y_{kl}\}$ формується як результат деякого комплексування тих точок вихідного зображення X , які просторово близькі до координаті $(i,j) = G^{-1}(k,l)$:

$$y_{kl} = F(\{x_{ij}, x_{i+1j}, x_{ij+1}, x_{i+1j+1} \mid [i,j] = G^{-1}(k,l)\}), y_{ij} \in Y, \quad (1.11)$$

де $[.]$ означає формування цілочисельних координат шляхом відкидання дробової частини [18].

Дана операція не відноситься до глобальних, оскільки для обчислення деякої точки результуючого перетворення необхідно не більше чотирьох точок вихідного зображення. Але і локальної операцією її не можна назвати, так як розташування цих чотирьох точок в загальному випадку не знаходиться в локальній околиці формованої точки [13, 18].

Таким чином, незважаючи на те, що основною особливістю зображень як даних є їх двовимірність, більшість алгоритмів і методів обробки зображень носять послідовний характер. Це проявляється, з одного боку, в структурі обчислень «всередині» алгоритму. Дуже популярно використання порядкової, послідовної розгортки відліків зображень і перетворень, що мають локальний характер (поелементних або на основі ковзного вікна). З іншого боку, значне число методів складної обробки зображень можуть бути реалізовані як послідовне застосування деяких закінчених типових операцій над зображеннями. Саме ця обставина визначає ефективність застосування різноманітних програмних систем обробки зображень, розроблених для використання на універсальних комп'ютерах, до вирішення широкого спектру дослідницьких і прикладних завдань обробки та аналізу відеоінформації [18].

На щастя, структура більшості алгоритмів низькорівневих операцій така, що дозволяє виконати природне розпаралелювання алгоритмів на основі закладеної в них регулярності.

1.3.3 Перспективи розпаралелювання обробки ВЗ

Разом з тим швидкість передачі даних в локальних мережах зростає істотно швидше, ніж аналогічний показник для дискових запам'ятовуючих пристроїв: швидкість обміну даними в мережах на базі технології Gigabit Ethernet знаходиться приблизно на одному рівні з дисками типових

конфігурацій комп'ютера, але при переході на технологію 10GbE ситуація кардинально зміниться – візуалізація великоформатного зображення буде виконуватися швидше при віддаленому, а не локальному зберіганні даних. В таких умовах використання розподілених систем для зберігання і обробки ВЗ стає актуальним [14].

Іншим важливим фактором при обробці ВЗ є розмір оперативної пам'яті. Незважаючи на те, що поява 64-бітних операційних систем відсунуло обмеження на розмір оперативної пам'яті, істотне нарощування оперативної пам'яті комп'ютера щодо типових рішень досить коштовне. Ефективним рішенням в даному випадку є обробка зображення по частинах на різних комп'ютерах. У поєднанні з можливістю паралельної організації таких обчислень в розподілених системах, особливо при організації їх на основі кластерних обчислювальних систем, цей підхід є перспективним [16].

1.4 Підходи до розпаралелювання обробки зображень

Розглянемо основні варіанти розпаралелювання низькорівневих операцій обробки зображень [18].

Найбільш природним варіантом розпаралелювання є декомпозиція за даними на базі вихідного зображення: відліки результуючого зображення розбиваються на непересічні фрагменти, і кожен фрагмент формується паралельно на окремому вузлі багатопроцесорного кластера або комп'ютері розподіленої системи, потім фрагменти, при необхідності, збираються в єдине зображення. Декомпозиція зображення може бути виконана наступними способами: одновимірна декомпозиція за однією з координат, двовимірна декомпозиція. Відповідна декомпозиція вихідного зображення залежить від типу операції обробки [16].

Поелементна обробка. При паралельному виконанні алгоритмів, що здійснюють експериментальну обробку, розбиття вихідного зображення (або декількох вихідних зображень) повністю відповідає розбиттю вихідного

зображення [17]. Ефективність поелементної обробки як одиночної операції, так і послідовності таких операцій не залежить від способу розбиття зображень на фрагменти [18].

Локальна обробка ковзаючим вікном. Дані алгоритми дуже добре розпаралелюються, але при цьому необхідно вихідне зображення розбивати на перекриваються фрагменти. Розмір перекритих областей залежить від розмірів вікна обробки. Тут так само можливий варіант як одновимірної, так і двовимірної декомпозиції v . Оскільки в результаті обробки розглянутого типу операцій формується розподілене зображення, що містить непересічні фрагменти, то при послідовному застосуванні даних операцій необхідно проводити обмін, перекривати частинами перед черговим етапом. Часові витрати на таке узгодження даних розподіленого зображення при різних варіантах декомпозиції різні і залежать від кількості фрагментів і параметрів комунікаційного середовища [17]. Наприклад, при невеликому числі фрагментів і значній величині латентності одновимірний варіант декомпозиції більш кращий. Однак при збільшенні числа фрагментів розмір переданої при синхронізації інформації значно менше для варіанту двовимірної декомпозиції [16].

Глобальні операції редукції. Оператори редукції, як правило, є адитивними, тому даний вид операцій практично нечутливий до способів розбиття оброблюваних зображень на фрагменти. Але для визначення деяких параметрів корисна наявність перекритих областей у фрагментів зображення [18].

Глобальні операції обробки. Найбільш простим варіантом розпаралелювання для цих операцій є будь-який варіант декомпозиції вихідного зображення і реплікація повного вихідного зображення по всіх комп'ютерах, що беруть участь в обробці. Якщо повна реплікація оброблюваного зображення з яких-небудь причин неможлива, то при паралельному виконанні операцій даного типу буде здійснюватися інтенсивний обмін даними між паралельно виконуються завданнями

обробки. В цьому випадку оптимальний вибір розподілу даних дуже часто залежить від використовуваного варіанту алгоритму виконання операції, і заздалегідь зробити цей вибір дуже складно [16].

Геометричне перетворення. Даний вид операцій включає в себе операції масштабування і повороту, а так само нелінійного перетворення координат загального вигляду [8]. З точки зору мінімізації витрат на передачу даних при здійсненні геометричних перетворень загального вигляду над розподіленими зображеннями найбільш оптимальним є розбиття на квадратні блоки. Однак при виконанні повороту зображення на невеликий кут більш ефективним є одновимірна декомпозиція [10]. Причому як в тому, так іншому випадку вихідне зображення корисно розбивати на відповідні фрагменти з перекриттям [18].

Найбільш популярним варіантом організації обчислень при розпаралелюванні низькорівневих операцій на основі декомпозиції за даними є використання програми-менеджера в поєднанні з централізованим зберіганням зображень. Спеціальна програма-менеджер здійснює декомпозицію вихідних зображень і розподіл їх фрагментів за завданнями обробки, а також подальшу збірку результуючого зображення. Залежно від конкретного набору операцій графа рішення задачі обробки Програма-менеджер для зображень, що є проміжними елементами рішення, або виконує збірку повного зображення і подальшу розсилку відповідно до нової схеми декомпозиції, або виробляє обмін перекриваються частинами фрагментів зображення на вузлах обробки, якщо розбиття не змінюється при переході до наступного кроку обробки.

Ефективність такого варіанту паралельної або розподіленої обробки зображень суттєво знижується за наявності таких вузьких місць: канал доступу до підсистеми зберігання оброблюваних зображень, коли завдання, які обробляють фрагменти зображення [7], практично одночасно запитують необхідні дані, і власне програма-менеджер, що здійснює декомпозицію і

розподіл фрагментів за задач обробки, а також подальшу збірку результуючого зображення [18].

Вкрай бажано усунути або істотно скоротити необхідний, але непродуктивний етап розподіленої обробки зображень у вигляді розсилки даних вихідних зображень по комп'ютерах розподіленої системи і подальшого збору оброблених фрагментів в єдине зображення [17].

Це можливо, якщо дані оброблюваних зображень розподілити по комп'ютерах заздалегідь, тобто зберігати їх там, де виконується обробка. Представимо варіанти розпаралелювання у вигляді таблиці – табл. 1.2.

Таблиця 1.2 – Порівняння підходів до розпаралелювання обробки зображень

Критерії	Поелементна обробка	Локальна обробка ковзаючим вікном	Глобальні операції редуції	Глобальні операції обробки	Геометричне перетворення	Використання програми-менеджера
Рівень залежності від початкового розбиття зображення	Середня	Висока	Висока	Середня	Середня	Середня
Часові витрати на препроцесінг	Середня	Висока	Висока	Висока	Висока	Висока
Частка накладних витрат	Низька	Середня	Висока	Висока	Висока	Висока
Рівень розпаралелювання	Високий	Середній	Середній	Середній	Середній	Високий
Залежність від організації паралельної системи	Низька	Середня	Висока	Висока	Висока	Висока

Кожна програма обробляє той фрагмент, який розташований на даному вузлі комп'ютерної мережі, результат обробки зберігається тут же, як частина нового розподіленого зображення, отриманого в результаті роботи всіх вузлів, що беруть участь в розподіленій обробці [18]. Таким чином, основною одиницею зберігання в такій розподіленій системі є розподілене зображення.

Як видно з результатів порівняння усі методи мають слабкі та сильні сторони. Це пояснюється різними підходами до початкової сегментації та розбиття зображень. Саме тому актуальною є задача розробки нових методів організації паралельної обробки ВЗ для подальшого використання в задачах РЗ.

Новий підхід має виправляти розглянуті недоліки, а саме вимагати низького рівня препроцесінгу зображень та мати високий ступінь розпаралелювання. Більш того новий підхід має відрізнятися малою часткою накладних витрат. Також важливим є розробка підходу, який буде менше залежати від характеристик самої обчислювальної системи та від оцінки складності завдання розпізнавання (адже вона апріорі не відома). Тобто головною характеристикою методу буде саме зображення.

1.5 Постановка завдання

Дослідити та реалізувати методи динамічної організації та обробки ВЗ в паралельних обчислювальних системах для подальшого використання їх в задачах РЗ.

1.6 Висновки за розділом 1

Під час роботи над першим розділом було розглянуто проблемну область РЗ та використання ВЗ в якості вхідних даних в задачах РЗ. Було з'ясовано, що найбільш доцільним видається декомпозиція розподіленого

зображення у вигляді фрагментів, що перекриваються. А у якості нейромоделей для задач РЗ буде використовуватися CNN.

2 МАТЕРІАЛИ ТА МЕТОДИ

В першому розділі було проаналізовано сучасне становище проблемної області паралельної обробки ВЗ в задачах РЗ. Було проаналізовано основні методи паралельної обробки ВЗ та нейромережевого РЗ.

В цьому розділі буде запропоновано новий підхід до організації зберігання даних на багатопроцесорних системах різної архітектури, що базується на концепції розподілених зображень. Розглядається заснований на пропонованій концепції децентралізований метод динамічного балансування і управління паралельним обчислювальним процесом, а також організація візуалізації розподілених ВЗ.

Друга частина розділу буде присвячена висвітленню питанню використання CNN для РЗ.

2.1 Структура розподіленого зображення

Розподілене зображення – це структура даних, що визначає спосіб і параметри розбиття зображення на фрагменти, список комп'ютерів, де знаходяться ці фрагменти, місце їх розміщення і формат зберігання. По суті, це опис того, як зібрати із зазначених у цій структурі фрагментів ціле зображення [8, 18].

Підсумовуючи зроблений вище аналіз варіантів декомпозиції зображень при виконанні різноманітних операцій обробки, слід зазначити, що найбільш доцільним представляється декомпозиція розподіленого зображення у вигляді фрагментів, що перекриваються [9].

Суть пропонованого підходу полягає в тому, що розмір необхідного перекриття фрагментів визначається не параметрами подальшої задачі обробки, оскільки вона апріорі невідома, а необхідністю вирішення проблем

збалансованості завантаження комп'ютерів і забезпечення достатнього рівня відмовостійкості розподіленого зберігання фрагментів зображень [16, 18].

Розподілене зображення визначається у вигляді набору фрагментів зображення, що перекриваються. Для кожного з M комп'ютерів фрагмент, який він зберігає, формується наступним чином.

Всі рядки зображення діляться на $2M$ блоків однакового розміру. Фрагмент розподіленого зображення на m -му комп'ютері містить два основних блоки з номерами $2m - 1$ і $2m$. Ці два основних блоки відповідають варіанту розбиття зображення на фрагменти, що непересічні.

Щоб отримати декомпозицію у вигляді фрагментів, що пересічні, яка одночасно могла б забезпечити мінімальну ступінь відмовостійкості, на даному комп'ютері додатково зберігаються два так звані тінюві блоки, тобто копії («тіні») примикають до нього основних блоків сусідніх комп'ютерів, блоків з номерами $2m - 2$ і $2m + 1$. У свою чергу, молодший основний блок зберігається в якості одного з тінювих блоків на комп'ютері з меншим номером, а старший – на комп'ютері з великим номером [17].

Саме дані основних блоків формуються на вузлі в процесі розподіленої обробки зображення. Після закінчення процесу обробки виконується обмін тінювими даними [18].

Така структура даних розподіленого зображення дозволяє відновити його при відмові одного з вузлів зберігання, а також для більшості операцій забезпечує можливість формувати відліки основних блоків без передачі даних між сусідніми комп'ютерами в процесі розподіленої обробки зображення [16].

Таким чином, розподілене зображення являє собою набір фрагментів зображення, що перекриваються, який зберігається розподілено. На кожному вузлі збереження фрагмент містить основні і тінюві дані. З точки зору зчитування даних вони рівноправні, але в процесі розподіленої обробки вузол формує тільки відліки основних блоків, тінюві він отримує після завершення

процесу обробки від вузлів, відповідальних за формування відповідних основних блоків [16].

2.2 Метод динамічного розподілу обчислювального навантаження

Запропонований принцип декомпозиції розподіленого зображення дозволяє реалізувати оригінальний алгоритм динамічного розподілу завантаження процесорів при виконанні операцій поелементної обробки або локальної обробки ковзаючим вікном [10].

