

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ
НАВЧАННЯ З ПІДКРІПЛЕННЯМ
RESEARCH AND SOFTWARE IMPLEMENTATION OF IMAGE
CLASSIFICATION METHODS BASED ON REINFORCEMENT LEARNING

Виконав: студент(ка) 2 курсу, групи КНТ-111м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Капленко А.А.

(прізвище та ініціали)

Керівник Коцур М.І.

(прізвище та ініціали)

Рецензент Гофман Є.О.

(прізвище та ініціали)

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

Інститут, факультет 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
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Капленка Артема Анатолійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів класифікації зображень на основі навчання з підкріпленням. Research and Software Implementation of Image Classification Methods Based on Reinforcement Learning

керівник проєкту (роботи) Коцур Михайло Ігорович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “04” листопада 2022 року № 363

2. Строк подання студентом проєкту (роботи) 12 грудня 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблемної області та постановка завдань дослідження. 2. Матеріали та методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 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
1-5 Основна частина	Коцур М.І., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “05” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	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	Розділи 3,4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-12 тижні	Додатки
8	Нормоконтроль та рецензування.	13 тиждень	
9	Захист роботи.	14 тиждень	

Студент(ка)

(підпис) Капленко А.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Коцур М.І.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
118 с., 7 табл., 28 рис., 4 дод., 35 джерел.

КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, МАШИННЕ НАВЧАННЯ,
НАВЧАННЯ З ПІДКРІПЛЕННЯМ, МОДУЛЯРНІ НЕЙРОННІ МЕРЕЖІ,
PYTHON, TENSORFLOW.

Об'єкт дослідження – процес використання методів навчання з підкріпленням для вирішення задач класифікації зображень.

Предмет дослідження – методи навчання з підкріпленням автоматизації класифікації зображень.

Мета роботи – дослідити, проаналізувати та реалізувати методи навчання з підкріпленням для побудови модулярних нейронних мереж у якості основи для моделей класифікації зображень.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, технології навчання з підкріпленням, мова програмування Python, фреймворк TensorFlow, середовище розробки Anaconda, персональний комп'ютер (в комплектації: процесор Intel Core i7-10700K, 8 Gb оперативної пам'яті, SSD на 500 Gb, під управлінням операційної системи Microsoft Windows 10).

Результати. Створено програмне забезпечення, що реалізує метод навчання з підкріпленням для побудови модулярної нейронної мережі для класифікації зображень.

Висновки. Розроблено програмний продукт для побудови та навчання модулярної нейронної мережі для автоматизації класифікації зображень.

Галузь використання – системи розпізнавання зображень для технічного, медичного та військового застосування.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 118 pages, 28 figures, 7 tables, 4 appendixes, 35 sources.

IMAGE CLASSIFICATION, MACHINE LEARNING, REINFORCEMENT LEARNING, MODULAR NEURAL NETWORKS, PYTHON, TENSORFLOW.

The object of research is the process of using reinforcement learning methods to solve image classification problems.

The subject of the research is training methods with reinforcement of image classification automation.

The aim of this paper is to investigate, analyze, and implement reinforcement learning methods for building medullary neural networks as the basis for image classification models.

Materials, methods and technical means: structural and Object-Oriented Programming, reinforcement learning technologies, Python programming language, TensorFlow framework, Anaconda development environment, Personal Computer (Complete Set: Intel Core i7-10700k processor, 8 Gb of RAM, 500 Gb SSD, running Microsoft Windows 10 operating system).

Results. Software has been developed that implements a reinforcement learning method for building a medullary neural network for image classification.

Conclusions. A software product has been developed for building and training a medullary neural network for automating image classification.

The field of use is image recognition systems for technical, medical and military applications.

ЗМІСТ

	С.
Вступ.....	9
1 Аналіз проблемної області та постановка завдань дослідження.....	11
1.1 Класифікація зображень	11
1.2 Порядок виконання класифікації зображень	14
1.3 Підходи машинного навчання для навчання моделей класифікації зображень	26
1.4 Постановка завдання	35
1.5 Висновки за розділом 1	35
2 Матеріали та методи	36
2.1 Модулярні нейронні мережі	36
2.2 Використання МНМ та навчання з підкріпленням для класифікації зображень	41
2.3 Класифікація зображень нейромоделями на базі МНМ з навчанням із підкріпленням	47
2.4 Висновки за розділом 2	49
3 Розробка архітектури програми	50
3.1 Опис основних вимог до програми.....	50
3.2 Вибір мови програмування.....	51
3.3 Вибір фреймворку	56
3.4 Вибір середовища розробки	63
3.5 Проектування структури програмної системи	66
3.6 Вимоги до технічних засобів.....	67
3.7 Висновки за розділом 3	67
4 Основні рішення щодо реалізації компонентів системи	68
4.1 Представлення основних рішень розробки.....	68
4.2 Висновки за розділом 4.....	73
5 Експлуатація, тестування та експериментальне дослідження програми	74
5.1 Вхідні дані для тестування	74

5.2 Метрики оцінювання.....	76
5.3 Результати тестування.....	78
5.4 Аналіз результатів	79
Висновки	81
Перелік джерел посилання	82
Додаток А Технічне завдання	86
Додаток Б Опис програми	91
Додаток В Текст програми	95
Додаток Д Слайди презентації.....	110

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BCE	– binary cross-entropy;
CLIP	– contrastive language-image pretraining;
CNN	– convolutional neural network;
GELU	– gaussian error linear unit;
GPU	– graphics processing unit;
ILSVRC	– ImageNet large-scale visual recognition challenge;
LSTM	– long-short term memory;
MLP	– multi-layer perceptrons;
MNIST	– modified national institute of standards and technology;
MSE	– mean squared error;
MSP	– multi-head attention network;
ReLU	– випрямлені лінійні одиниці;
ResNet	– residual network;
SVHN	– the street view house numbers;
ViT	– vision transformer;
БД	– база даних;
МНМ	– модулярна нейронна мережа;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер.

ВСТУП

Сьогодні, зі зростаючою мінливістю, необхідністю та застосуванням штучного інтелекту, такі сфери, як машинне навчання та його підмножини, глибоке навчання та нейронні мережі, набрали величезний імпульс.

Навчання вимагає програмного забезпечення та інструментів, таких як класифікатори, які обробляють величезну кількість даних, аналізують їх та витягують корисні функції.

Мета процесу класифікації полягає в тому, щоб розділити всі пікселі в цифровому зображенні на один з декількох класів. Зазвичай для виконання класифікації використовуються багатоспектральні дані, і справді спектральна структура, присутня в даних для кожного пікселя, використовується як числова основа для категоризації.

Мета класифікації зображень полягає в тому, щоб ідентифікувати та зобразити як унікальний рівень сірого (або кольору) особливості, що зустрічаються на зображенні, з точки зору об'єкта, який ці особливості насправді представляють на місцевості.

Класифікація зображень є, мабуть, найважливішою частиною аналізу цифрових зображень. Класифікація об'єктів є складним завданням, і тому класифікація зображень була важливим завданням в області комп'ютерного зору.

Класифікація зображень відноситься до віднесення зображень до одного з ряду заздалегідь визначених класів. Потенційно існує N класів, до яких може бути віднесено дане зображення.

Ручна перевірка та класифікація зображень може бути виснажливим завданням, особливо коли їх величезна кількість, і тому було б дуже корисно, якби ми могли автоматизувати весь цей процес за допомогою комп'ютерного зору.

Досягнення автономного водіння також є прекрасним прикладом використання класифікації зображень у реальному світі. Додатки включають

автоматизовану організацію зображень, веб-сайти зі стоковими фотографіями та відео, візуальний пошук для покращення можливості виявлення продукту, великі візуальні бази даних, розпізнавання зображень та облич у соціальних мережах тощо; саме тому потрібні класифікатори для досягнення максимально можливої точності.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

Перший розділ буде присвячено детальному аналізу проблемної області пов'язаної із класифікацією зображень і машинним навчанням та сучасних проблем та їх існуючих рішень. Під час роботи буде проведено дослідження основних термінів таких, як класифікація зображень, розпізнавання зображень, машинне навчання та навчання з підкріпленням.

1.1 Класифікація зображень

1.1.1 Визначення проблемної області класифікації зображень.