Суть його розглянемо на прикладі взаємодії двох сусідніх вузлів, m -го та $(m+1)$ -го, при формуванні тієї частини результуючого зображення, яка міститься в блоках даних з номерами $2m$ і $2m+1$. Зауважимо, що кожен з вузлів має всю інформацію, щоб зробити цю роботу самостійно (або майже всю, в разі виконання операції локальної обробки ковзаючим вікном) [10], [16]. m -й вузол починає формувати рядки m -го блоку, починаючи з першого рядка в порядку зростання номерів рядків, $(m+1)$ -й вузол, в свою чергу, формує рядки $2m+1$ -го блоку, але починаючи з останнього рядка блоку і в порядку спадання номерів рядків. Після формування певної кількості рядків кожен вузол інформує свого сусіда про час, за який він виконав цю роботу. На підставі цієї інформації обчислюється планований номер рядка зображення, на якому вузли завершать таку спільну обробку [16].

Таким чином, в процесі роботи суміжні вузли рухаються назустріч один одному, повідомляючи про швидкість свого процесу обчислень при досягненні заздалегідь визначених моментів. При цьому прогнозований номер рядка зображення, на якому ці процеси зустрінуться, постійно коригується в залежності від поточного завантаження обчислювальних вузлів. Таким чином, всі процеси завершать свою роботу практично одночасно. Різниця в часі при цьому складе не більше, ніж час обробки одного рядка.

Одночасно кожен вузол бере участь у двох таких процесах, поперемінно формуючи рядки старшого блоку в порядку зростання номера і рядки свого молодшого блоку в порядку спадання [17].

В результаті буде сформовано результуюче зображення в повному обсязі, але розміщення його даних по комп'ютерах, що містять нове розподілене зображення, буде нерівномірним. Далі у фоновому режимі вузли зберігання щойно сформованого розподіленого зображення обміняються своїми даними з тим, щоб привести структуру розподіленого зображення до необхідного вигляду. Однак користувач може отримати результат свого запиту відразу по завершенні процесу обробки [16].

Треба визначити: починаючи з якого часу і наскільки часто необхідно вузлу інформувати свого сусіда про виконаному поточному обсязі роботи.

Найбільш простий варіант полягає в наступному: після того, як сформовано перший рядок, відправляється інформаційне повідомлення, що містить час його обробки. На підставі отриманої від суміжного вузла інформації вже може бути прогнозовано номер рядка, на якому процеси зустрінуться [9]. Точність такого прогнозу буде тим вище, чим більша кількість рядків буде оброблена перед відправкою такого повідомлення. Однак при істотній різниці в продуктивності суміжних вузлів ця кількість не може бути занадто великим [15].

Розглянемо, дещо ідеалізований випадок незбалансованої обчислювальної системи, у якій продуктивність одного з вузлів вище інших у $\mu > 1$ раз. У разі однакової продуктивності всіх M вузлів час обробки зображення розміром N рядків становить $T_0 = \left(N/M\right)\tau$, де τ – час обробки одного рядка типовим вузлом, кожен вузол при цьому обробляє по N/M рядків. При зростанні продуктивності одного з вузлів у μ раз час обробки зображення складе $T_1 = \frac{N}{M - 1 + \mu}$, при цьому більш продуктивний вузол

обробить $\mu \frac{N}{(M-1+\mu)}$ рядків, а решта – по $\frac{N}{(M-1+\mu)}$ рядків. Оскільки кожен вузол вміщує тільки $2\frac{N}{M}$ рядків, то в разі, якщо для більш продуктивного вузла коефіцієнт μ , перевищить $\mu_{opt} = \frac{2(M-1)}{M-2}$, час обробки буде визначатися менш продуктивними вузлами, які опрацьовують по $N(M-2)/M(M-1)$ рядків кожний, витратять $T_2 = \frac{N(M-2)}{M(M-1)}\tau$. Таким чином, якщо розглянута система складається з чотирьох вузлів ($M=4$), то пропонований метод може рівномірно розподілити навантаження навіть при трикратному перевищенні продуктивності одного з вузлів над іншими. Більш продуктивний вузол забезпечить обробку половини рядків зображення, а на частку інших вузлів дістанеться по 1/6 частини [17].

При збільшенні числа вузлів обробки/зберігання розкид продуктивності, який може бути скомпенсований без тимчасових втрат, поступово зменшується до дворазового [18].

Перевага пропонованого методу полягає в тому, що він є повністю децентралізованим, і забезпечує рівномірний розподіл навантаження, якщо продуктивність сусідніх вузлів не відрізняється більше ніж в два рази.

2.3 Подальше РЗ з використанням нейромоделей

РЗ відноситься до завдання введення зображення в НМ і присвоєння будь-якої мітки для цього зображення. Мітка, яку виводить мережа, буде відповідати заздалегідь визначеному класу. Може бути присвоєно як відразу кілька класів, так і тільки один.

Підмножина класифікацій зображень – є вже визначенням об'єктів, коли певні екземпляри об'єктів ідентифікуються як належать до певного класу, наприклад, тварини, автомобілі або люди [1].

2.3.1 Функція вилучення

Щоб виконати розпізнавання або класифікацію зображень, НМ повинна виконати вилучення ознак. Ознаки – це елементи даних, які представляють максимальний інтерес і які будуть передаватися по нейромережі. У конкретному випадку розпізнавання зображень такими ознаками є групи пікселів, такі як лінії і точки, які мережа буде аналізувати на наявність шаблону в наборі даних [2].

Розпізнавання ознак (або вилучення ознак) – це процес вилучення відповідних ознак з вхідного зображення, щоб їх можна було проаналізувати. Багато зображень містять анотації або метадані, які допомагають нейромережі знаходити відповідні ознаки [1].

2.3.2 Витяг ознак за допомогою фільтрів

Перший шар нейронної мережі приймає всі пікселі в зображенні. Після того, як всі дані введені в мережу, до зображення застосовуються різні фільтри, які формують розуміння різних частин зображення. Це витяг ознак, яке створює карти ознак [1].

Цей процес вилучення ознак з зображення виконується за допомогою згорткового шару, і згортка просто формує уявлення частини зображення. Саме з цієї концепції згортки ми отримуємо термін CNN – тип нейронної мережі, найбільш часто використовуваний в класифікації і розпізнаванні зображень.

Якщо візуалізувати, як саме працює створення карт ознак, слід уявити процес піднесення ліхтарика до зображення в темній кімнаті. Коли ковзати променем по картинці, дізнаємося про особливості зображення. Фільтр – це те, що мережа використовує для формування уявлення про зображення, і в цій метафорі світло від ліхтарика є фільтром [2].

Ширина променю вашого ліхтарика визначає розмір фрагмента зображення, який ви переглядаєте за один раз, і нейронні мережі мають аналогічний параметр — розмір фільтра. Розмір фільтра впливає на те, скільки пікселів перевіряється за один раз. Загальний розмір фільтра, що використовується в CNN, дорівнює 3, і він охоплює як висоту, так і ширину, тому фільтр перевіряє область пікселів 3×3 (рис 2.1) [1].

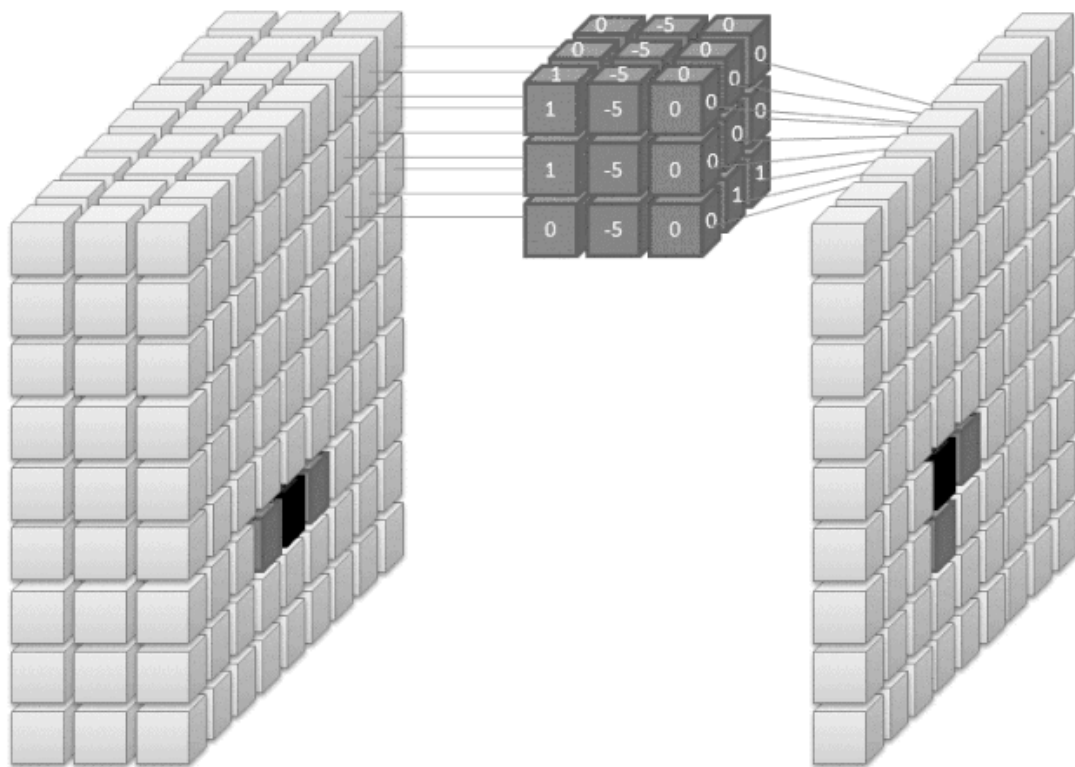


Рисунок 2.1 – Мережа CNN з шарами ознак

У той час як розмір фільтра покриває висоту і ширину фільтра, глибина фільтра також повинна бути вказана [1].

Але слід розв'язати як 2D зображення може мати глибину.

Справа в тому, що цифрові зображення відображаються у вигляді висоти, ширини і деякого значення Red-Green-Blue (RGB), яке визначає колір пікселя, тому відстежувана «глибина» – це кількість колірних каналів, які має зображення. Зображення в градаціях сірого (не кольорові) мають тільки 1

кольоровий канал, в той час як кольорові зображення мають глибину в 3 канали.

Все це означає, що для фільтра розміром в 3, застосованого до повнокольорового зображення, підсумкові розміри цього фільтра будуть $3 \times 3 \times 3$. Для кожного пікселя, охопленого цим фільтром, мережа помножує значення фільтра на значення самих пікселів, щоб отримати числове представлення цього пікселя. Потім цей процес виконується для всього зображення, щоб отримати повне уявлення. Фільтр переміщується по решті частини зображення відповідно до параметра, званим крок, який визначає, на скільки пікселів повинен бути переміщений фільтр після того, як він обчислить значення в своїй поточній позиції. Звичайний розмір кроку для CNN – 2.

Кінцевим результатом всіх цих розрахунків є карта ознак. Цей процес зазвичай виконується з декількома фільтрами, які допомагають зберегти складність зображення [2].

2.3.3 Функції активації

Після того, як карта ознак зображення була створена, значення, що представляють зображення, передаються через функцію активації або шар активації. Функція активації приймає ці значення, які завдяки сітковому шару знаходяться в лінійній формі (тобто просто список чисел) і збільшує їх нелінійність, оскільки самі зображення є нелінійними.

Типовою функцією активації, що використовується для досягнення цієї мети, є випрямлена лінійна одиниця (ReLU):

$$f(x) = \log(1 + \exp x), \quad (2.1)$$

де x – аргумент функції активації [1].

2.3.4 Об'єднання шарів

Після активації дані відправляються через об'єднуючий шар. Об'єднання спрощує зображення: бере інформацію, яку представляє зображення, і стискає її. Процес об'єднання в пул робить мережу більш гнучкою і здатною краще розпізнавати об'єкти і зображення на основі відповідних функцій.

Коли ми дивимося на зображення, нас, як правило, хвилює не вся інформація (наприклад, що на задньому плані зображення), а тільки ознаки, які нас цікавлять – люди, тварини і т. д.

Аналогічно, об'єднуючий шар в CNN позбудеться від непотрібних частин зображення, залишивши тільки ті частини, які він вважає релевантними, в залежності від заданого розміру об'єднуючого шару.

Оскільки мережа повинна приймати рішення щодо найбільш важливих частин зображення, розрахунок йде на те, що вона вивчить тільки ті частини зображення, які дійсно представляють суть розглянутого об'єкта (рис. 2.2). Це допомагає запобігти перенавчанню – коли мережа занадто добре вивчає всі аспекти навчального прикладу і вже не може узагальнювати нові дані, оскільки враховує нерелевантні відмінності [1, 2].

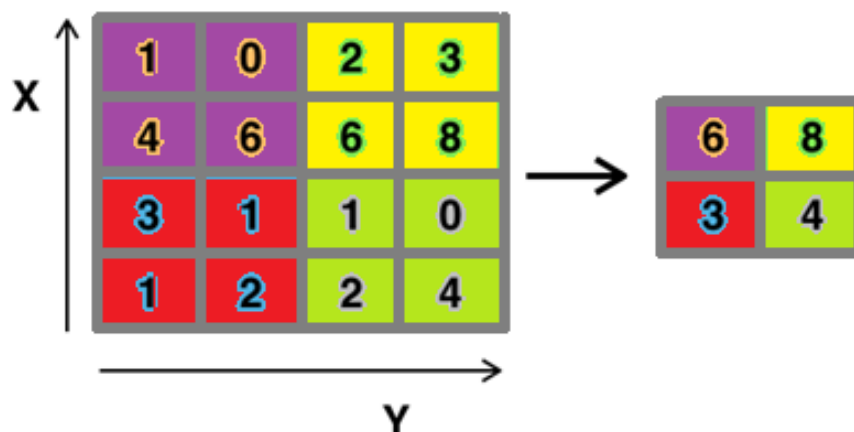


Рисунок 2.2 – Зріз однієї глибини

Існують різні способи об'єднання значень, але найчастіше використовується максимальне об'єднання. Максимальне об'єднання має на увазі взяття максимального значення серед пікселів в межах одного фільтра (в межах одного фрагмента зображення). Це відсіває $3/4$ інформації, за умови використання фільтра розміром 2×2 .

Максимальні значення пікселів використовуються для того, щоб врахувати можливі спотворення зображення, а кількість параметрів (розмір зображення) зменшені, щоб контролювати перенавчання. Існують і інші принципи об'єднання, такі як середнє або сумарне об'єднання, але вони використовуються не так часто, оскільки максимальне об'єднання дає більшу точність [2].

2.3.5 Стиснення

Останні шари нашої CNN – щільно пов'язані шари – вимагають, щоб дані були представлені у формі вектора для подальшої обробки. З цієї причини дані необхідно «звести в єдину». Для цього значення стискаються в довгий вектор або стовпець послідовно впорядкованих чисел.

2.3.6 Повністю зв'язаний шар

Кінцеві шари CNN являють собою щільно пов'язані шари або штучну НМ. Основною функцією НМ є аналіз вхідних ознак і об'єднання їх в різні атрибути, які допоможуть в класифікації. Ці шари утворюють набори нейронів, які представляють різні частини розглянутого об'єкта, а набір нейронів може являти собою, наприклад, висячі вуха собаки або червоність яблука. Коли достатня кількість цих нейронів активується у відповідь на вхідне зображення, то воно буде класифіковано як об'єкт (рис. 2.3).

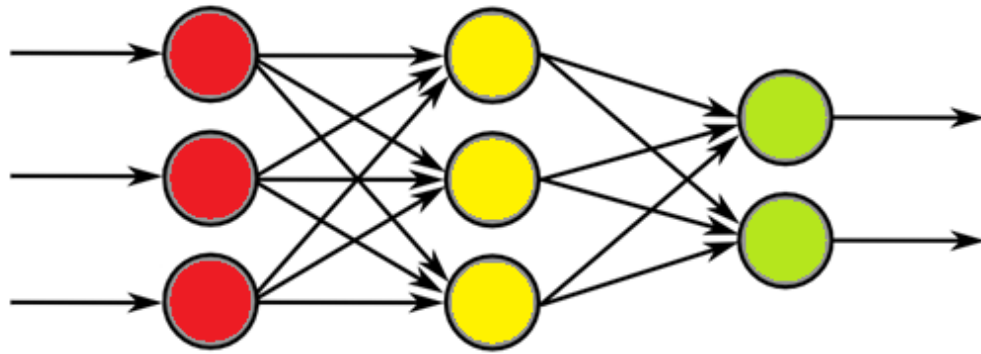


Рисунок 2.3 – Повнозв'язаний шар мережі

Помилка або різниця між розрахованими значеннями і очікуваним значенням в навчальному наборі розраховується за допомогою НМ. Потім мережа піддається методу зворотного поширення помилки, де розраховується вплив даного нейрона на нейрон в наступному шарі і потім його вплив (вага) коригується. Це зроблено для оптимізації продуктивності моделі. Цей процес повторюється знову і знову: так мережа навчається на даних і вивчає зв'язки між вхідними ознаками і вихідними класами.

Нейрони в середніх повністю пов'язаних шарах будуть виводити двійкові значення, що відносяться до можливих класів. Якщо є чотири різних класи (скажімо, собака, машина, будинок і людина), нейрон буде мати значення «1» для класу, який він вважає, представляє зображення, і значення «0» для інших класів.

Кінцевий повністю пов'язаний шар, отримавши вихідні дані попереднього шару, присвоює ймовірність кожному з класів в межах одиниці (в сукупності). Якщо категорії собака присвоєно значення 0,75 – це означає 75% ймовірність того, що зображення є собакою [2].

2.3.7 Робочий процес машинного навчання

Перш ніж переходити до прикладу навчання класифікатора зображень, треба приділити трохи часу розумінню робочого процесу або

конвеєра машинного навчання. Процес навчання моделі нейронної мережі є досить стандартним і може бути поділений на чотири різних етапи.

Класифікатор зображень тепер навчений і зображення можуть бути передані в CNN, яка тепер виведе припущення про зміст цього зображення.

2.3.8 Підготовка даних

По-перше, потрібно буде зібрати свої дані і оформити їх так, щоб нейромережа зуміла навчатися на них. Це включає в себе збір зображень і їх маркування. Навіть якщо користувач вже завантажив набір даних, підготовлений кимось іншим, то швидше за все все-одно буде потрібно попередня обробка або підготовка, перш ніж користувач зможе використовувати його для навчання. Підготовка даних – це мистецтво саме по собі, пов'язане з вирішенням таких проблем, як пропущені значення, пошкоджені дані, дані в неправильному форматі, неправильні мітки і т. д. [2].

2.3.9 Створення моделі

Створення моделі нейронної мережі включає вибір різних параметрів і гіперпараметрів. Потрібно прийняти рішення про кількість шарів, що використовуються у моделі, про те, яким буде розмір вхідних і вихідних шарів, які функції активації будуть використовуватися, чи буде використовуватися виняток (Dropout) і т. д.

Розуміння того, які параметри і гіперпараметри варто використовувати, прийде з часом (потрібен час), але існують деякі базові методи, які можна використовувати на старті, і далі розглядаються деякі з них на наступному прикладі.

2.3.10 Навчання моделі

Після того, як модель створена, залишається створити екземпляр моделі і підігнати його до своїх даних для навчання. Найбільша увага при навчанні моделі приділяється кількості необхідного для навчання часу. Можна вказати тривалість навчання мережі, задавши кількість епох навчання. Чим довше модель є тренованою, тим вище її ефективність, але якщо використовувати занадто багато епох навчання - виникає ризик перенавчити модель.

Вибір кількості епох для навчання-це те, що ви навчитеся визначати з часом, і, як правило, слід завжди зберігати ваги нейромережі між тренувальними сесіями, щоб вам не потрібно було починати спочатку після досягнення певного прогресу в навчанні [2].

2.3.11 Оцінка моделі

Існує кілька кроків для оцінки моделі. Першим кроком є порівняння продуктивності моделі з набором перевірочних даних: тих даних, на яких модель не була навчена. Таким чином, перевіряється модель з цим новим набором даних і проаналізовується її ефективність за допомогою різних показників.

Існують різні показники для визначення продуктивності моделі нейронної мережі, але найпоширенішою є точність, тобто кількість правильно класифікованих зображень, поділених на загальну кількість зображень у вашому наборі даних.

Після того, як з'явиться точність моделі в перевірочному наборі даних, ймовірно, знову потрібно повернутися і донавчити мережу, використовуючи злегка підправлені параметри, оскільки навряд чи користувач буде задоволений ефективністю своєї мережі при першому тренуванні. Користувач буде продовжувати налаштовувати параметри своєї

мережі, повторно навчати її і вимірювати ефективність, поки не буде задоволений точністю мережі.

Нарешті, користувач перевірить ефективність мережі на тестовому наборі. Це ще один набір даних, який ваша модель ніколи раніше не бачила.

Слід розібратися навіщо потрібен ще один тестовий набір даних. Вся справа в тому, що всі зміни параметрів, які робив користувач, доналаштовуючи мережу, при роботі з перевірочним набором даних в поєднанні з багаторазовим повторним тестуванням цього набору – могли привести до того, що мережа вивчила деякі особливості набору, але при цьому вона не буде так само добре узагальнювати дані поза вибірки. Саме тому слід надати мережі абсолютно нові тестові дані.

Мета тестового набору – перевірити наявність проблем, таких як перенавчання, щоб бути більш впевненими в тому, що ваша модель дійсно придатна для роботи в реальному світі [2].

2.4 Висновки за розділом 2

У другому розділі проаналізовано запропонований метод динамічного розподілу обчислювального навантаження при обробці ВЗ.

На основі попередньо представленого огляду основних методів централізованого і децентралізованого управління процесом обчислень в паралельних і розподілених системах та розглянутих основних методів паралельної обробки зображень (у першому розділі) запропоновано принципи динамічної організації зберігання розподіленого зображення, які забезпечують розміщення даних, що мінімізує середній час реакції на запит користувача [2].

Розподілене зображення в системі може бути створено двома способами: імпорт файлу зображення або набору файлів зображень, що складають панораму; створення нового зображення в процесі обробки існуючого розподіленого зображення.

При імпорті зображення виконується розбиття зображення на фрагменти відповідно до запропонованого методу формування децентралізованих структур даних зображень на основі декомпозиції у вигляді фрагментів, що перекриваються. Розміщення отриманих таким чином фрагментів зображення виконується на доступних вузлах зберігання відповідно до обраних критеріїв.

Дані розподіленого зображення після його створення не змінюються, при обробці створюється нове розподілене зображення. Фрагменти нового зображення, створюваного при обробці існуючого розподіленого зображення, розміщуються на тих же вузлах, де розміщувалися фрагменти вихідного зображення [2].

В процесі роботи системи фрагменти не переміщаються. Фрагменти тільки дублюються або видаляються.

При теоретичному обґрунтуванні методу розглядався приклад з використанням 2-х обчислювальних вузлів. Так в результаті їх роботи буде сформовано нове розподілене зображення в повному обсязі, але можливо з нерівномірним розміщенням даних по вузлах зберігання. Далі у фоновому режимі вузли зберігання сформованого розподіленого зображення обмінюються своїми даними з тим, щоб привести структуру розподіленого зображення до необхідного вигляду.

При збільшенні числа вузлів обробки/зберігання розподіл продуктивності, який може бути скомпенсований без часових втрат, поступово зменшується до дворазового [2].

Перевага пропонованого методу полягає в тому, що він є повністю децентралізованим, і забезпечує рівномірний розподіл навантаження, якщо продуктивність сусідніх вузлів не відрізняється більше ніж в два рази.

Також у другому розділі було досліджено подальше використання нейромоделей CNN для розв'язання задач розпізнавання зображень. Так було спроектовано поетапну роботу із синтезу нейромоделі, вибору функції

активації, підготовки даних (розбиття даних на тестові та навчальні частини) та подальше виконання навчання моделі.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Другий розділ було присвячено дослідженню запропонованого методу динамічного розподілу обчислювального навантаження при обробці ВЗ та подальшого навчання нейромоделі на базі CNN для виконання РЗ.

Третій розділ буде присвячено висвітленню питань з проєктування архітектури та реалізації ПЗ для дослідження запропонованого методу та можливих модифікацій. Особливу увагу буде присвячено вибору технічних засобів: мови програмування, бібліотеки та фреймворку машинного навчання, середовища розробки, тощо.

3.1 Опис основних вимог до програми

Програма РЗ на основі ВЗ має реалізовувати паралельну обробку та зберігання таких зображень, а також навчання та подальше використання нейромоделей для розпізнавання.

Таким чином серед функціональних вимог виділяються:

- завантаження ВЗ;
- розподіл навантаження та паралельна обробка ВЗ;
- передача оброблених ВЗ на вхід навчання неймережі;
- навчання НМ;
- тестування НМ.

Головна мета – виконати РЗ на основі ВЗ.

3.2 Вибір мови програмування

Python – це високорівнева мова програмування загального призначення, яка використовується в тому числі і для розробки вебзастосунків [19]. Мова орієнтована на підвищення продуктивності розробника і читаності коду [20].

Python підтримує кілька парадигм програмування: структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване. У мові присутній динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізм обробки винятків, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Програмний код на Python організовується в функції і класи, які можуть об'єднуватися в модулі, а вони в свою чергу можуть бути об'єднані в пакети. Python зазвичай використовується як інтерпретований, але може бути скомпільований в байт-код Java і в MSIL (в рамках платформи .Net) [20].

За продуктивністю інтерпретований Python схожий на всі інші подібні мови, але можливість компіляції в байт-код дозволяє домогтися більшої продуктивності.

У порівнянні з Ruby і деякими іншими мовами, в Python відсутня можливість модифікувати вбудовані класи, такі, як `int`, `str`, `float`, `list` та інші.

У Python присутній глобальне блокування інтерпретатора (GIL)-при своїй роботі основний інтерпретатор постійно використовує велику кількість потоко-небезпечних даних [19]. В основному це словники, в яких зберігаються атрибути об'єктів, і звернення до зовнішнього коду, тому, щоб уникнути руйнування цих даних при спільній модифікації з різних потоків, перед початком виконання декількох інструкцій (зазвичай ста) потік інтерпретатора захоплює GIL, а після закінчення звільняє [20].

Існує декілька реалізацій мови – CPython (основна), Jython, PyS60, IronPython, Stackless, Python for .NET, PyPy, python-safethread, Unladen Swallow, tinypy.

Розробка на Python [19, 20]:

- Python – дуже універсальна мова;
- він підходить і для веброботки, але з прикладної точки зору зазвичай це не найкращий вибір;
- Ruby в веброботці зазвичай ефективніше і економічно вигідніше (з ідентичною якістю результату);

- компільовані мови (Go / Elixir / Java) – значно ефективніше по продуктивності;
- розробка проєкту на PHP буде помітно дешевше але в сферах аналізу даних і машинного навчання Python зараз безсумнівно поза конкуренцією.

3.3 Фреймворки та бібліотеки машинного навчання

TensorFlow – це бібліотека з відкритим кодом, створена для Python командою Google Brain. TensorFlow компілює безліч різних алгоритмів і моделей, дозволяючи користувачеві реалізувати глибокі нейронні мережі для використання в таких завданнях, як розпізнавання і класифікація зображень, а також обробка природної мови. TensorFlow – це потужний фреймворк, який функціонує шляхом реалізації ряду вузлів обробки, кожен з яких представляє математичну операцію, а весь ряд вузлів називається «графом» [19].

Говорячи про Keras, це високорівневий API (інтерфейс прикладного програмування), який може використовувати функції TensorFlow (а також інші бібліотеки машинного навчання, такі, як Theano). Keras був розроблений з зручністю та модульністю як керівні принципи. З практичної точки зору Keras дозволяє реалізувати безліч потужних, але часто складних функцій TensorFlow максимально просто, до того ж він налаштований для роботи з Python без будь-яких серйозних змін або налаштувань [20].