Сучасний стан області

Технології штучного інтелекту, такі як машинне навчання, глибоке навчання та комп'ютерний зір, можуть допомогти використовувати автоматизацію для структурування та організації майже усіх даних [1].

Очікується, що приблизно на момент 2025 року, людство буде колективно генерувати, записувати, копіювати та обробляти близько 175 зеттабайт даних. Це велика кількість даних, більш того, немає ознак уповільнення створення даних.

Зображення, включаючи картинки і відео – складають основну частину загальносвітових даних [2]. Для інтерпретації та систематизації цих даних розробники сучасних обчислювальних систем та програмного забезпечення (ПЗ) звертаються до класифікації зображень на основі штучного інтелекту [2].

Класифікація зображень аналізує цифрові зображення (фотографії, малюнки, скан-копії і т.п.) за допомогою аналітичних моделей на основі штучного інтелекту, які можуть ідентифікувати та розпізнавати широкий спектр критеріїв – від змісту зображення до часу доби [1]-[2].

Класифікація зображень (іноді, поняття класифікації зображень та розпізнавання зображень поєднують) – це завдання (порівняння) зіставлення певних ознак з даним зображенням [1].

Загалом класифікація зображень може бути:

- за однією ознакою;
- за двома та більше ознаками.

Класифікація зображень – це завдання класифікації та присвоєння міток групам зображень або векторів всередині зображення на основі певних критеріїв [3]. Мітка може бути призначена на основі одного або декількох критеріїв.

Класифікація зображень – це серйозне ресурсоємне завдання для порівняння сучасних архітектур і методологій в області комп'ютерного зору [1], [3].

У класифікації за однією ознакою кожне зображення має лише одну мітку або анотацію, як впливає з назви. Як результат, для кожного зображення, яке бачить модель, вона аналізує та класифікує його лише за одним критерієм [4].

Наприклад, можна запрограмувати модель штучного інтелекту для класифікації зображень на основі того, зображують вони денні чи нічні сцени [1]-[2].

З іншого боку, при класифікації за декількома ознаками зображення можуть мати кілька міток, причому деякі зображення містять усі мітки, які ви використовуєте одночасно [3].

Прикладом класифікації за кількома мітками є класифікація постерів фільмів, де фільм може бути частиною більш ніж одного жанру [1].

Слід розглянути більш детально ці види класифікації.

1.1.2 Класифікація зображень за однією ознакою

Класифікація за однією ознакою є найпоширенішим завданням класифікації при класифікації зображень з учителем [5].

Як випливає з назви, для кожного зображення в класифікації за однією ознакою присутня одна ознака (мітка або анотація). Таким чином, модель виводить одне значення або прогноз для кожного зображення, яке вона бачить [5].

Результатом роботи моделі є вектор з довжиною, що дорівнює кількості класів, і значенням, що позначає оцінку приналежності зображення до цього класу [5].

Функція активації Softmax використовується для того, щоб переконатися, що сума балів дорівнює одиниці, і максимальна кількість балів береться для формування вихідних даних моделі [6], формула (1.1):

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \quad (1.1)$$

де z – вектор дійсних значень;

i та j – лічильники для перебору елементів вектору та позначення їх кількості.

Хоча спочатку здається, що Softmax не надає ніякого значення прогнозу, оскільки максимально ймовірний клас не змінюється після його застосування, він допомагає прив'язати вихідні дані між одиницею і нулем, допомагаючи оцінити достовірність моделі за шкалою Softmax [6].

Деякі приклади наборів даних класифікації з однією міткою включають Modified National Institute of Standards and Technology (MNIST), The Street View House Numbers (SVHN), ImageNet та інші [1]-[4].

Класифікація з однією міткою може бути багатокласовою класифікацією, де існує більше двох класів, або двійковою класифікацією, де кількість класів обмежена лише двома.

1.1.3 Класифікація зображень за двома та більше ознаками

Класифікація за двома та більше ознаками – це завдання класифікації, при якому кожне зображення може містити більше одну ознаку, а деякі зображення можуть містити всі ознаки одночасно [7].

Хоча в деякому відношенні це здається схожим на класифікацію за однією ознакою, постановка проблеми є більш складною в порівнянні з класифікацією за однією ознакою [8].

Завдання класифікації за декількома ознаками широко поширені в області медичного діагностування та візуалізації, де у пацієнта може бути більше одного захворювання, яке можна діагностувати за візуальними даними у вигляді рентгенівських променів [7]-[8].

Крім того, в природному середовищі задача маркування зображень також може бути оформлена як задача класифікації з декількома мітками, що вказує на об'єкти, присутні на зображеннях [8].

1.2 Порядок виконання класифікації зображень

Розглянемо класифікацію зображень за етапами.

Весь процес починається з попередньої обробки – підготовки даних.

Цей крок покращує дані зображення, усуваючи небажані деформації та покращуючи певні ключові аспекти зображення, щоб моделі комп'ютерного зору могли працювати з цими вдосконаленими даними. По суті, відбувається очищення даних з їх підготовкою до обробки моделлю штучного інтелекту [7]-[8].

Очищення даних, яке іноді називають препроцесингом, є важливим кроком у підготовці даних для навчання вашої моделі, оскільки неточності даних призводять до неточностей у моделі класифікації зображень [1]-[4]. У якості результатів очищення даних можна помітити, що відбулося:

- видалення дублікатів: дублювання даних уповільнює процес навчання і може призвести до того, що модель додасть більшої ваги дубльованим даним без необхідності [7]-[8];

- видалення нерелевантних даних: включення нерелевантних даних не допоможе навчити модель досягти бажаної мети [7]-[8];

- фільтрування небажаних зачумлених даних (інформаційних викидів): деякі дані хоча і технічно релевантні, не допомагають при навчанні моделі штучного інтелекту [7]-[8]. Дані, які сильно виходять за рамки норми, можуть спотворити прогнози вашої моделі, тому краще просто видалити їх [3];

- виявлення відсутніх даних: відсутні дані можуть викликати проблеми в процесі навчання – в процесі очищення даних відсутні дані можуть бути ідентифіковані і відповідним чином оновлені [7]-[8];

- виправлення структурних помилок: більшість методів машинного навчання не здатні ідентифікувати помилки так, як це зробила б людина, а це означає, що кожна частина даних повинна бути точно організована [7], [8].

Добре організовані дані гарантують більшу вірогідність успіху, коли справа доходить до навчання моделі класифікації зображень – або будь-якої моделі штучного інтелекту [7]-[8].

Наприклад, скажімо, є набір модних образів. Задача полягає в тому, щоб переконатися, що всі зображення мають високу якість, добре освітлені та не мають дублікатів. Етап попередньої обробки – це коли переконуємося, що весь контент релевантний, а продукти чітко видно [7]-[8].

Виявлення об'єктів: визначення місця розташування об'єктів у наборі зображень. Це процес визначення місця розташування об'єкта, який включає сегментацію зображення та визначення місця розташування об'єкта [4].

Використовуючи попередній приклад моди, алгоритм міг би знайти спідниці, блузки, штани і т.д. у цьому випадку модель може бути навчена розпізнавати блузки у верхній частині зображення і спідниці в нижній частині [3].

Розпізнавання об'єктів і навчання: маркування розташованих зображень.

Методи глибокого навчання виявляють закономірності на зображенні та характеристики, які можуть бути унікальними для певної ознаки.

Модель навчається на цьому наборі даних і в майбутньому стає більш точною [7]-[8].

Після того, як були позначені дані, потрібно навчити свою модель штучного інтелекту [7]-[8]. Це передбачає завантаження великої кількості даних на кожну з ознак, щоб дати моделі штучного інтелекту можливість чогось навчитися. Чим більше навчальних даних завантажуються, тим точніше модель визначатиме вміст кожного зображення [6].

Класифікація об'єктів: модель готова класифікувати ваші зображення. Це заключний крок у процесі-розроблення моделі штучного інтелекту, яка класифікує зображення за кількома різними критеріями [7]-[8].

Метод використовує відповідний класифікаційний підхід для класифікації спостережуваних елементів за заздалегідь визначеними класами [1].

Це робиться шляхом порівняння шаблонів зображень із бажаними шаблонами.