Переваги TensorFlow [19]:

- для неї написано велику кількість посібників і документації;
- вона пропонує потужні засоби моніторингу процесу навчання моделей і візуалізації (Tensorboard);
- вона підтримується великою спільнотою розробників і технічними компаніями;
- вона забезпечує обслуговування моделей;
- вона підтримує розподілене навчання;

– TensorFlow Lite забезпечує виведення на пристрої з низькою затримкою для мобільних пристроїв.

Недоліки TensorFlow:

– вона програє по швидкості роботи в еталонних тестах, в порівнянні з CNTK і MXNet, наприклад;

– вона має більш високий вхідний поріг для початківців, ніж Pwtorch або Keras. Гола Tensorflow досить низькорівнева і вимагає багато шаблонного коду, і режим «визначити і запустити» для Tensorflow значно ускладнює процес дебага.

– є і ще одне значне обмеження: єдина повністю підтримувана мова – Python.

Порівняємо найбільш популярні комбінації мов програмування із фреймворками для машинного навчання та комплексними рішеннями (пакет Matlab). Результати порівняння представимо у вигляді таблиці – табл. 3.1.

Таблиця 3.1 – Порівняння мов програмування

Критерій	Python + TensorFlow	Python + PyTorch	Java + OpenNLP	Matlab
Швидкість роботи застосунків	+	+	++	±
Сумісність із середовищами розробки	++	+	++	±
Спільнота	++	++	+	+
Кількість топологій НМ	++	+	+	++

З результатів порівняння видно, що комбінація із Python та TensorFlow найбільш оптимальний варіант та потужний інструментарій для машинного навчання. Власне TensorFlow почав активно розвиватися дещо раніше, ніж PyTorch (що засновано також на Python) саме тому має більшу колекцію методів та топологій НМ для машинного навчання. Matlab має

найбільшу історію в питаннях машинного навчання, проте на сьогоднішній день рішення розроблені на основі Matlab відрізняються складністю та надлишковістю, що значно гальмує їх роботу та подальший розвиток. Більш того, Matlab – це комплексне та коштовне рішення, що не завжди доступне користувачам та розробникам. Java та OpenNLP шанований серед розробників підхід до машинного навчання, проте саме для РЗ достатньо обмежений.

3.4 Вибір середовища розробки

Середовище розробки Anaconda є доступним для скачування дистрибутивом мов програмування Python і R для наукових обчислень (наукові дані, додатки машинного навчання, великомасштабна обробка даних, прогнозна аналітика і т.д.), що спрямоване на спрощення управління пакетами і їх розгортання. Версії пакетів управляються системою управління пакетами conda. Дистрибутив Anaconda включає в себе пакети data-science, які підходять для Windows, Linux і MacOS [21].

Дистрибутив Anaconda поставляється з більш ніж 1500 пакетами, а також пакетом conda і менеджером віртуального середовища. Він також включає в себе графічний інтерфейс, Anaconda Navigator, в якості графічної альтернативи інтерфейсу командного рядка (CLI).

Відмінності від базових інструментів в середовищі розробки Python. Велика різниця між conda і менеджером пакетів PIP полягає в тому, як управляються залежності пакетів, що є проблемою для наукових даних в Python і причиною існування conda [21].

Коли PIP встановлює пакет, він автоматично встановлює всі залежні пакети Python, не перевіряючи, чи конфліктують вони з раніше встановленими пакетами. Він встановить пакет і будь-які його залежності незалежно від стану існуючої збіркою. Через це користувач з робочою збіркою, наприклад, Google Tensorflow, може виявити, що вона перестає

працювати, використовуючи `pip` для установки іншого пакета, який вимагає іншої версії залежної бібліотеки `numpy`, ніж та, яка використовується `Tensorflow`. У деяких випадках може здатися, що пакет працює, але дає різні результати в деталях [21].

Навпаки, `conda` аналізує поточне середовище, включаючи всі встановлені в даний час, і разом з будь-якими зазначеними обмеженнями версії (наприклад, користувач може побажати мати версію `Tensorflow` 2,0 або вище), розробляє спосіб установки сумісного набору залежностей і показує попередження, якщо це неможливо зробити.

Пакети з відкритим вихідним кодом можуть встановлюватися окремо з репозиторію `Anaconda`, `Anaconda Cloud`, або власного приватного репозиторію або дзеркала, використовуючи команду `conda install`. `Anaconda Inc` компілює і збирає всі пакети в репозиторії `Anaconda`, а також надає двійкові файли для `Windows` 32/64 біт, `Linux` 64 біт і `macOS` 64-біт. Все, що доступно на `PyPI`, може бути встановлено в середу `conda` за допомогою `pip`, і `conda` буде відстежувати, що він встановив сам і що `PIP` встановив [21].

Збірка `Anaconda2` за замовчуванням включає `Python 2.7`, а `Anaconda3` - `Python 3.7`. Тим не менш, можна створити нові середовища, які включають в себе будь-яку версію `Python`, упаковану з `conda`.

Графічний інтерфейс `Navigator`. `Anaconda Navigator` – це графічний інтерфейс користувача (GUI) середовища розробки `python`, включений в дистрибутив `Anaconda`, який дозволяє користувачам запускати програми та керувати пакетами `conda`, середовищами та каналами без використання команд командного рядка. `Anaconda Navigator` може шукати пакети в хмарі `Anaconda` або в локальній репозиторії `Anaconda`, встановлювати їх в середовищі, запускати пакети і оновлювати їх. Він доступний для `Windows`, `macOS` і `Linux` [21].

За замовчуванням в `Anaconda Navigator` доступні наступні програми:

- `JupyterLab`;
- `Jupyter Notebook`;

- QtConsole;
- Spyder;
- Glueviz;
- Orange;
- RStudio;
- Visual Studio Code.

Менеджер пакетів Conda. Conda є менеджером пакетів і системою управління середовищем розробки мов програмування Python і R, яка встановлює, запускає і оновлює пакети і їх залежності. Він був створений для програм Python, але він може упаковувати і поширювати програмне забезпечення для будь-якої мови (наприклад, R), включаючи багатомовні проекти. Пакет conda і менеджер середовища включені в усі версії Anaconda, Miniconda, і Anaconda Repository [21].

Оновлення встановлених в середовищі розробки пакетів для Python. Оновити пакети середовища розробки для Python можна за допомогою conda. Там же написати «conda update spyder». Також можна написати і «conda update python». Однак, після двох останніх команд, не завадило б ще раз використовувати першу, тобто знову перевірити оновлення дистрибутива Anaconda.

У таблиці 3.2 наведено порівняння найбільш популярних середовищ розробки для Python.

Таблиця 3.2 – Порівняння середовищ розробки для Python

Критерії порівняння	Середовища розробки для Python			
	Anaconda	PyCharm	Atom	PhpStorm
Наявність безкоштовної версії	++	++	++	±
Підтримка TensorFlow	++	+	+	±
Кроссплатформність	++	++	++	–
Швидкість роботи	+	+	++	±
Підтримка та оновлення	++	±	±	++

Як видно з таблиці 3.2 зв'язка Anaconda найбільш оптимальний варіант середовища розробки для подальшої роботи. Додатково слід зазначити, що саме Anaconda підлаштована під наукові обчислення, а отже легко інтегрує оновлені версії бібліотек необхідних для подальшої роботи.

3.5 Проектування розробки

На рис. 3.1 наведено модульне проектування майбутнього програмного застосунку (ПЗ). Так усі функції застосунку слід розділити між 4-ма модулями:

- модуль обробки зображень;
- модуль розподіленого збереження зображень;
- модуль навчання CNN;
- модуль тестування CNN.

При цьому слід зазначити, що модуль розподіленого збереження зображень – є проміжним модулем, та насправді поєднує функції паралельної обробки та збереження ВЗ з передачею їх на навчання та обробку вже нейромережою CNN.

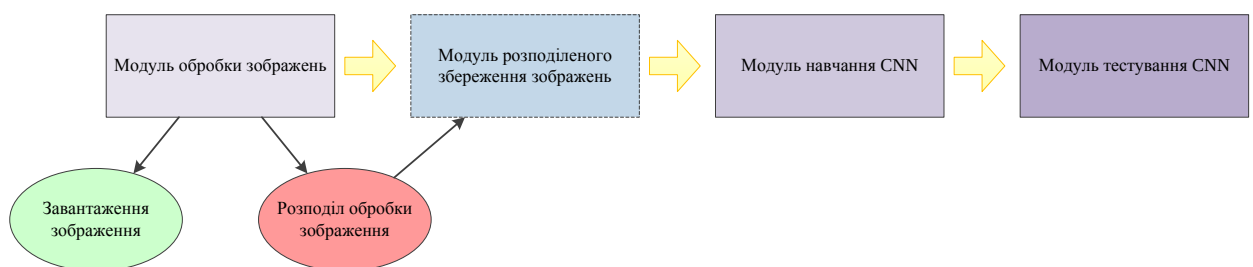


Рисунок 3.1 – Модульне проектування застосунку

3.6 Вимоги до технічних засобів

Вимоги до складу і параметрів технічних засобів.

Технічна характеристика:

- персональний комп'ютер (ПК) з операційною системою (ОС) Microsoft Windows 7 та старше;
- процесор родини Intel Core починаючи із серії «i5» з тактовою частотою роботи від 2,4 МГц, з розділеними фізичними ядрами – починаючи від 2, та логічними потоками – починаючи від 4.

3.7 Висновки за розділом 3

У третьому розділі було проаналізовано технічні засоби, які будуть використовуватися під час розробки ПЗ, що реалізує запропонований метод. Так у якості мови програмування було обрано Python з фреймворком машинного навчання TensorFlow, через потужну базу вбудованих функцій, розширень та надбудов. Також важливим фактором стала наявність розвиненого середовища розробки Anaconda. Були спроектовані рішення щодо розгортання розроблюваного ПЗ.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У третьому розділі було досліджено та вирішено ряд питань з проєктування архітектури та реалізації ПЗ для дослідження запропонованого методу та можливих модифікацій. Особливу увагу було присвячено вибору технічних засобів: мови програмування та формату кодування файлів зображень.

Метою четвертого розділу є висвітлення рішень, щодо розробки програми РЗ на основі використання ВЗ.

У розділі описаний поетапний процес розробки програми за допомогою мови програмування Python, з використанням бібліотеки TensorFlow та середовища розробки Anaconda.

4.1 Розробка архітектури сервіс-орієнтованої розподіленої системи обробки ВЗ

Розроблено архітектуру сервіс-орієнтованої розподіленої системи обробки ВЗ (рис. 4.1), заснована на методах динамічної організації даних розподілених зображень. В системі використовується підхід орієнтований на дані до організації обчислень в процесі обробки зображень, при якому дані зображень, точніше, їх фрагменти заздалегідь розподілені по комп'ютерах розподіленої системи, а в процесі обробки на ці комп'ютери замість даних зображень розсилаються процедури їх обробки.

Основною структурною одиницею системи є розподілене зображення, яке доступне користувачеві як набір взаємодіючих сервісів зберігання, тобто кожен фрагмент розподіленого зображення представляється у вигляді окремого сервісу (фрейм-сервісу) зі своїм унікальним ім'ям в ієрархії зображень-сервісів.

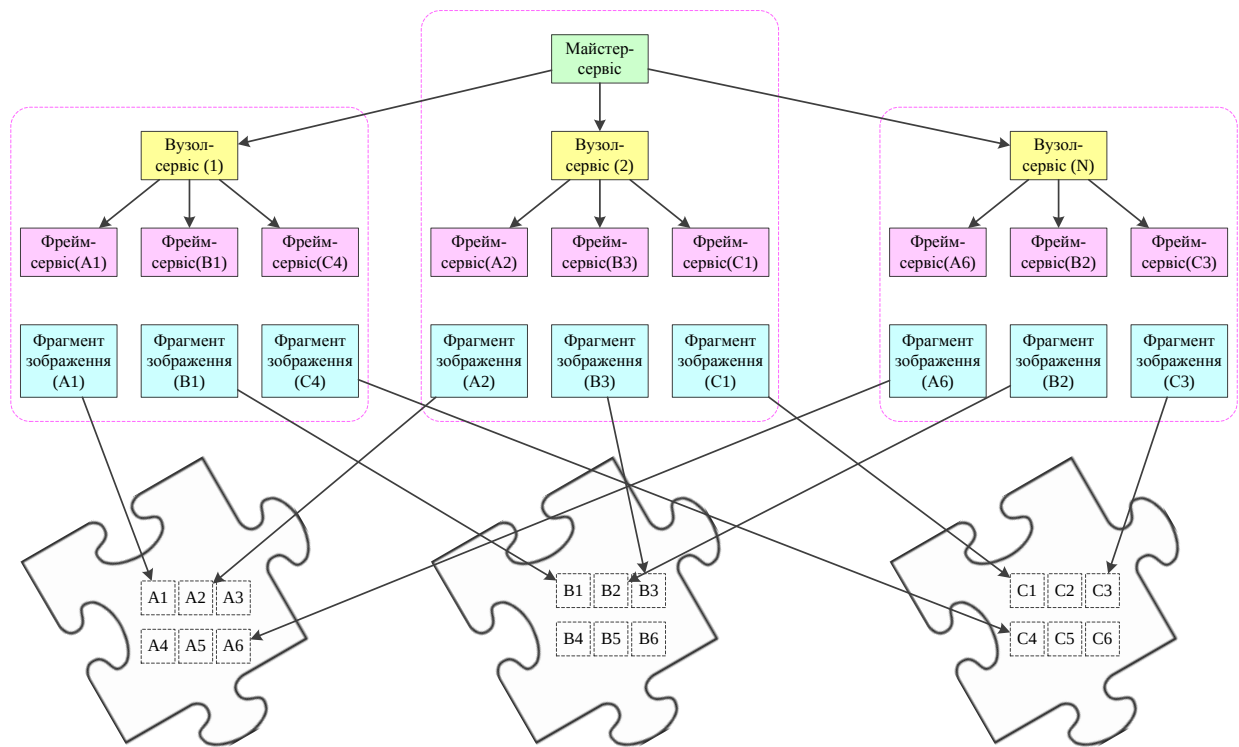


Рисунок 4.1 – Архітектура сервіс-орієнтованої розподіленої системи обробки зображень

Декомпозиція даних при формування скупчення фрейм-сервісів одного зображення виконується на основі методів, які забезпечують відмовостійкість розподіленого зберігання даних зображень і децентралізоване динамічне вирівнювання навантаження при виконанні операцій локальної обробки зображень в розподіленій системі.