Тепер елементи, які додавались як теги на попередньому кроці, будуть розпізнані алгоритмом на реальних зображеннях [7]-[8].

Підключення до робочого процесу штучного інтелекту. Після завершення цього процесу тепер ви можна підключити свою модель штучного інтелекту, що класифікує зображення, до робочого процесу штучного інтелекту. Це визначає вхідні дані – звідки надходять нові дані, і вихідні дані – що відбувається після того, як дані були класифіковані.

Наприклад, дані можуть надходити з нових запасів, а результатом може бути додавання даних до таблиці Google [7]-[8].

1.2.1 Моделі для класифікації зображень

У сучасному комп'ютерному зорі архітектура моделі найчастіше у якості найбільш оптимального стандарту використовується – згорткова нейронна мережа (Convolutional neural network – CNN) [9].

Операція згортки вводить матрицю, яка називається ядром, на вхід, виконуючи попарне множення матриці та підсумовуючи результат. Зауважимо, що фільтр і ядро – це одне і те ж [10].

Щоб застосувати фільтр, слід уявити, що він наводиться на вхідну матрицю, починаючи з верхнього правого краю, помножаємо кожне значення на відповідне значення ядра і підсумовуємо їх [9]-[10]. Потім ядро зміщується на певну величину, яка називається кроком, зазвичай на одиницю, і процес повторюється.

Щоб застосувати фільтр, слід уявити, що він наводиться на вхідну матрицю, починаючи з верхнього правого краю, кожне значення помножається на відповідне значення ядра і підсумовуються [9]-[10]. Потім ядро зміщується на певну величину, яка називається кроком, в даному випадку на одиницю, і процес знову повторюється [9]-[10].

Кількість ядер, що застосовуються одночасно, називається функціями. Наприклад, якщо застосовується чотири ядра 3×3 до зображення, то отримана матриця матиме чотири елементи [9]-[10].

На практиці, зазвичай, випадковим чином ініціалізуються номери ядра, складають їх разом і оновлюються під час навчання з використанням градієнтного спуску [9]-[10].

Одним з ключових аспектів операції згортки є розподіл ваг: найчастіше, одне і те ж ядро, сіра матриця, повторно використовується на кожному кроці, що охоплює все зображення [9]-[10]. Це значно скорочує

кількість необхідних ваг, знижуючи обчислювальні витрати. Більше того, це призводить до зміщення положення, оскільки мережу вчиться дивитися на вхідну матрицю, об'єднуючи пікселі близько один до одного. Це допомагає, оскільки сусідні пікселі, як правило, семантично пов'язані [9]-[10].

Зазвичай у нейронних мережах кілька блоків згорткової операції складаються разом, утворюючи шар. Потім кілька шарів складаються разом, як конструктори, щоб сформувати остаточну модель. Кількість шарів називається глибиною; чим більше шарів – тим глибша мережа [9]-[10].

Щоб ще більше знизити накладні витрати на обчислення, між шарами застосовуються різні види пулів. Управління пулом – складний процес. Він бере матрицю і зменшує її, підсумовуючи її значення. Найпоширенішим є максимальне об'єднання, де дослідник визначає розмір вікна та приймає максимальне значення всередині нього [9]-[10].

Кінцевий вектор, отриманий в результаті подачі пакета зображень у цю серію шарів згортки, потім передається повністю зв'язаному шару, який виводить вектор з класами [9]-[10].

1.2.2 Моделі на базі мережі AlexNet

На початку, активно використовувалися невеликі набори даних, такі як CIFAR та NORB, що складаються з десятків тисяч зображень [11]. Цих наборів даних було достатньо для моделей машинного навчання для вивчення основних завдань розпізнавання. Однак реальні прикладні задачі виявилися значно складнішими. Для науковців було розроблено та поширено нові великі набори даних, такі як ImageNet, які складаються з сотень тисяч або мільйонів позначених зображень, які емулюють реальні задачі, які вимагають надзвичайно ефективної моделі глибокого навчання [11].

CNN завжди були ідеальною моделлю для розпізнавання об'єктів – це надійні моделі, якими легко керувати та ще простіше навчати. Вони не стикаються із перенавчанням в будь-яких тривожних масштабах при

використанні на мільйонах зображень [11]. Їх продуктивність майже ідентична стандартним нейронним мережам прямого зв'язку однакового розміру. Єдина проблема: їх важко застосувати до зображень з високою роздільною здатністю. У масштабі ImageNet необхідно було впровадити інновацію, яка була б оптимізована для графічних процесорів (graphics processing unit – GPU) і скорочувала час навчання при одночасному підвищенні продуктивності [12].

ImageNet: набір даних, що складається з понад 15 мільйонів зображень з високою роздільною здатністю, позначених 22 тисячами ознаками-класами. ImageNet навіть має власний конкурс: ImageNet large-Scale visual Recognition Challenge (ILSVRC) [9]-[12]. Цей конкурс використовує підмножину зображень ImageNet і ставить перед дослідниками завдання досягти найменшої кількості помилок у топ-1 та топ-5 (частота помилок у топ-5 – це відсоток зображень, на яких правильна мітка не входить до п'яти найбільш вірогідних міток моделі) [9]-[12]. У цьому змаганні дані не є проблемою; існує близько 1,2 мільйона навчальних зображень, 50 тисяч перевірочних зображень і 150 тисяч тестових зображень. Автори застосували фіксовану роздільну здатність 256x256 пікселів для своїх зображень, обрізавши центральну частину кожного зображення розміром 256x256 пікселів [9]-[12].

Архітектура AlexNet складається з восьми шарів: п'яти згорткових шарів і трьох повністю пов'язаних шарів. Але не це робить AlexNet особливим; це деякі з використовуваних функцій, які є новими підходами до згорткових нейронних мереж [9]-[12].

Відносна нелінійність. AlexNet використовує випрямлені лінійні одиниці (ReLU) замість функції tanh, яка була стандартною на той час. Перевага ReLU полягає в часі навчання; CNN, що використовує ReLU, змогли досягти 25% помилки в наборі даних CIFAR-10 у шість разів швидше, ніж CNN, що використовує tanh [9]-[12].

У ті часи GPU все ще мали 3 гігабайти пам'яті (в наші дні такі види пам'яті були б новачками). Це було особливо погано, оскільки в навчальному наборі було 1,2 мільйона зображень. AlexNet дозволяє проводити навчання з декількома GPU, поміщаючи половину нейронів моделі на один GPU, а іншу половину – на інший GPU. Це не тільки означає, що можна навчити більшу модель, але й скорочує час навчання [9]-[12].

CNN традиційно об'єднують виходи сусідніх груп нейронів без перекриття. Однак, коли автори ввели перекриття, вони побачили зниження похибки приблизно на 0,5% і виявили, що моделі з перекриваються об'єднанням, як правило, важче налаштувати [9]-[12].

AlexNet мав 60 мільйонів параметрів, що було серйозною проблемою з точки зору перенавчання. Для зменшення перенавчання були використані два методи [9]-[12]:

- збільшення даних. Автори використовували перетворення збереження міток, щоб зробити свої дані більш різноманітними. Зокрема, вони генерували переклади зображень і горизонтальні відображення, що збільшувало Навчальний набір в 2048 разів [9]-[12]. Вони також провели аналіз основних компонентів значень пікселів Red-Green-Blue (RGB) для зміни інтенсивності каналів RGB, що знизило рівень помилок top-1 більш ніж на 1%;

- випадання. Цей метод полягає у відключенні нейронів із заздалегідь визначеною ймовірністю (наприклад, 50%). Це означає, що кожна ітерація використовує іншу вибірку параметрів моделі, що змушує кожен нейрон мати більш надійні функції, які можуть бути використані з іншими випадковими нейронами. Однак відсів також збільшує час навчання, необхідний для збіжності моделі [9]-[12].

У версії конкурсу ImageNet 2010 року найкраща модель отримала 47,1% помилок у топ-1 та 28,2% помилок у топ-5. AlexNet значно випередив цей показник із 37,5% помилок у топ-1 та 17,0% помилок у топ-5 [9]-[12]. AlexNet здатна розпізнавати об'єкти, що не мають центру, і більшість його

п'яти найкращих класів для кожного зображення є прийнятними. AlexNet переміг у конкурсі ImageNet 2012 року з частотою помилок у топ-5 15,3%, порівняно з другим місцем у топ-5 з частотою помилок 26,2% [9]-[12].

AlexNet – це неймовірно потужна модель, здатна досягати високої точності в дуже складних наборах даних. Однак видалення будь-якого з згорткових шарів різко знизить продуктивність AlexNet. AlexNet є провідною архітектурою для будь-якого завдання виявлення об'єктів і може мати величезне застосування в області комп'ютерного зору і завдань штучного інтелекту [9]-[12]. У майбутньому AlexNet може використовуватися частіше, ніж CNN, для завдань зображення [9]-[12].

Важливою віхою у забезпеченні більш широкого застосування глибокого навчання AlexNet також можна віднести впровадження глибокого навчання у суміжні галузі, такі як обробка природної мови та аналіз медичних зображень.

1.2.3 Моделі на базі мережі Residual Network

Після першої архітектури на базі CNN (AlexNet), яка виграла конкурс ImageNet 2012, кожна наступна архітектура-переможець використовує більше шарів у глибокій нейронній мережі для зменшення частоти помилок [7]-[8]. Це працює для меншої кількості шарів, але коли збільшують кількість шарів, виникає загальна проблема глибокого навчання, пов'язана з так званим градієнтом зникнення/вибуху. Це призводить до того, що градієнт стає рівним 0 або занадто великим. Таким чином, коли ми збільшуємо кількість шарів, рівень помилок навчання та тестування також збільшується [9]-[12].

Так в [] можна помітити, що 56-шаровий CNN дає більшу частоту помилок як у навчальному, так і в тестовому наборі даних, ніж 20-шарова архітектура CNN. Проаналізувавши детальніше частоту помилок, автори змогли прийти до висновку, що це викликано зникаючим/вибухаючим градієнтом [9]-[12].

ResNet, який був запропонований у 2015 році дослідниками Microsoft Research, представив нову архітектуру під назвою залишкова мережа.

Residual Network (ResNet): для вирішення проблеми градієнта зникнення/вибуху в цій архітектурі була введена концепція, яка називається залишковими блоками. У цій мережі використовують метод, який називається пропуском з'єднань. З'єднання з пропуском з'єднує активації шару з іншими шарами, пропускаючи деякі шари між ними. Це утворює залишковий блок. Повторні мережі створюються шляхом укладання цих залишкових блоків разом [9]-[12].

Підхід цієї мережі полягає в тому, що замість того, щоб шари вивчали базове відображення, мережі дозволяють відповідати залишковому відображенню [9]-[12].

Перевага додавання цього типу пропущеного з'єднання полягає в тому, що якщо будь-який рівень зашкодить продуктивності архітектури, то він буде пропущений регуляризацією. Таким чином, це призводить до навчання дуже глибокої нейронної мережі без проблем, спричинених градієнтом зникнення / вибуху [9]-[12].

Існує подібний підхід, який називається Highway networks, ці мережі також використовують пропускання з'єднання. Подібно до Long-short term memory (LSTM), ці пропусканні з'єднання також використовують параметричні механізми-ворота. Ці клапани визначають, скільки інформації проходить через пропущене з'єднання. Однак ця архітектура не забезпечує кращої точності, ніж архітектура ResNet [9]-[12].

Мережева архітектура: ця мережа використовує 34-шарову просту мережеву архітектуру, яка потім додає швидке підключення. Потім ці швидкі з'єднання перетворюють архітектуру в залишкову мережу [9]-[12].

1.2.4 Моделі на базі мереж Vision Transformers

Концепція Vision Transformer (ViT) є розширенням оригінальної концепції Transformer [8]. Це лише застосування Transformer в області зображень з невеликими змінами в реалізації для обробки різних модальностей даних. Більш конкретно, ViT використовує різні методи для токенизації та вбудовування [9]-[12]. Однак загальна архітектура залишається незмінною. Вхідне зображення розбивається на набір ділянок зображення, які називаються візуальними маркерами. Візуальні маркери вбудовані в набір закодованих векторів фіксованого розміру. Положення ділянки на зображенні вбудовується разом з закодованим вектором і подається в мережу кодера Transformer, яка по суті така ж, як та, яка відповідає за обробку текстового введення [9]-[12].

У кодері ViT є кілька блоків, і кожен блок складається з трьох основних елементів обробки: Layer Norm, Multi-head Attention Network (MSP) та Multi-Layer Perceptrons (MLP). Layer Norm підтримує процес навчання в потрібному руслі і дозволяє моделі адаптуватися до відмінностей між навчальними зображеннями. MSP – це мережа, відповідальна за генерацію карт уваги із заданих вбудованих візуальних токенів [9]-[12]. Ці карти уваги допомагають мережі зосередитися на найбільш важливих областях зображення, таких як об'єкт. Концепція карт уваги така ж, як і в традиційній літературі з комп'ютерного. MLP – це двошарова класифікаційна мережа з Gaussian Error Linear Unit (GELU) в кінці. Кінцевий блок MLP, який також називають головою MLP, використовується як вихід трансформатора. Додаток softmax на цьому виході може надавати класифікаційні мітки (тобто якщо додаток є класифікацією зображень) [9]-[12].

Завдяки їх універсальному характеру, застосування ViT охоплює майже всі аспекти комп'ютерного зору. Це включає класифікацію зображень, перетворення зображення в текст/перетворення тексту в зображення,

візуальне мислення, асоціативне навчання та мультимодальне навчання [9]-[12].

Завдання класифікації зображень є найбільш поширеною проблемою в області комп'ютерного зору. Найсучаснішими методами для задач класифікації зображень є методи, засновані на CNN. ViT не забезпечують порівнянної продуктивності на малих і середніх наборах даних, однак вони перевершили CNN на дуже великих наборах даних. Це пояснюється тим, що CNN кодує локальну інформацію на зображенні ефективніше, ніж ViT, через застосування локально обмежених сприйнятливих полів [9]-[12].

Більш досконала форма класифікації зображень може бути досягнута шляхом створення підпису, що описує вміст зображення, замість мітки з одним словом. Це стало можливим завдяки використанню ViT. ViT вивчають загальне уявлення даної модальності даних замість грубого набору міток, тому можна згенерувати описовий текст для даного зображення [9]-[12].

Більш просунутим застосуванням ViT є вивчення асоціацій, що існують між зображенням і фрагментом тексту. Це програма, яка вимагає справжньої здатності до узагальнення, оскільки вимагає абстрактного представлення як тексту, так і зображення. Цього можна досягти, навчивши два окремі кодери перетворення для фрагмента тексту та зображення. Потім закодоване зображення, а також текстові об'єкти можна порівняти на предмет їх відповідної подібності шляхом побудови матриці косинусної подібності. Відповідна реалізація для такої моделі трансформатора запропонована в Contrastive Language-Image Pre-Training (CLIP) [9]-[12].

1.2.5 Порівняння нейромережових основ для моделей класифікації зображень

Представимо порівняння розглянутих вище моделей на основі штучних нейронних мереж у вигляді таблиці 1.1. У якості критеріїв порівняння будемо використовувати такі:

- структурна складність – структурна складність мережі, що розглядається;
- використання GPU під час навчання – можливість використання GPU під час навчання з метою прискорення цього процесу;
- складність функцій активації – оцінка складності та ресурсоемності функції активації;
- робота на великих масивах даних – загальна оцінка роботи мережі з вибірками великих розмірів (Big Data).