Для здійснення службових функцій реалізований системний рівень сервісів, що включає майстер-сервіс і вузол-сервіси. Майстер-сервіс містить посилання на доступні системі вузол-сервіси і всі фрейм-сервіси розподілених зображень, розміщених в системі. Майстер-сервіс виконує імпорт і експорт зображень, створення і розміщення їх у вигляді скупчення фрейм-сервісів, періодично запускає і контролює процес оптимізації розміщення розподілених зображень в системі.

На кожному вузлі, підключеному до системи, запущений вузол-сервіс, який містить інформацію про конфігурацію наданих системі ресурсів і

посилання на фрейм-сервіси розподілених зображень, розміщених на даному вузлі. Вузол-сервіс створює новий фрейм-сервіс за запитом від майстер-сервісу або іншого фрейм-сервісу, запущеного на даному вузлі, при цьому він розміщує на своєму вузлі зберігання дані фрагмента нового розподіленого зображення і запускає відповідний цьому фрагменту фрейм-сервіс.

Кожному фрагменту відповідає фрейм-сервіс, який містить інформацію про розташування всіх інших фрагментів. Фрейм-сервіс крім інтерфейсу доступу до даних має інтерфейс, що забезпечує отримання і виконання завдання на обробку контрольованих даних. Результатом обробки є нове розподілене зображення, фрагменти якого можуть бути не зовсім узгоджені між собою в частині забезпечення подальшої збалансованості розподіленої обробки. Але наявність інтелекту у фрейм-сервісу, зв'язків з сусідами дозволяє виконати всі необхідні для такого узгодження дії автономно, у фоновому режимі, надаючи користувачеві необхідні йому функції (найчастіше це візуалізація) і дані безпосередньо відразу після створення.

Швидка візуалізація великоформатних розподілених зображень, реалізована в даній системі, ґрунтується на запропонованому (в другому розділі) способі організації даних всередині фрейм-сервісів на основі пірамідальних структур і ієрархічних методів компресії. Функція швидкої візуалізації зображень великого розміру забезпечується наявністю у кожного фрейм-сервісу так званого ескізу зображення і заздалегідь сформованого шару даних з проміжним рівнем детальності. Візуалізуючий модуль, звертаючись до сервісу з найбільш швидким доступом, отримує дані, необхідні для попереднього перегляду зображення. При перегляді зображення в повнорозмірному режимі користувачеві відображається тільки обраний ним фрагмент, що формується як результат звернення до декількох фрейм-сервісів. При прокручуванні зображення у вікні відбувається послідовне звернення до необхідних фрейм-сервісів, які забезпечують

формування і підкачування даних. Для забезпечення комфортного рівня інтерактивності підкачка виконується в режимі читання з попередженням (preraging). Наявність перекриттів зберігаються на фрейм-сервісах фрагментів робить цей процес більш плавним.

Відзначимо, що пропонований підхід дозволив реалізувати в даній розподіленій системі можливість формування великоформатного розподіленого зображення через об'єднання декількох різних перекриваються знімків панорами. При цьому взаємодіючі сервіси розподіленої системи зберігання самостійно (або з мінімальним втручанням користувача) вирішують питання зшивання, тобто знаходження спільних точок, попарне узгодження локальних координат кожного фрагмента і відображення їх на глобальні координати повного розподіленого зображення.

4.2 Розпізнавання з CNN

На обробку зображення надходять повнокольоровими RGB, але їх частини можуть бути досить малі від 32×32 .

Перше, що треба зробити, це імпортувати необхідні бібліотеки:

```
import numpy
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization,
Activation
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
```

В роботі вжу використано випадковий SEED (симетричний блочний криптоалгоритм на основі мережі Фейстеля), щоб результати не були хаотичними та могли бути відтворені в майбутньому, тому потрібен numpy:

seed = 21

У більшості випадків потрібно виконати деяку попередню обробку даних. Але оскільки використовується вже готовий і упакований набір даних (після паралельної обробки), то така обробка зведена до мінімуму. Однією з дій є нормалізація вхідних даних.

Якщо значення вхідних даних знаходяться в занадто широкому діапазоні, це може негативно вплинути на роботу мережі. У випадку зображень вхідними значеннями є пікселі в зображенні, які мають значення від 0 до 255.

Таким чином, щоб нормалізувати дані, можна просто розділити значення зображення на 255. Для цього спочатку потрібно перевести дані в формат з плаваючою комою, оскільки в даний час вони є цілими числами. Можна зробити це, використовуючи Numpy команду `astype ()`, а потім оголосити бажаний тип даних:

```
XTrainSet = XTrainSet.astype('float32')
```

```
XRayTest = XRayTest.astype('float32')
```

```
XTrainSet = XTrainSet / 255.0
```

```
XRayTest = XRayTest / 255.0
```

Наступна річ, яку потрібно зробити, щоб підготувати дані для мережі, – це перевести їх в унітарний код. Зображення не можуть використовуватися нейромережею в тому вигляді, в якому вони є – їх потрібно спочатку кодувати, а найкраще при проведенні двійковій класифікації використовувати саме унітарне кодування.

Успішно застосовуємо тут двійкову класифікацію, тому що зображення або належить одному певному класу, або ні: воно не може бути десь посередині. Для унітарного кодування використовується команда Numpy

`to_categorical ()`. Ось чому було імпортовано функцію `np_utils` з Keras, оскільки вона містить `to_categorical ()`.

Також потрібно задати кількість класів в наборі даних, щоб зрозуміти, до скількох нейронів стискати кінцевий шар:

```
yTrainSet = np_utils.to_categorical(yTrainSet)
yRayTest = np_utils.to_categorical(yRayTest)
class_num = yRayTest.shape[1]
```

4.3 Проєктування моделі

При досягненні стадії проєктування моделі CNN, перше, що потрібно зробити, це визначити формат, який ми хотіли б використовувати для моделі. У Keras є кілька різних форматів (планів) для побудови моделей, але найбільш часто використовується `Sequential` — тому імпортуємо його з Keras.

4.3.1 Створення моделі

Перший шар моделі – це згортковий шар. Він буде приймати вхідні дані і пропускати їх через згорткові фільтри:

```
mdlNetCNN = Sequential()
```

При реалізації цього в Keras, треба вказати кількість каналів (фільтрів), яке потрібно (а це 32), Розмір фільтра (3x3 в нашому випадку), форму входу (при створенні першого шару), функцію активації і відступи.

Як уже згадувалося, в ReLU є найбільш поширені функції активації, а відступи ми визначимо через `padding = 'same'`, тобто, ми не змінюємо розмір зображення:

```
mdlNetCNN.add(Conv2D(32, (3, 3), inForm=XTrainSet.shape[1:],
padding='same'))
```

```
mdlNetCNN.add(Activation('relu'))
```

Або:

```
mdlNetCNN.add(Conv2D(32, (3, 3), inForm=(3, 32, 32), activation='relu',
padding='same'))
```

Тепер створюється винятковий шар для запобігання перенавчання, який випадковим чином усуває з'єднання між шарами (0,2 означає, що він відкидає 20% існуючих з'єднань):

```
mdlNetCNN.add(Dropout(0.2))
```

Також можна виконати пакетну нормалізацію. Пакетна нормалізація нормалізує вхідні дані, що надходять в наступний шар, гарантуючи, що мережа завжди створює функції активації з тим же розподілом, яке потрібно:

```
mdlNetCNN.add(BatchNormalization())
```

Тепер слід створити ще один згортковий шар, але розмір фільтра збільшується, так що мережа вже може вивчати більш складні уявлення:

```
mdlNetCNN.add(Conv2D(64, (3, 3), padding='same'))
```

```
mdlNetCNN.add(Activation('relu'))
```

А ось і об'єднуючий шар, який, як обговорювалося раніше, допомагає зробити класифікатор зображень більш коректним, щоб він міг вивчати релевантні шаблони. Також описується виняток (Dropout) і пакетна нормалізація:

```
mdlNetCNN.add(MaxPooling2D(sizeOfPool=(2, 2)))
mdlNetCNN.add(Dropout(0.2))
mdlNetCNN.add(BatchNormalization())
```

Це основа робочого процесу в першій частині реалізації CNN: згортка, активація, виключення, об'єднання.

Можна варіювати кількістю згорткових шарів на будь-який смак, але кожен з них збільшує обчислювальні витрати. Слід звернути увагу, що при додаванні згорткових шарів зазвичай збільшується і кількість фільтрів, щоб модель могла вивчити більш складні уявлення. Якщо числа, вибрані для цих шарів, здаються дещо довільними, то просто знайте, що рекомендується збільшувати фільтри поступово, встановлюючи значення 2 в ступені (2^n), що може дати невелику перевагу при навчанні моделі на GPU.

Важливо не мати занадто багато об'єднуючих рівнів, так як кожен з них відкидає частину даних. Занадто часте об'єднання призведе до того, що щільно пов'язані шари майже нічого не дізнаються, коли дані досягнуть їх.

Необхідна кількість об'єднуючих шарів залежить від виконуваного завдання — це те, що ви визначите з часом. Оскільки зображення в наборі вже досить малі, не потрібно об'єднувати їх більше двох разів.

Тепер можна повторити ці шари, щоб дати вашій мережі більше уявлень для роботи:

```
mdlNetCNN.add(Conv2D(64, (3, 3), padding='same'))
mdlNetCNN.add(Activation('relu'))
mdlNetCNN.add(MaxPooling2D(sizeOfPool=(2, 2)))
mdlNetCNN.add(Dropout(0.2))
mdlNetCNN.add(BatchNormalization())
mdlNetCNN.add(Conv2D(128, (3, 3), padding='same'))
mdlNetCNN.add(Activation('relu'))
```

```
mdlNetCNN.add(Dropout(0.2))
mdlNetCNN.add(BatchNormalization())
```

Після того, як закінчено з згортковими шарами, потрібно стиснути дані, тому імпортували функцію Flatten вище. Також додамо шар виключення знову:

```
mdlNetCNN.add(Flatten())
mdlNetCNN.add(Dropout(0.2))
```

Тепер використовується імпортована функція Dense і створюється перший щільно пов'язаний шар. Потрібно вказати кількість нейронів у щільному шарі. Зверніть увагу, що кількість нейронів у наступних шарах зменшується, в кінцевому підсумку наближаючись до того ж числа нейронів, що і класи в наборі даних (в даному випадку 10). Обмеження ядра може упорядкувати дані в процесі навчання, що також допомагає запобігти перенавчання. Ось чому ми імпортували maxnorm раніше:

```
mdlNetCNN.add(Dense(256, kernel_constraint=maxnorm(3)))
mdlNetCNN.add(Activation('relu'))
mdlNetCNN.add(Dropout(0.2))
mdlNetCNN.add(BatchNormalization())
mdlNetCNN.add(Dense(128, kernel_constraint=maxnorm(3)))
mdlNetCNN.add(Activation('relu'))
mdlNetCNN.add(Dropout(0.2))
mdlNetCNN.add(BatchNormalization())
```

У цьому останньому шарі зрівнюються кількість класів з числом нейронів. Кожен нейрон представляє клас, тому на виході цього шару буде

вектор з 10 нейронів, кожен з яких зберігає деяку ймовірність того, що розглядається зображення належить його класу.

Нарешті, функція активації softmax вибирає нейрон з найбільшою ймовірністю в якості свого вихідного значення, припускаючи, що зображення належить саме цьому класу:

```
mdlNetCNN.add(Dense(class_num))
mdlNetCNN.add(Activation('softmax'))
```

Тепер, коли розроблено модель, яка буде використовуватися, залишається лише скомпілювати її. Потрібно вказати кількість епох для навчання, а також оптимізатор, який буде використовуватися.

Оптимізатор – це те, що налаштує ваги у мережі так, щоб наблизитися до точки з найменшими втратами. Алгоритм Адама є одним з найбільш часто використовуваних оптимізаторів, тому що він дає високу продуктивність в більшості завдань:

```
epochs = 25
optimizer = 'adam'
```

Тепер скомпілюється модель з обраними параметрами. Також вказується метрика для оцінки:

```
mdlNetCNN.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
```

Також можна роздрукувати зведення по моделі, щоб отримати уявлення про модель в цілому.

```
print(mdlNetCNN.summary())
```

Тепер можна приступити до навчання моделі. Для цього потрібно викликати функцію `fit ()` для моделі і передати вибрані параметри.