Таблиця 1.1 – Порівняння нейромережевих моделей для класифікації зображень

	AlexNet	ResNet	ViT
Структурна складність	На базі CNN, більш спрощена архітектура (менше шарів, але більше нейронів)	На базі CNN, більш спрощена архітектура (менше шарів)	Більш складна, складена структура (Layer Norm + MSP+ MLP)
Використання GPU під час навчання	Можливе	Можливе	Можливе (ускладнене пересилкою даних)
Складність функцій активації	Ускладнена функція для підвищення точності (ReLU)	Ускладнена функція для підвищення точності (ReLU)	Складена функція для різних блоків (GELU + шари softmax)
Робота на великих масивах даних	Можлива (середні точність та швидкість роботи)	Можлива (середня точність та більша швидкість роботи)	Можлива (висока точність та низька швидкість роботи)

Переглядаючи результати порівняння можна прийти до висновку, що на сьогоднішній день загальна тенденція – це використання різноманітних модифікацій CNN. При цьому не завжди більш складна топологія гарантує більшу точність як у випадку порівняння результатів AlexNet та ResNet, адже ResNet відрізняється більш простою та зрозумілою архітектурою, але демонструє вищу точність. З іншого боку, використання комбінації з шарів різних топологій штучних нейронних мереж, як у випадку ViT демонструє найвищу точність, але це накладає значні обмеження у використанні сучасних обчислювальних можливостей (виконання паралельних обчислень на GPU). Це пояснюється тим, що під час виконання обчислень на паралельних потоках, постійні синхронізація та пересилка даних між потоками, вимагають іноді значно більше ресурсів, ніж самі паралельні обчислення.

Саме тому актуальною залишається задача вибору найбільш оптимальної архітектури нейромоделі для класифікації зображень, яка б відрізнялася топологічною простотою та можливістю повноцінного використання можливостей паралельних обчислювальних систем.

1.3 Підходи машинного навчання для навчання моделей класифікації зображень

Щоб навчити модель класифікувати зображення, потрібні величезні обсяги даних [13]. Підходів для машинного навчання моделі найчастіше виділяють два:

- з учителем;
- без вчителя.

Більш поширеним з двох є навчання з учителем. Це коли експерт надає системі зразки даних, які вже мають ознаку із позначенням класів (або правильними відповідями). Це вчить модель розпізнавати кореляції та застосовувати процедури до нових даних [13].

Навчання без вчителя зустрічається не так часто, як навчання з учителем. Навчання без вчителя характеризується неохайними, необробленими даними без участі людини. Цей підхід не використовує навчальні дані [14].

Однак навчання без вчителя може призвести до розв'язання задач, коли людина-експерт, ще не визначилася [13].

1.3.1 Навчання з учителем

Коли моделі надаються сигнали про помилки навчання, наприклад, класифікується конкретне зображення як кішка, але це була собака – виконується навчання з учителем. Це найпоширеніший сценарій, коли позначаються набори даних парами зображень та класів [15]-[16].

Нейронні мережі навчаються шляхом мінімізації функції, яка називається функцією втрат, за допомогою градієнтного спуску [15]-[16].

Для класифікації за однією ознакою визначають за математичною бінарною крос-ентропійною функцією втрат [15]-[16], формула (1.2):

$$L = -\frac{1}{m} \sum [y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)], \quad (1.2)$$

де m – це розмах (кількість елементів) вибірок;

y_i – ознака, що позначає один з класів;

$\hat{y}_i$ – значення, що визначається моделлю.

Функція Softmax. Модель виводить вектор довжиною, що дорівнює кількості класів, це оцінки класів. Оскільки вони можуть бути будь-якими, застосовується спеціальна функція, яка називається Softmax [15]-[16].

Це призводить до того, що ці числа підсумовуються до одиниці, тому кожен окремий елемент обмежений між 0 і 1. Чим ближче до одиниці, тим модель вважає цей клас правильним [15]-[16].

Softmax з (1.1) в цьому випадку перевизначається наступним чином, формула (1.3):

$$Softmax(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum e^{\hat{y}_j}}. \quad (1.3)$$

Отже, береться експонента вихідних даних і ділиться на суму експоненти всіх вихідних даних [15]-[16]. Наприклад, якщо є два класи, то результат: $\hat{y} = [2, 5]$.

Очевидно, що значення для класу 2, $\hat{y}_2$, є найбільшим, але необхідно, щоб вони були між 0 і 1. Отже, давайте застосується функція, і спочатку знайдеться знаменник: $\sum e^{\hat{y}_j} = 7,3 + 148 = 155$.

Потім береться показник кожного результату і ділиться на нього. Ділення на експоненту кожного вихідного значення [13]:

$$Softmax(\hat{y}) = \left[\frac{e^2}{155}; \frac{e^5}{155} \right] = [0,05; 0,95].$$

Число підсумовується до одиниці, і тепер фіксується правильна оцінка, присвоєна моделі раніше. Є причина чому було неможливо просто нормалізувати результати [15]-[16].

Якщо нормалізувати попередні значення, то на виході виходить $[0,28; 0,71]$, а якщо з Softmax $[0,05; 0,95]$. Softmax відсуває значення далі, ніж нормалізація, фактично приймаючи майже максимальне значення, звідси і назва Softmax [14].

Перехресна ентропія. Розглянемо приклад, якщо мета 0, але модель видає 0,9, тобто дуже близько до 1, значення втрат буде рівним.

Пам'ятаймо, що в цьому випадку модель виводить вектор довжиною, рівною кількості класів, вони є оцінкою класу. Ці числа в сумі дорівнюють одиниці, тому кожен окремий елемент обмежений між 0 і 1 [15]-[16].

Чим ближче до одиниці, тим більше модель вважає цей клас правильним: $L = -(0 - 2,30)$.

Якщо пригадати регресію, можна подумати, що можливим є застосувати середньоквадратичну помилку (mean squared error – MSE). MSE не оцінює модель з таким штрафом, як бінарна крос-ентропія (binary cross-entropy – BCE). Якщо обчислюється градієнт щодо $\hat{y}_i$ [15]-[16]:

$$\frac{\delta L}{\delta \hat{y}_i} = \frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} = 0 + \frac{1}{0,1} = 10.$$

У той час як з MSE:

$$L_{MSE} = (y_i - \hat{y}_i)^2 \mid \frac{\partial L_{MSE}}{\partial \hat{y}_i} = 2 \cdot (0 - 0,9) = 1,8.$$

Градієнт від BCE в 5 разів більше, таким чином, модель буде оштрафована більше [15]-[16].

Класифікація за декількома ознаками. Для класифікації за декількома ознаками необхідно, щоб результати були незалежними один від одного, оскільки одночасно може існувати кілька класів. Наприклад, якщо є два класи одночасно, результат може бути [20;21], і не можливо застосувати Softmax, оскільки це надасть більше значення одному конкретному значенню – найбільшому [15]-[16].

У цьому випадку застосовуємо сигмоїдальну функцію перед BCE., яка відображає між $[-1;1]$, вона визначається наступним чином [15]-[16]:

Sigmoid / Logistic

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Це гарантує, що кожне окреме значення у вихідних даних моделі обмежується одним і тим же діапазоном [15]-[16].

1.3.2 Навчання без учителя

Тут відсутня анотація, просто велика колекція (вибірка) необроблених зображень, і дозвіл для моделі з'ясувати, що вони собою являють під час навчання. Існують різні підходи до вирішення цього завдання [14].

Контрастне навчання. Під час контрастного навчання береться вхідне зображення x , якимось чином обирається позитивне зображення x_{pos} та негативне зображення x_{neg} і намагаємось навчити модель підрахунку балів таким чином, щоб оцінка $(x, x_{pos}) > \text{оцінка}(x, x_{neg})$.

Такі ескізи будуються випадковим чином, переконуючись, що кожен раз є інший позитивний і негативний приклад.

Якщо є мітки, негативний зразок повинен бути з іншого класу, тоді як позитивний – з того ж класу. Коли немає міток, одним із простих способів отримати зразок є вибір двох частин одного і того ж зображення для позитивного та однієї частини іншого випадкового зображення для негативного [13].

Враховуючи векторне представлення зображення, стратегія підштовхнути два позитивні якомога ближче, а негативні – якомога далі. Один із способів-вивчити метрику подібності, D [15]-[16], формула (1.4).

D – відстань між цими уявленнями.

$$D_i = \|x, x_i\|_2, \quad (1.4)$$

де D – відстань між позитивними та негативними уявленнями.

Якщо позитивний, необхідно, щоб функція втрат L підштовхнула їх ближче: $L_i = D_i^2$.

Отже, необхідно, щоб x і x_{pos} мали великий точковий продукт. З іншого боку, якщо є негативний зразок, відбувається навпаки: $L_i = \max(0, \varepsilon - D_i)^2$. Тут ε – діє як маржинальне значення [15]-[16].

Залежно від того, як вирішується використовувати приклад, є можливість отримати попарні втрати, так з (x, x_{pos}) і (x, x_{neg}) або потрібні втрати з (x, x_{pos}, x_{neg}) всі разом.

При втратах Triplet спочатку обчислюються відстані між (x, x_{pos}) і (x, x_{neg}) , а потім використовуємо їх у втраті [15]-[16], формула (1.5):

$$\begin{aligned} D_{pos} &= \|x, x_{pos}\|^2, \\ D_{neg} &= \|x, x_{neg}\|^2, \\ L &= \max(0, D_{pos}^2 - D_{neg}^2 - m), \end{aligned} \tag{1.5}$$

де m – змінна, яка діє як запас.

Ще один дуже поширений варіант обрахунку втрат – це InfoNCE, в якій тут можемо навести багато негативних прикладів [15]-[16].

InfoNCE – це логарифмічна ймовірність прогнозування правильного прикладу із заданого набору прикладів на основі оцінки моделі, формула (1.6).

$$InfoNCE = -E \left[\log \frac{s(x, y)}{\sum_{y_i} s(x, y_i)} \right], \tag{1.6}$$

де s – функція оцінки, зазвичай є внутрішнім продуктом. Чим вище, тим більше вони схожі, і навпаки.

Одним з дуже успішних методів, що використовують контрастне навчання, є SimCLR, в якому є мережа, як правило, ResNet-50, і лінійна проекція [15]-[16].

Відбирається міні-пакет із набору даних і для кожної точки даних, застосовується збільшення даних і передача їх через мережу, і лінійна проекція отримує два подання [15]-[16]. Це ефективно створює дві позитивні пари, оскільки збільшуються однакові зображення, потім застосовуються контрастні втрати до всіх збільшених точок даних у пакеті, відстежуючи позитивні пари [15]-[16].

1.3.3 Навчання з підкріпленням

Навчання підкріплення – це область машинного навчання, яка розглядає прийняття відповідних заходів для максимізації винагороди в конкретній ситуації [13]-[16]. Цей підхід використовується різним програмним забезпеченням і машинами для пошуку найкращої можливої поведінки або шляху, який варто обрати в конкретній ситуації. Навчання з підкріпленням відрізняється від навчання з учителем тим, що при навчанні з учителем навчальні дані містять ключ до відповіді, тому модель навчається з правильною відповіддю сама по собі, тоді як при навчанні з підкріпленням відповіді немає, але агент підкріплення вирішує, що робити для виконання заданого завдання. За відсутності навчального набору даних він зобов'язаний вчитися на своєму досвіді [13]-[16].

У відомому прикладі наводиться робот, алмаз і вогонь. Мета робота полягає в тому, щоб отримати нагороду у вигляді діаманта і уникнути перешкод, які запускаються [13]-[16]. Робот вчиться, пробує всі можливі шляхи, а потім вибираючи той, який дає йому нагороду з найменшою кількістю перешкод. Кожен правильний крок дасть роботів винагороду, а

кожен неправильний крок забере винагороду у робота. Загальна нагорода буде розрахована, коли вона досягне остаточної нагороди-діаманта [13]-[16].

Головні принципи навчання з підкріпленням [13]-[16]:

- вхідні дані: вхідними даними має бути початковий стан, з якого почнеться модель;
- результат: існує багато можливих результатів, оскільки існує багато рішень конкретної проблеми;
- навчання базується на вхідних даних, модель поверне стан, і користувач вирішить винагородити або покарати модель на основі її результатів;
- модель продовжує вчитися.
- найкраще рішення приймається на основі максимальної винагороди.

1.3.4 Порівняння підходів машинного навчання для класифікації зображень

Виконаємо порівняння методів машинного навчання для класифікації зображень за такими критеріями:

- вимоги до вхідних даних – загальна оцінка вимог підходу машинного навчання та відповідних методів до організації та представлення вхідних даних;
- вимоги до метапараметрів навчання – вимоги до параметрів та налаштувань необхідних для виконання процесу навчання;
- складність організації – оцінка складності організації процесу навчання моделі, в тому числі з точки зору обчислювальних ресурсів;
- адаптованість рішень (моделей) – оцінка адаптованості та універсальності отриманих рішень, їх можливість роботи при оновлених даних про задачу (або середовище).

Результати аналізу представлені у вигляді таблиці 1.2.

Таблиця 1.2 – Порівняння підходів до машинного навчання

	Навчання з учителем	Навчання без учителя	Навчання з підкріпленням
Вимоги до вхідних даних	Найвищі	Мінімальні	Середні
Вимоги до метапараметрів навчання	Високі	Високі	Середні
Складність організації	Середня	Висока	Висока
Адаптованість рішень (моделей)	Середня	Висока	Найвища

З результатів порівняння можна зробити висновок, що загалом навчання з учителем та без учителя, як основні підходи до машинного навчання вже налічують велику кількість опрацьованих та досліджених переваг та недоліків. Так сама концепція навчання з учителем вимагає детального препроцесингу вхідних даних з їх чіткою класифікацією на навчальній частині. Навчання без учителя не вимагає такого від даних та може працювати навіть з даними без особливої перевірки, проте більшість методів навчання без вчителя вимагають попереднього визначення багатьох метапараметрів для початку процесу навчання.

Навчання з підкріпленням більш сучасний підхід, який намагається вирішувати знані проблеми попередніх стратегій. Так вимоги до вхідних даних про середовище в нього є, але вони не надмірні, а метапараметри зазвичай обмежуються визначенням критеріїв зупинки та визначення успішності/неуспішності навчання. Так само рішення (моделі) отримані на основі навчання з підкріпленням відрізняються найбільшою адаптивністю та універсальністю адже одразу синтезуються з урахуванням мінливості середовища в якому діє агент.

1.4 Постановка завдання

Нехай задано вибірку даних, яка складається з зображень, які характеризуються низкою ознак – $\{x_1, x_2, \dots, x_n\}$. Тоді використовуючи модель на основі штучної нейронної мережі – $Model = \{ANN\}$, необхідно пропускаючи через неї всю вибірку даних $DataSet = \{x_1, x_2, \dots, x_n, y\}$, визначити клас кожного з зображень – визначити залежну ознаку $\{y\}$. Умовами регулювання процесу навчання є визначення нагороди/штрафу під час навчання, з метою отримання рішення $Model = \{ANN\}$, що характеризується найбільшою точністю роботи: $Error_{ANN} \rightarrow 0$.

1.5 Висновки за розділом 1

Під час роботи над першим розділом було розглянуто проблемну область класифікації зображень. В результаті проведеного аналізу було з'ясовано, що одним з найбільш актуальних підходів до автоматизації класифікації зображень, є використання аналітичних моделей на базі машинного навчання, зокрема штучних нейронних мереж. Найбільш перспективним є підхід навчання з підкріпленням.

2 МАТЕРІАЛИ ТА МЕТОДИ

В першому розділі було проаналізовано сучасне становище у автоматичній класифікації зображень. Було проаналізовано сучасні практики застосування машинного навчання та засобів штучних нейронних мереж для вирішення актуальних проблем класифікації зображень. Попередній аналіз довів, що навчання з підкріпленням є перспективною основою для підвищення точності методів класифікації зображень.

Тому актуальним є дослідження та реалізація нових методів навчання з підкріпленням на базі штучних нейронних мереж, які б були вільні від недоліків попередніх структур та топологій.

Другий розділ буде присвячено дослідженню медулярних нейронних мереж. Особливостей їх архітектури та будови. Також окрему увагу буде приділено методам навчання такої топології.

Розділ буде організовано наступним чином: на початку, буде розглянуто роботу в області медулярних нейронних мереж із введенням спеціалізованих позначень і термінології; потім буде детально описано компоненти базової реалізації пропонованого підходу навчання з підкріпленням для навчання аналітичних моделей на базі медулярних нейронних мереж з урахуванням особливостей топології та виправлення недоліків існуючих методів.