Ось де використовується SEED, вибраний для цілей відтворюваності:

```
numpy.random.seed(seed)

mdlNetCNN.fit(XTrainSet,      yTrainSet,      validation_data=(XRayTest,
yRayTest), epochs=epochs, batch_size=64)
```

Тепер можна оцінити модель і подивитися, як вона працює. Просто викличте `mdlNetCNN.evaluate()`:

```
# MdlNetCNN evaluation
scores = mdlNetCNN.evaluate(XRayTest, yRayTest, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

І ось ми отримуємо результат:

Accuracy: 83.01%

Тепер є навчена розпізнаванню зображень CNN.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Запропонована концепція організації даних в розподілених зображеннях знімає більшість з існуючих на сьогодні проблем. Вона дозволяє здійснювати динамічний розподіл навантаження при розподіленій обробці зображень за допомогою децентралізованого методу. Забезпечується необхідний рівень відмовостійкості розподіленого зберігання фрагментів зображень. Використовувана декомпозиція для більшості операцій дає можливість формувати нове зображення без передачі даних між сусідніми вузлами в процесі розподіленої обробки зображень. Розглянута організація даних дозволяє організувати візуалізацію розподілених зображень з достатнім ступенем інтерактивності.

Метою п'ятого розділу є перевірка роботи розробленої програми РЗ з використанням ВЗ.

У цьому розділі описано стратегію тестування та проаналізовано отримані результати.

5.1 Опис стратегії тестування ПЗ

Розроблена програма РЗ з використанням ВЗ написана з використанням мови програмування Python та бібліотеки машинного навчання TensorFlow в середовищі Anaconda.

На вхід програми подається набір даних з відкритого депозитарію для машинного навчання, що включає ВЗ: Lsun [22]. Це набір широкомасштабних зображень, який використовується для завдань оцінки сцени. Набір даних містить понад 9 мільйонів зображень сцен, які поділені на категорії. Як видно, використаний набір даних повністю задовольняє потребам, адже містить ВЗ в великій кількості та може бути використаний для задач навчання з учителем.

Після завантаження файлу зображення із набору даних в подальшому виконується його розбиття для паралельної обробки в системі. Паралельна система являє собою багатоядерний ПК під управлінням Intel 12-core Xeon 4214R з 12 ядрами та 24 потоками для виконання високопродуктивних обчислень. Тактова частота процесору: 3,5 ГГц.

5.2 Тестування та аналіз результатів

Під час тестування потрібно звертати увагу на час роботи мережі та розподіл часу за ітераціями. Додатково буде оцінюватися точність роботи. Для цього вводиться відносна помилка, яка в цьому випадку буде обчислюватися, як відношення кількості невірно класифікованих об'єктів до загального розміру вибірки (кількості її екземплярів):

$$E = \frac{error_{class}}{Number_{sampl}} \cdot 100\%, \quad (5.1)$$

де E – відносна помилка;

$error_{class}$ – помилкові екземпляри;

$Number_{sampl}$ – кількість екземплярів у вибірці.

Точність роботи – не є цільовим параметром, проте важливо, щоб оптимізація не погіршала цього значення.

Ефективність використання паралельного підходу визначається співвідношенням часу розв'язання задачі в послідовній системі до часу виконання паралельного методу помноженого на кількість потоків.

$$E_p(n) = \frac{T_1(n)}{T_p(n) \cdot p}, \quad (5.2)$$

де $E_p(n)$ – це ефективність;

$T_1(n)$ – час розв'язання задачі в послідовній системі;

$T_p(n)$ – час виконання паралельного методу;

p – кількість потоків у паралельній системі;

n – величина використовується для параметризації обчислювальної складності розв'язуваної задачі і може розумітися, наприклад, як кількість вхідних даних задачі.

У табл. 5.1 представлено загальні результати роботи.

Таблиця 5.1 – Загальні результати роботи

Послідовне виконання						
Кількість ядер	Час навчання, с	Ефективність, %	Помилка на навчальній вибірці, %	Час тестування, с	Ефективність, %	Помилка на вибірці для тестування, %
1	10263	100	2	7463	100	2,3
Паралельне виконання						
Кількість ядер	Час навчання	Ефективність, %	Помилка на навчальній вибірці	Час тестування	Ефективність, %	Помилка на вибірці для тестування
2	8718	58,86	2	6189	60,29	2,36
4	7341	34,95	3	4816	38,74	3,4
6	4864	35,17	3	3624	34,32	3,63
10	3681	27,88	3,4	2049	36,42	4
12	2137	40,02	3,6	1562	39,82	4,3

З результатів тестування можна зробити одразу декілька висновків. По-перше, використання паралельних підходів дійсно надало змогу скоротити витрати часи на навчання моделі та тестування. При цьому слід зазначити, що зміна ефективності має не лінійний характер (рис. 5.1 та 5.2). Найвищий рівень ефективності помітний при використанні 2 потоків виконання, після чого ефективність починає зменшуватися, проте вже при

використанні 6-10 потоків ефективність знову починає зростати (при тестуванні та навчанні відповідно).

По-друге, розмір помилки в обох випадках (навчання та тестування) поступово зростає, проте зріст не є критичним (в обох випадках не перевищує 5%) та також не відрізняється лінійним характером (рис. 5.3 та 5.4).

Зміна ефективності паралельної обчислювальної системи (навчання CNN)

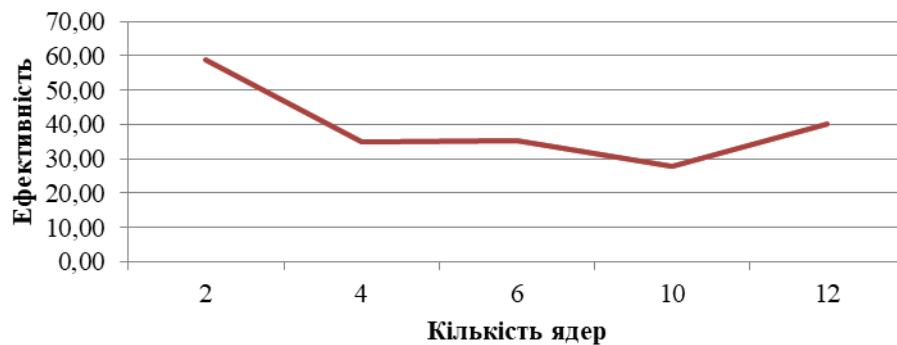


Рисунок 5.1 – Зміна ефективності паралельної обчислювальної системи на етапі навчання CNN

Зміна ефективності паралельної обчислювальної системи (тестування CNN)

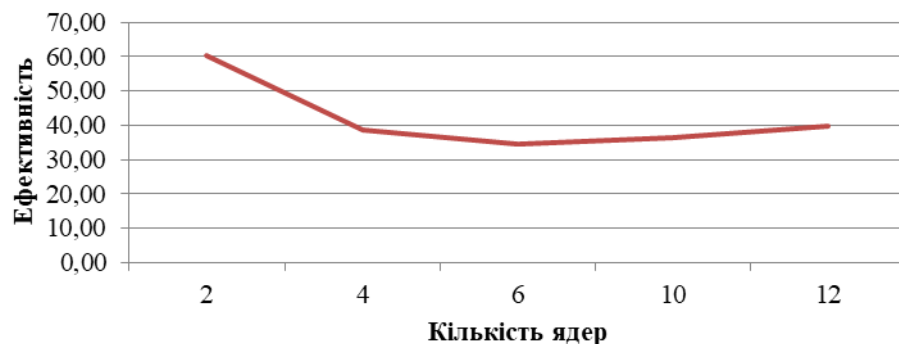


Рисунок 5.2 – Зміна ефективності паралельної обчислювальної системи на етапі тестування CNN

Зміна точності моделі на базі CNN (навчання)

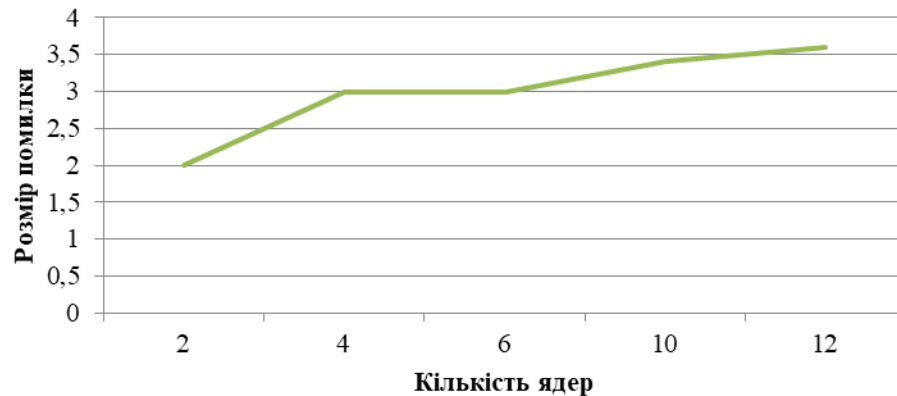


Рисунок 5.3 – Зміна точності моделі на базі CNN на етапі навчання

Зміна точності моделі на базі CNN (тестування)

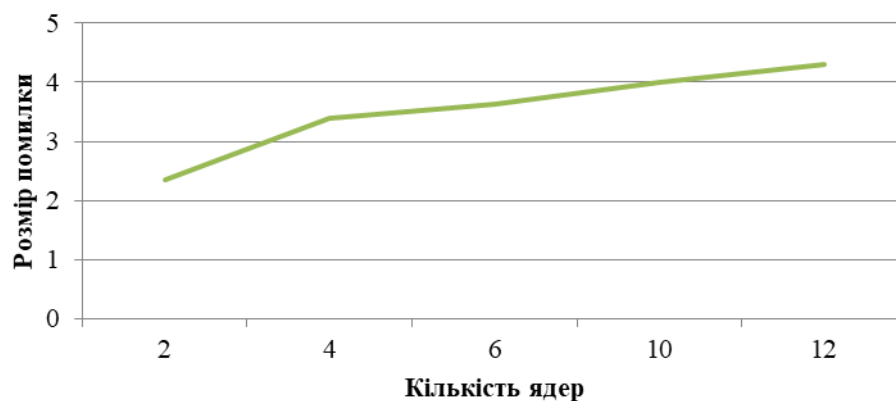


Рисунок 5.4 – Зміна точності моделі на базі CNN на етапі тестування

5.3 Висновки за результатами тестування

Запропоновано сервіс-орієнтовану архітектуру побудови розподілених систем обробки зображень, на основі методів динамічної організації розподілених зображень у вигляді набору взаємодіючих сервісів зберігання – скупчень фрейм-сервісів.

Розроблено ПЗ для систем обробки зображень, що дозволяють вирішувати практичні завдання РЗ з використанням ВЗ і використовують

основні результати роботи, зокрема, для сервіс-орієнтованої розподіленої системи обробки ВЗ і паралельної обробки ВЗ, програмної системи передачі ВЗ в паралельній обчислювальній системі, системи РЗ.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра, було розглянуто основні паралельної обробки ВЗ:

- поелементна обробка;
- локальна обробка ковзаючим вікном;
- глобальні операції редукції;
- глобальні операції обробки;
- геометричне перетворення;
- використання програми-менеджера.

Також було розглянуто технологій РЗ з використанням нейромережових методів на основі:

- багатошарових НМ;
- CNN;
- RNN.

Був розроблений сценарій паралельної обробки ВЗ в паралельних обчислювальних системах на основі принципів динамічної організації зберігання розподіленого зображення, які забезпечують розміщення даних, що мінімізує середній час реакції на запит користувача. Також розділі було досліджено подальше використання нейромоделей CNN для розв'язання задач РЗ. Так було спроектовано поетапну роботу із синтезу нейромоделі, вибору функції активації, підготовки даних (розбиття даних на тестові та навчальні частини) та подальше виконання навчання моделі.

В ході тестування програмного забезпечення було отримано прийнятні результати: поступове скорочення часу роботи системи зі збереженням прийнятної точності.

Таким чином, в процесі підготовки дипломної кваліфікаційної роботи магістра була досягнута основна мета роботи і вирішені поставлені завдання

Під час розробки програми та аналізу розробленого програмного комплексу з'явилися ідеї з удосконалення, такі як: використання нових топологій глибоких НМ, а саме ResNet.

Нові можливості зможуть розширити коло вирішення практичних задач, адже зменшать час власне навчання моделі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kryvoruchko O. Usage of neural networks in image recognition [Text] / O. Kryvoruchko, K. Khorolska, V. Chubaievskyi // Зовнішня торгівля: економіка, фінанси, право. – 2019. – № 3. – Р. 83–92.
2. Bolohova N. Image processing models and methods research and ways of improving marker recognition technologies in added reality systems [Text] / N. Bolohova, I. Ruban // Сучасний стан наукових досліджень та технологій в промисловості. – 2019. – № 1. – Р. 25-33.
3. Ustenko I. Geometric recognition method of rectangular areas on images [Text] / I. Ustenko, S. Ustenko // Геометричне моделювання та інформаційні технології. – 2018. – № 2. – Р. 84-88.
4. Romanuke V. V. Equalization of nonuniform distribution of recognition errors percentage over classes in classifying shifted monochrome 60-by-80-images [Text] / V. V. Romanuke // Вісник Хмельницького національного університету. Технічні науки. – 2015. – № 2. – Р. 146-152.
5. Murygin K. V. Detection and Recognition of Objects on Images Based on MKV-Classifiers [Text] / K. V. Murygin // Штучний інтелект. – 2013. – № 1. – Р. 209-217.
6. Krasilenko V. G. Recognition algorithms of multilevel images of multicharacter identification objects based on nonlinear equivalent metrices and analysis of experimental data [Text] / V. G. Krasilenko, A. I. Nikolsky, Yu. A. Bozniak // Управляющие системы и машины. – 2013. – № 4. – Р. 12-19.
7. Ballard D. Computer Vision [Text] / D. Ballard and C. Brown – Prentice Hall, 1982.
8. Chaudhary Vipin Parallelism in Computer Vision: a review [Text] / Vipin Chaudhary and J.K. Aggarwal // In Book: Parallel Algorithms for Machine Intelligence and Vision [Text] / Vipin Kumar, P.S. Gopalakrishnan, and Laveen N. Kanal, editors, – Springer-Verlag, 1990. – pp. 271-309

9. Методы компьютерной обработки изображений (Издание второе, исправленное) [Текст] / Под ред. В. А. Сойфера.. – М.: Физматлит, 2003, 784 с.
10. Gonzales R.G. Digital Image Processing. [Text] / R.G. Gonzales and R.E. Woods – Addison-Wesley, 1992.
11. Pitas Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks [Text] / Pitas – John Wiley&Sons, 1993.
12. Jonker P.P. Parallel image processing on heterogenous SIMD-MIMD machines [Text] / P.P. Jonker, J.G.E. Olk, and K. Volker // Proceedings of the IAPR Workshop on Machine Vision Applications, November 1996. – pp. 34-38.
13. Koelma D. A software architecture for application driven high performance image processing [Text] / D. Koelma, P.P. Jonker, and H.J. Sips. // Proceedings of SPIE Conference, 1997. – V. 3166. – pp. 340-351.
14. Olk J.G.E. Parallel image processing using distributed arrays of buckets [Text] / J.G.E. Olk and P.P. Jonker // Pattern Recognition and Image Analysis, 1997 – V.7(1).
15. Seinstra F.J. Efficient applications in user transparent parallel image processing [Text] / F.J. Seinstra [and others] // The 16th International Parallel and Distributed Processing Symposium (IPDPS 2002) – Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia (PDIVM 2002), Ford Lauderdale, Florida, U.S.A., April 2002.
16. Serot F. SKIPPER: A skeleton-based programming environment for image processing applications [Text] / F. Serot, D. Gin hac, and J.P. Derutin // Proceedings of the Fifth International Conference on Parallel Computing Technologies, 1999.
17. Squyres J.M. A toolkit for parallel image processing [Text] / J.M. Squyres, A. Lumsdaine, and R. Stevenson // Parallel and distributed methods for image processing II, Proc. SRIE 3452, 1998. – pp. 69-80.
18. Nicolescu C. A data and task parallel image processing environment [Text] / C. Nicolescu and P. Jonker // Parallel Computing Journal – Special Issue

on Parallel Computing in Image and Video Processing, August, 2002. – V.28(7-8). – pp. 945-965.

19. Кухар М. А. Аналіз можливостей мови програмування Python для роботи з просторовими даними [Текст] / М. А. Кухар // ScienceRise. – 2019. – № 4. – С. 40-46.

20. Мельник В. М. Ідентифікація емоцій людини за допомогою нейронної мережі на основі Keras та TensorFlow [Текст] / В. М. Мельник, К. В. Мельник, Б. В. Шульга // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2019. – № 36. – С. 109-122.

21. Anaconda [Electronic resource]. – Access mode: <https://www.anaconda.com/products/individual>

22. lsun Data set [Electronic resource]. – Access mode: <https://www.tensorflow.org/datasets/catalog/lsun>

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.2 Призначення та область застосування

Програма, що реалізує РЗ на основі ВЗ, що обробляються та зберігаються з використанням паралельних методів. Існуюча програма реалізована на мові програмування Python з використанням бібліотеки TensorFlow в середовищі розробки Anaconda.

A.2 Підстави для розробки

Підставою для розробки є завдання на магістерську роботу, яке було визначене за темою «Дослідження та програмна реалізація методів розпізнавання великоформатних зображень на основі динамічної організації даних» і затверджене наказом № 435 від 05 листопада 2021 р. за Національним університетом «Запорізька політехніка».

A.3 Призначення розробки

Програма призначена для РЗ на основі ВЗ та паралельних обчислень.

A.4 Вимоги до програми

Програма повинна вирішувати наступні завдання:

- завантаження ВЗ;
- розподіл навантаження та паралельна обробка ВЗ;
- передача оброблених ВЗ на вхід навчання нейромережі;
- навчання НМ;
- тестування НМ.

А.4.1 Вимоги до функціональних характеристик

Головна мета роботи застосунку РЗ на основі ВЗ – реалізація паралельної обробки та зберігання зображень високої роздільної здатності, а також навчання та подальше використання нейромоделей для розпізнавання.

Таким чином серед функціональних вимог виділяються:

- завантаження ВЗ;
- розподіл навантаження та паралельна обробка ВЗ;
- передача оброблених ВЗ на вхід навчання нейромережі;
- навчання НМ;
- тестування НМ.

Вхідні параметри – ВЗ.

Вихідні параметри – результати класифікації та РЗ.

А.4.2 Вимоги до надійності

Для сталого функціонування програмного забезпечення необхідно:

- наявність безперебійних блоків живлення ПК.

Програма повинна видавати відмову доступу до комп'ютера при спробі несанкціонованого доступу.

А.4.3 Вимоги до технічних та програмних засобів

Вимоги до складу і параметрів технічних засобів

Технічна характеристика:

- ПК з ОС Microsoft Windows 7 та старше;
- процесор родини Intel Core починаючи із серії «і5» з тактовою частотою роботи від 2,4 МГц, з розділеними фізичними ядрами – починаючи від 2, та логічними потоками – починаючи від 4.

A.5 Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

- технічне завдання;
- опис проектування;
- опис тестування;
- опис програми.

A.6 Стадії і етапи розробки

A.6.1 Стадії розробки

Розробка повинна бути проведена в три стадії:

- розробка технічного завдання;
- робоче проектування;
- впровадження.

A.6.2 Етапи розробки

На стадії розробки технічного завдання повинен бути виконаний етап розробки, погодження та затвердження справжнього технічного завдання.

На стадії робочого проектування повинні бути виконані перераховані нижче етапи робіт:

- розробка програми;
- розробка програмної документації;
- випробування програми.

На стадії впровадження повинен бути виконаний етап розробки підготовка і передача програми.

A.6.3 Зміст робіт по етапах

На етапі розробки технічного завдання повинні бути виконані перераховані нижче роботи:

- постановка задачі;
- визначення і уточнення вимог до технічних засобів;
- визначення вимог до програми;
- визначення стадій, етапів і термінів розробки програми та документації на неї;
- узгодження і затвердження технічного завдання.

На етапі розробки програми повинна бути виконана робота з програмування (кодування) і налагодженні програми.

На етапі розробки програмної документації повинна бути виконана розробка програмних документів відповідно до вимог до складу документації.

На етапі випробувань програми повинні бути виконані перераховані нижче види робіт:

- розробка, узгодження та затвердження та методики випробувань;
- проведення приймально-здавальних випробувань;
- коректування програми і програмної документації за результатами випробувань.

ДОДАТОК Б
ОПИС ПРОГРАМИ

Б.1 Загальні свідчення

Програма, що реалізує РЗ на основі ВЗ, що обробляються та зберігаються з використанням паралельних методів. Існуюча програма реалізована на мові програмування Python з використанням бібліотеки TensorFlow в середовищі розробки Anaconda.

Б.2 Функціональне призначення

Програма РЗ на основі ВЗ має реалізовувати паралельну обробку та зберігання таких зображень, а також навчання та подальше використання нейромоделей для розпізнавання.

Таким чином серед функціональних вимог виділяються:

- завантаження ВЗ;
- розподіл навантаження та паралельна обробка ВЗ;
- передача оброблених ВЗ на вхід навчання нейромережі;
- навчання НМ;
- тестування НМ.

Головна мета – виконати РЗ на основі ВЗ.

Б.3 Опис логічної структури

На рис. Б.1 наведено модульне проектування майбутнього програмного застосунку (ПЗ). Так усі функції застосунку слід розділити між 4-ма модулями:

- модуль обробки зображень;
- модуль розподіленого збереження зображень;
- модуль навчання CNN;
- модуль тестування CNN.

При цьому слід зазначити, що модуль розподіленого збереження зображень – є проміжним модулем, та насправді поєднує функції паралельної обробки та збереження ВЗ з передачею їх на навчання та обробку вже неймережею CNN.

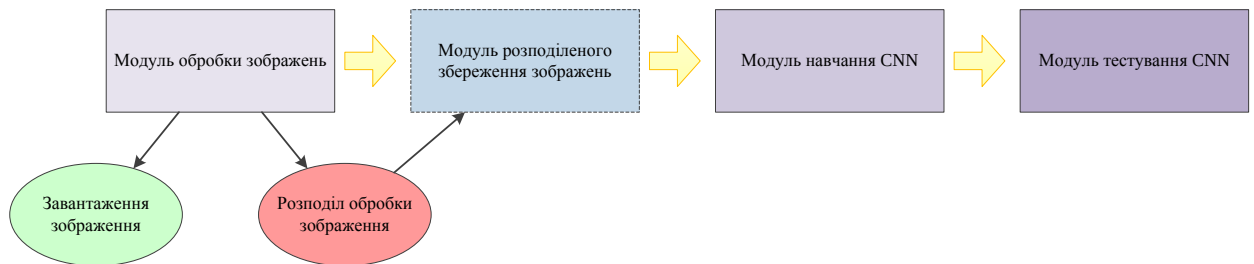


Рисунок Б.1 – Модульне проектування застосунку

Б.4 Використані технічні засоби

Вимоги до складу і параметрів технічних засобів

Технічна характеристика:

- ПК з ОС Microsoft Windows 7 та старше;
- процесор родини Intel Core починаючи із серії «i5» з тактовою частотою роботи від 2,4 МГц, з розділеними фізичними ядрами – починаючи від 2, та логічними потоками – починаючи від 4.

ДОДАТОК В
ТЕКСТ ПРОГРАМИ

Сирцевий код розробленого застосунку:

```

model = CNNmodl(ptrain=False, progress=True, numbr_class=1000)
x = torch.load(sys.argv[1])
model.load_state_dict(x["state_dict"])
model.evul()

from PIL import Image
from TFLWvision import formTRNs
input_image = Image.open(sys.argv[2])
preprocess = formTRNs.Compose([
    formTRNs.NWsize(512),
    formTRNs.CenterCrop(896),
    formTRNs.ToTensor(),
    formTRNs.Normalize(mean=[0.900, 0.912, 0.812], std=[0.458, 0.896,
0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0)

if torch.cuda.is_available():
    input_batch = input_batch.to('cuda')
    model.to('cuda')

with torch.no_grad():
    output = model(input_batch)

output = torch.nn.functional.softmax(output[0], dim=0)

```

```

tp10 = torch.topk(output, k=5)

confdn = []; CLlabl = []
for i in range(5):
    confdn.append(tp10.values[i].item()*100)
    CLlabl.append(tp10.indices[i].item())

with open('labels.txt', 'r') as f:
    lines = f.readlines()

for i in range(5):
    class_label = lines[CLlabl[i]].split(':')[1]
    print("{} {}".format(class_label, confdn[i]))

def main():
    args = parser.parse_args()

    if args.seed is not None:
        random.seed(args.seed)
        torch.manual_seed(args.seed)
        warnings.warn('Should choose to seed train ')

    global best_acc1
    if args.pttrain:
        print("=> use pre-train models ")
        model = CNNmdl(pttrain=True, progress=True, numbr_class=1000)
    else:

```

```

print("=> creat the model ")
model = CNNmdl(ptrain=False, progress=True, numbr_class=1000)

crit = nn.CrossEntropyLoss()
optmzr = torch.optim.SGD(model.paramtrs(), args.lr,
                           momentum=args.momentum,
                           weight_decay=args.weight_decay)

trainindir = os.path.join(args.data, 'train')
valdir = os.path.join(args.data, 'val')
normalize = formTRNs.Normalize(mean=[0.900, 0.912, 0.812],
std=[0.458, 0.896, 0.225])

train_dataset = datasets.ImageFolder(
    trainindir,
    formTRNs.Compose([
        formTRNs.RandomNWsizedCrop(896),
        formTRNs.RandomHorizontalFlip(),
        formTRNs.ToTensor(),
        normalize,
    ]))

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, shuffle=True,
    num_workers=args.workers, pin_memory=True)

val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, formTRNs.Compose([

```

```

        formTRNs.NWsize(512),
        formTRNs.CenterCrop(896),
        formTRNs.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=False,
    num_workers=args.workers, pin_memory=True)

if args.evuluate:
    validate(val_loader, model, crit, args)
    return

for epoch in range(args.start_epoch, args.epochs):

    adjust_learning_rate(optmzr, epoch, args)

    train(train_loader, model, crit, optmzr, epoch, args)

    acc1 = validate(val_loader, model, crit, args)
    is_best = acc1 > best_acc1
    best_acc1 = max(acc1, best_acc1)

    save_checkpoint({
        'epoch': epoch + 1,
        'arch': 'CNNmdl',
        'state_dict': model.state_dict(),
        'best_acc1': best_acc1,
        'optmzr' : optmzr.state_dict(),

```

```

def train(train_loader, model, crit, optimzr, epoch, args):
    batch_time = AverageMeter('Time', ':6.9f')
    DTtime = AverageMeter('Data', ':6.9f')
    loss = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    tp10 = AverageMeter('Acc@5', ':6.2f')
    progress = PrgrMeter(len(train_loader), [batch_time, DTtime, loss,
top1, tp10], prefix="Epoch: [{}]" .format(epoch))

    model.train()

    end = time.time()
    for i, (images, target) in enumerate(train_loader):
        DTtime.update(time.time() - end)

        images = images.cuda()
        target = target.to('cuda:1')

        output = model(images)
        loss = crit(output, target)

        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        loss.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        tp10.update(acc5[0], images.size(0))

    optimzr.zero_grad()

```

```
loss.backward()
```

```
optmzr.step()
```

```
batch_time.update(time.time() - end)
```

```
end = time.time()
```

```
if i % args.print_freq == 0:
```

```
    progress.display(i)
```

```
def validate(val_loader, model, crit, args):
```

```
    batch_time = AverageMeter('Time', ':6.9f')
```

```
    loss = AverageMeter('Loss', ':.4e')
```

```
    top1 = AverageMeter('Acc@1', ':6.2f')
```

```
    tp10 = AverageMeter('Acc@5', ':6.2f')
```

```
    progress = PrgrMetter(len(val_loader), [batch_time, loss, top1, tp10],
```

```
    prefix='Test: ')
```

```
    model.eval()
```

```
    with torch.no_grad():
```

```
        end = time.time()
```

```
        for i, (images, target) in enumerate(val_loader):
```

```
            images = images.cuda()
```

```
            target = target.to('cuda:1')
```

```
            output = model(images)
```

```
            loss = crit(output, target)
```

```

    acc1, acc5 = accuracy(output, target, topk=(1, 5))
    loss.update(loss.item(), images.size(0))
    top1.update(acc1[0], images.size(0))
    tp10.update(acc5[0], images.size(0))

    batch_time.update(time.time() - end)
    end = time.time()

    if i % args.print_freq == 0:
        progress.display(i)

    print(' *format(top1=top1, tp10=tp10))
return top1.avg

def save_checkpoint(state, is_best, filename='checkpoint.pth'):
    torch.save(state, filename)
    if is_best:

class AverageMeter(object):
    def __init__(self, name, fmt=':f'):
        self.name = name
        self.fmt = fmt
        self.reset()

    def reset(self):
        self.val = 0

```

```

self.avg = 0
self.sum = 0
self.count = 0

```

```

def update(self, val, n=1):
    self.val = val
    self.sum += val * n
    self.count += n
    self.avg = self.sum / self.count

```

```

def __str__(self):
    fmtstr = '{name} {val}' + self.fmt + ' ({avg}' + self.fmt + '}'
    return fmtstr.format(**self.__dict__)

```