2.1 Модулярні нейронні мережі

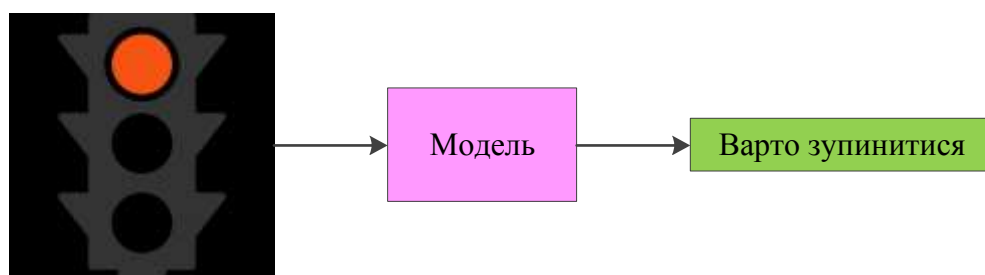
Основна ідея модулярних нейронних мереж (МНМ) ґрунтується на модульності, та полягає в тому, що: найкращий підхід до вирішення заданої складної задачі – це не виконання однієї гігантської задачі, а система окремих і в основному незалежних підзадач, що працюють разом для досягнення однієї великої мети [17]-[20]. Ця концепція має біологічну основу. У мозку існують функціонально спеціалізовані області, специфічні

для різних когнітивних процесів. У частині мозку, яка називається таламусом, знаходиться бічне колінчасте ядро, яке розділене на шари, які окремо обробляють колір і контраст: обидва основні компоненти зору [17]–[20]. Модульні нейронні мережі використовують цю ідею для вирішення складних завдань штучного інтелекту. Кілька незалежних нейронних мереж навчаються одночасно для конкретної підзадачі, і їх результати в кінці об'єднуються для виконання одного завдання. Переваги модульних нейронних мереж включають [17]–[20]:

- простоту структурної будови та організації;
- комбінація технік та методів навчання;
- масштабованість рішень;
- ефективне використання обчислювальних систем.

Як відомо, штучний інтелект за своєю суттю не володіє мовою, тому головне завдання – навчити аналітичну модель міркувати над заданим текстом. Під міркуванням маються на увазі такі завдання, як арифметика, сортування, порівняння та підрахунок [17]–[20].

Припустимо, необхідно створити систему відповідей на запитання, Рисунок 2.1 ілюструє, як працює модель контролю якості [17]–[20].



Як рухатися далі?

Рисунок 2.1 – Візуалізація відповіді на запитання – завдання відповісти на запитання про зображення, щоб показати, що система розуміє зображення.

Так, в задачі з визначенням, про що говорить знак – кольорова позначка світлофора на рис. 2.1. Для відповіді на таке питання необхідно

виконати кілька кроків міркування: знайти світлофор, визначити кольорову позначку, з'ясувати колір а потім надати рішення про подальші дії [17]-[20].

МНМ здатні забезпечити таке міркування. Використовуючи дане питання, модель збирає конкретну мережеву архітектуру, а потім виконує зібрану мережу нейронних модулів для виведення відповіді, що показано на рис. 2.2 [17]-[20].

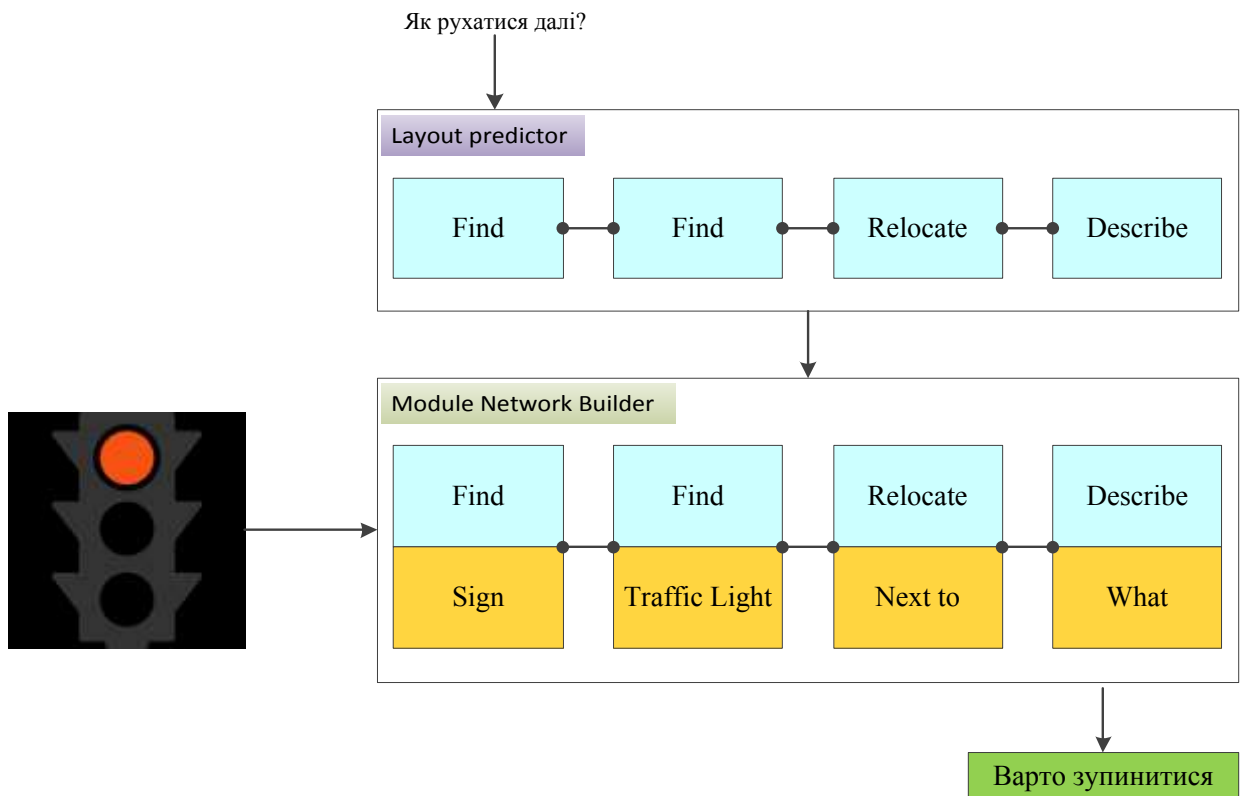


Рисунок 2.2 – Для кожного екземпляра модель спочатку передбачає макет, потім, використовуючи функції зображення, вона будує зібрану мережу нейронних модулів для виведення відповіді

2.1.1 Предиктор компоновки

Для вирішення проблеми контролю якості було створено безліч неідентичних макетів моделей [17]-[20]. Недавній огляд літератури на цю тему показав, що більшість з них перевершують за конкретною формою питання. Компонент гнучкої компоновки створений для того, щоб використовувати переваги всіх цих різних моделей. Щоб знайти найкращий

макет l для конкретного питання q , спочатку потрібно розрахувати розподіл ймовірностей $p(q|l)$, а потім, використовуючи пошук променя, макет з максимальною ймовірністю буде визначений пізніше. Припустимо, що кожен шар являє собою послідовність модулів [17]-[20]:

$$l = \{m^{(t)}\}.$$

Оскільки маємо послідовну форму модулів, модель LSTM може бути використана для пошуку розподілу ймовірностей для кожного можливого розташування. По-перше, передаємо функції питання $f_Q(q)$ в модель LSTM для генерації прихованих станів [17]-[20], формула (2.1):

$$H = [h_1, h_2, \dots, h_T], \quad (2.1)$$

де q – запит на кожний стан T .

Ваги уваги вхідних питальних слів формула (2.2) прогнозуються як:

$$a_{ti} = \frac{e^{e_{ti}}}{\sum_{j=1}^T e^{e_{tj}}} = \text{softmax}(e_{ti}), \quad (2.2)$$

де e_{ti} – модель вирівнювання формула (2.3) яка визначається як:

$$e_{ti} = v^T \tanh h(W_1 h_i + W_2 h_t), \quad (2.3)$$

де h_i і h_t – є виходами LSTM кодера і декодера відповідно.