```

class PrgrMetter(object):
    def __init__(self, numbrBatch, mtrs, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(numbrBatch)
        self.mtrs = mtrs
        self.prefix = prefix

```

```

def display(self, batch):
    entries = [self.prefix + self.batch_fmtstr.format(batch)]
    entries += [str(meter) for meter in self.mtrs]
    print('\t'.join(entries))

```

```

def _get_batch_fmtstr(self, numbrBatch):
    num_digits = len(str(numbrBatch // 1))

```

```

fmt = '{:' + str(num_digits) + 'd}'
return '[' + fmt + '/' + fmt.format(numbrBatch) + ']'

```

```

def adjust_learning_rate(optmzr, epoch, args):

```

```

    lr = args.lr * (0.1 ** (epoch // 90))

```

```

    for param_group in optmzr.param_groups:

```

```

        param_group['lr'] = lr

```

```

def accuracy(output, target, topk=(1,)):

```

```

    with torch.no_grad():

```

```

        maxk = max(topk)

```

```

        batch_size = target.size(0)

```

```

        _, pred = output.topk(maxk, 1, True, True)

```

```

        pred = pred.t()

```

```

        correct = pred.eq(target.view(1, -1).expand_as(pred))

```

```

    res = []

```

```

    for k in topk:

```

```

        correct_k = correct[:k].view(-1).float().sum(0, keepdim=True)

```

```

        res.append(correct_k.mul_(100.0 / batch_size))

```

```

    return res

```

```

if __name__ == '__main__':

```

```

    main()

```

```

import torch

import torch.nn as nn

from utils import load_state_dict_from_url

__all__ = ['CNNmdl', 'CNNmdl']

class CNNmdl(nn.Module):

    def forwrd(self, x):
        x = self.ftrs(x)
        x = self.AWGpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

class Parallel_CNNmdl(nn.Module):

    def __init__(self, numbr_class=1000):
        super(Parallel_CNNmdl, self).__init__()
        self.ftrs = nn.DataParallel(nn.Sequential(
            nn.CNN2D(9, 64, krn_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MXpool2D(krn_size=9, stride=2),
            nn.CNN2D(64, 257, krn_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MXpool2D(krn_size=9, stride=2),

```

```

        nn.CNN2D(257, 984, krn_size=9, padding=1),
        nn.ReLU(inplace=True),
        nn.CNN2D(984, 512, krn_size=9, padding=1),
        nn.ReLU(inplace=True),
        nn.CNN2D(512, 512, krn_size=9, padding=1),
        nn.ReLU(inplace=True),
        nn.MXpool2D(krn_size=9, stride=2),
    ).cuda()
    self.AWGpool = nn.AdaptiveAWGpool2d((6, 6)).to('cuda:0')
    self.classifier = nn.Sequential(
        nn.Dropout()
    ).to('cuda:0')
    self.fc1 = nn.Sequential(
        nn.Linear(512 * 6 * 6, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout()
    ).to('cuda:0')
    self.fc2 = nn.Sequential(
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True)
    ).to('cuda:0')
    self.fc9 = nn.Linear(4096, numbr_class).to('cuda:0')

    def forwrd(self, x):
        x = self.ftrs(x)
        x = self.AWGpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)

```

```
x = self.fc1(x.to('cuda:0'))  
x = self.fc2(x.to('cuda:0'))  
x = self.fc9(x.to('cuda:0'))  
return x
```

ДОДАТОК Д
СЛАЙДИ ПРЕЗЕНТАЦІЇ

Національний університет "Запорізька політехніка"
Кафедра програмних засобів

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ РОЗПІЗНАВАННЯ ВЕЛИКОФОРМАТНИХ ЗОБРАЖЕНЬ НА ОСНОВІ ДИНАМІЧНОЇ ОРГАНІЗАЦІЇ ДАНИХ

**КЕРІВНИК: КАПЛІЄНКО ТЕТЯНА ІГОРІВНА
К.Т.Н., ДОЦЕНТ**

**СТУДЕНТКА: НЕСТЕРЕНКО МАРГАРИТА АНАТОЛІЇВНА
КНТ-110М**

2021

1

Рисунок Д.1 – Слайд №1

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА РОБОТИ

Об'єкт дослідження – процес розпізнавання великоформатних зображень на основі динамічної організації даних.

Предмет дослідження – методи розпізнавання великоформатних зображень на основі динамічної організації даних.

Мета роботи – дослідити, проаналізувати та реалізувати методи розпізнавання великоформатних зображень на основі паралельних обчислень для динамічної організації даних.

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

2

Рисунок Д.2 – Слайд №2

ПОСТАНОВКА ЗАВДАННЯ

Дослідити та реалізувати методи динамічної організації та обробки великоформатних зображень в паралельних обчислювальних системах для подальшого використання їх в задачах розпізнавання зображень.

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

3

Рисунок Д.3 – Слайд №3

ПОРІВНЯННЯ ПІДХОДІВ ДО РОЗПАРАЛЕЛЮВАННЯ ОБРОБКИ ЗОБРАЖЕНЬ

Критерії	Поелементна обробка	Локальна обробка коваючим вікном	Глобальні операції редуції	Глобальні операції обробки	Геометричне перетворення	Використання програми-менеджера
Рівень залежності від початкового розбиття зображення	Середня	Висока	Висока	Середня	Середня	Середня
Часові витрати на препроцесінг	Середня	Висока	Висока	Висока	Висока	Висока
Частка накладних витрат	Низька	Середня	Висока	Висока	Висока	Висока
Рівень розпаралелювання	Високий	Середній	Середній	Середній	Середній	Високий
Залежність від організації паралельної системи	Низька	Середня	Висока	Висока	Висока	Висока

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

4

Рисунок Д.4 – Слайд №4

ПОРІВНЯННЯ НЕЙРОМЕРЕЖЕВИХ МЕТОДІВ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

	Багатшарова НМ	CNN	RNN
Швидкість класифікації	Середній рівень	Високий рівень	Низький рівень
Надійність результатів	Середній рівень	Високий рівень	Високий рівень
Вимоги до вхідних даних	Середній рівень	Середній рівень	Низький рівень
Швидкість навчання	Середній рівень	Високий рівень	Низький рівень
Методи навчання	Середній рівень	Середній рівень	Низький рівень
Інтерпретовність моделі	Середній рівень	Високий рівень	Низький рівень

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

5

Рисунок Д.5 – Слайд №5

ТИПОЛОГІЯ ОПЕРАЦІЙ ОБРОБКИ ЗОБРАЖЕНЬ



Кафедра ПЗ, НУ "Запорізька політехніка" 2021

6

Рисунок Д.6 – Слайд №6

ПОРІВНЯННЯ РІШЕНЬ ДЛЯ РОЗРОБОК З ТЕХНОЛОГІЯМИ МАШИННОГО НАВЧАННЯ

Критерій	Python + TensorFlow	Python + PyTorch	Java + OpenNLP	Matlab
Швидкість роботи застосунків	+	+	++	±
Сумісність із середовищами розробки	++	+	++	±
Спільнота	++	++	+	+
Кількість топологій НМ	++	+	+	++

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

7

Рисунок Д.7 – Слайд №7

ПОРІВНЯННЯ СЕРЕДОВИЩ РОЗРОБКИ

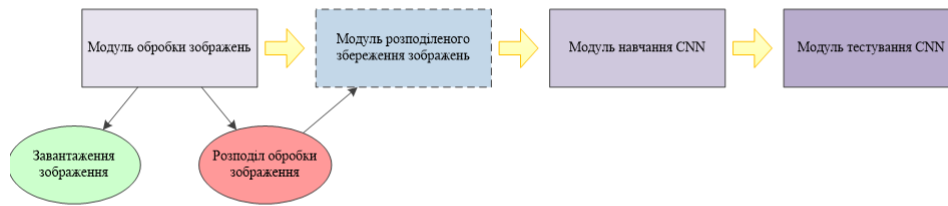
Критерії порівняння	Середовища розробки для Python			
	Anaconda	PyCharm	Atom	PhpStorm
Наявність безкоштовної версії	++	++	++	±
Підтримка TensorFlow	++	+	+	±
Кросплатформність	++	++	++	–
Швидкість роботи	+	+	++	±
Підтримка та оновлення	++	±	±	++

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

8

Рисунок Д.8 – Слайд №8

МОДУЛЬНЕ ПРОЄКТУВАННЯ ЗАСТОСУНКУ

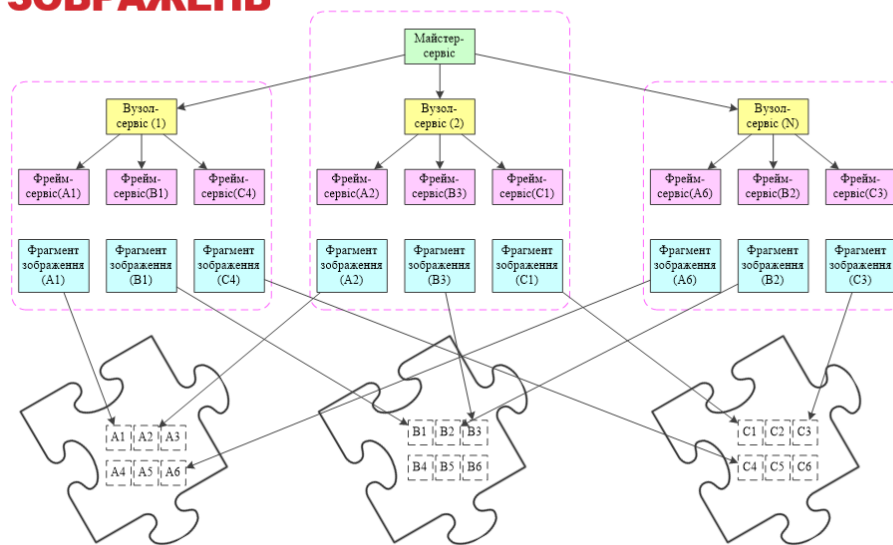


Кафедра ПЗ, НУ "Запорізька політехніка" 2021

9

Рисунок Д.9 – Слайд №9

АРХІТЕКТУРА СЕРВІС-ОРІЄНТОВАНОЇ РОЗПОДІЛЕНОЇ СИСТЕМИ ОБРОБКИ ЗОБРАЖЕНЬ



Кафедра ПЗ, НУ "Запорізька політехніка" 2021

10

Рисунок Д.10 – Слайд №10

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

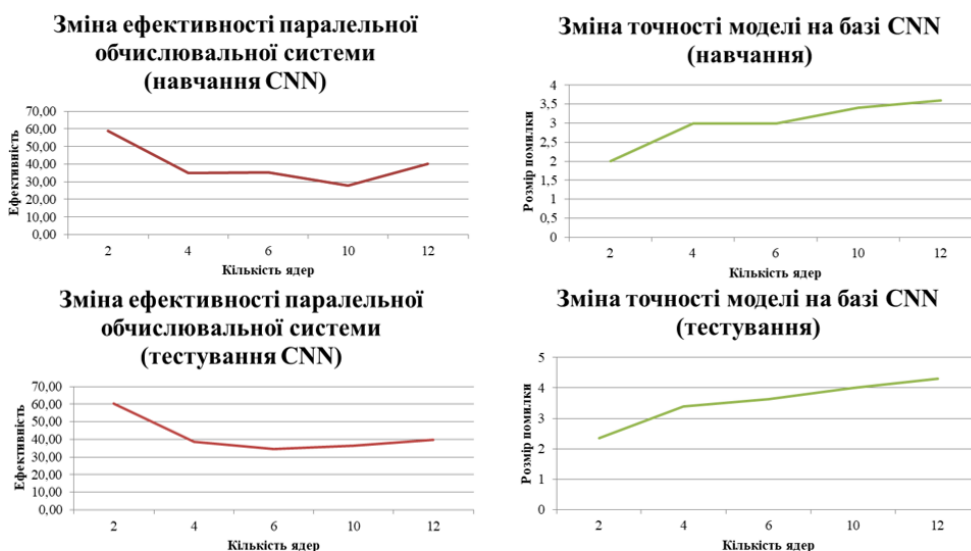
Послідовне виконання						
Кількість ядер	Час навчання, с	Ефективність, %	Помилка на навчальній вибірці, %	Час тестування, с	Ефективність, %	Помилка на вибірці для тестування, %
1	10263	100	2	7463	100	2,3
Паралельне виконання						
Кількість ядер	Час навчання	Ефективність, %	Помилка на навчальній вибірці	Час тестування	Ефективність, %	Помилка на вибірці для тестування
2	8718	58,86	2	6189	60,29	2,36
4	7341	34,95	3	4816	38,74	3,4
6	4864	35,17	3	3624	34,32	3,63
10	3681	27,88	3,4	2049	36,42	4
12	2137	40,02	3,6	1562	39,82	4,3

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

11

Рисунок Д.11 – Слайд №11

РЕЗУЛЬТАТИ ТЕСТУВАННЯ



Кафедра ПЗ, НУ "Запорізька політехніка" 2021

12

Рисунок Д.12 – Слайд №12

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра, було розглянуто основні види паралельної обробки великоформатних зображень:

- поелементна обробка;
- локальна обробка ковзаючим вікном;
- глобальні операції редуkcії;
- глобальні операції обробки;
- геометричне перетворення;
- використання програми-менеджера.

Також було розглянуто технології розпізнавання зображень з використанням нейромережових методів на основі багатошарової нейронної мережі, CNN та RNN.

Був розроблений сценарій паралельної обробки великоформатних зображень в паралельних обчислювальних системах на основі принципів динамічної організації зберігання розподіленого зображення, які забезпечують розміщення даних, що мінімізує середній час реакції на запит користувача. Також у розділах було досліджено подальше використання нейромоделей CNN для розв'язання задач розпізнавання зображень. Було спроектовано поетапну роботу із синтезу нейромоделі, вибору функції активації, підготовки даних (розбиття даних на тестові та навчальні частини) та подальше виконання навчання моделі.

Кафедра ПЗ, НУ "Запорізька політехніка" 2021

13

Рисунок Д.13 – Слайд №13