Контекстний вектор обчислюється за формулою (2.4) так:

$$c_t = \sum_{i=1}^T a_{ti} h_i. \quad (2.4)$$

Цей контекстний вектор буде використовуватися для обчислення ймовірності для наступного модуля [17]-[20], формула (2.5):

$$p(m^{(t)} | m^{(1)}, \dots, m^{(t-1)}, q) = \text{softmax}(W_3 h_t + W_4 c_t). \quad (2.5)$$

Нарешті, розподіл ймовірностей l визначається за формулою (2.6):

$$p(l | q) = \prod_{m_t \in l} p(m^{(t)} | m^{(1)}, \dots, m^{(t-1)}, q). \quad (2.6)$$

2.1.2 Модулі

Модулі створюються для самостійного виконання основних завдань з міркування. Подумайте про них як про функції, які приймають вектори уваги, текстові чи Графічні об'єкти, і вони мають деякі внутрішні параметри для виконання певного обчислення. Давайме поглянемо на деякі приклади [17]-[20]:

- `find [x]` – модуль пошуку згортає кожну позицію у вхідному зображенні з вектором ваги (відмінним для кожного x), щоб привернути увагу [17]-[20];

- `Find-max-num[P]` – враховуючи уривок, цей модуль спочатку обчислює розподіл токенів числа T , потім використовує це для обчислення ймовірності того, що кожен маркер числа має максимальне значення [17]-[20];

- `та/або` – модулі `And` та `Or` приймають два зображення як вхідні дані та повертають їх перетин або об'єднання відповідно [17]-[20];

– relocate – модуль переміщення змінює карту вхідного зображення на нову карту уваги, яка може бути корисною для просторового співвідношення [17]-[20].

Наскрізне навчання. Як і в інших нейронних мережах, в процесі навчання метою є мінімізація втрат. Але різниця тут полягає в тому, що модель спільно вивчає прогнозований макет $p(q | l)$ та параметри в кожному модулі [17]-[20].

2.2 Використання МНМ та навчання з підкріпленням для класифікації зображень

МНМ відносяться до штучних нейронних мереж, які складаються з декількох різних нейронних мереж, з'єднаних разом у поєднанні з посередником. Щоб проілюструвати цей момент далі, розглянемо споживача, який володіє кількома розумними пристроями, такими як смартфон, смарт-годинник і планшет, наприклад iPad, на додаток до ноутбука або настільного комп'ютера [17]-[20]. Незважаючи на різні можливості цих відповідних пристроїв, всі вони будуть підключені до модему або маршрутизатора, що дозволить користувачам зазначених пристроїв швидко і ефективно отримувати доступ до онлайн - і мобільних сервісів. Крім того, це онлайн-з'єднання також дозволяє користувачам поєднувати функціональність своїх різних пристроїв для досягнення певної мети, наприклад, передавати популярну телевізійну програму або телефонувати другові чи члену сім'ї, серед іншого [17]-[20].

З огляду на все вищесказане, МНМ дають розробникам ПЗ можливість використовувати можливості окремих нейронних мереж більш узгодженим і ефективним способом. Для цього кожна нейронна мережа в межах більшої модульної нейронної мережі буде використана для вирішення частини конкретної проблеми [17]-[20]. На цьому етапі посередник, відомий як інтегратор, буде використовуватися для організації та аналізу цих

численних модулів для створення кінцевого результату нейронної мережі. Завдяки такій конфігурації прості нейронні мережі можуть бути реалізовані більш складним чином, і деякі поширені програми цих мереж включають обробку зображень, стиснення вхідних даних високого рівня та програмне забезпечення для прогнозування фондового ринку [17]-[20].

Ансамблеве навчання. Концепції та ідеї, які лягли в основу створення модульних нейронних мереж, були вперше теоретизовано в 1980-х роках і призвели до розробки методу машинного навчання, який називається ансамблевим навчанням [17]-[20]. Цей метод заснований на ідеї, що слабші моделі машинного навчання можуть бути об'єднані разом, щоб створити єдину сильнішу модель [17]-[20]. Більш того, цей колективний підхід може бути використаний для отримання більш істотних результатів, ніж ті, які могли б бути отримані при використанні єдиної моделі глибокого навчання. Іншим способом концептуалізації цього процесу є підхід розділяй і володарюй, при якому велика проблема вирішується шляхом розбиття зазначеної проблеми на більш дрібні частини, які можуть бути вирішені більш простим або життєздатним способом [17]-[20].

Структура МНМ. Відповідно до ідеї про те, що нейронні мережі базуються на численних функціях і можливостях людського мозку, людський мозок складається з ієрархії мереж, що складаються з мільйонів нейронів, причому кожна мережа спеціалізована для виконання певної роботи, пов'язаної з функціонуванням людського тіла [17]-[20]. Щоб навести приклад цього, розглянемо 3 різні нейронні мережі з прямим зв'язком, які були навчені для вирішення проблем, пов'язаних з розпізнаванням образів, таких як розпізнавання номерних знаків. З багатьох причин ці 3 мережі можуть мати труднощі з аналізом та ідентифікацією номерних знаків на відео чи зображеннях на постійній основі [17]-[20]. Таким чином, розробник програмного забезпечення міг би об'єднати вихідні дані цих трьох моделей для створення єдиного вихідного сигналу з метою створення єдиної моделі, яка є більш точною, ніж попередні 3 моделі [17]-[20].

Крім того, розробники ПЗ можуть також використовувати МНМ, щоб розбити саму проблему машинного навчання на менші, більш керовані частини [17]-[20]. Повертаючись до прикладу розпізнавання номерних знаків, цю проблему можна розбити на три підзадачі, такі як ідентифікація номерних знаків на зображеннях, ідентифікація номерних знаків на відео та ідентифікація номерних знаків на відео, які зображують погану погоду, таку як сильний дощ або снігопад. Завдяки цим об'єднаним висновкам розробник міг би створити алгоритм глибокого навчання, який був би здатний ідентифікувати появу номерних знаків як на зображеннях, так і на відео, які зображують різні погодні умови, з якими може зіткнутися водій при проходженні жвавого перехрестя або вулиці [17]–[20].

2.2.1 Переваги використання МНМ для класифікації зображень

Використання МНМ дає широкий спектр переваг. Наприклад, поєднання декількох нейронних мереж, які були навчені для виконання певного завдання, може значно скоротити час навчання, який часто потрібен для навчання нейронних мереж, а також пов'язані з цим витрати [17]-[20]. І навпаки, цей підхід також дозволяє розробникам програмного забезпечення поєднувати різні типи моделей та підходів машинного навчання, які були б неможливі за допомогою традиційних методів. Нарешті, як уже було сказано, поєднання декількох нейронних мереж може призвести до розробки єдиної нейронної мережі, яка здатна працювати набагато точніше та ефективніше, ніж будь-яка з попередніх мереж була здатна самостійно [17]-[20].

Оскільки штучні нейронні мережі, як правило, за останні кілька років стали гарячою темою як для досліджень, так і для розробки програмного забезпечення, доречно, що МНМ – це підхід, який також стає все більш популярним [17]-[20]. Більше того, оскільки вартість навчання моделей машинного навчання на практиці може бути непомірною, МНМ дозволяють розробникам програмного забезпечення створювати моделі набагато

дешевше та стійкіше. При цьому МНМ будуть продовжувати використовуватися в найближчі роки, оскільки підходи, які вважалися неадекватними в попередніх сценаріях, можуть бути об'єднані для створення єдиного життєздатного рішення [17]-[20].

2.2.2 Переваги використання навчання з підкріпленням для навчання моделей на базі МНМ при вирішенні задачі класифікації зображень

При звичайному формулюванні задачі класифікації зображень, як завданні комп'ютерного зору воно зводиться до максимізації довірчої функції з набору гіпотетичних місць розташування цілі, де достовірність може бути вивчена в повністю контрольованій або слабо контрольованій установці [17]-[20]. У формулюванні розсудного вікна набір гіпотез складається з великого набору прямокутних вікон, і задача максимізації вирішується шляхом вичерпного пошуку. Оскільки цей процес, як правило, занадто дорогий на практиці, було запропоновано безліч методів для його прискорення, від методологій, які використовують властивості довірчої функції, до методів пропозиції або каскадних методів. Всі ці методи зберігають властивість вичерпного пошуку в просторі гіпотез, спрямоване або на зменшення кількості гіпотез для початку, або на їх ефективний пошук [17]-[20].

Навпаки, біологічні системи мають модель пошуку, яку можна охарактеризувати як покрокову фіксацію, коли послідовно досліджується невеликий набір місць розташування сцени, щоб накопичити достатні докази цільового розташування [17]-[20]. Залишаючи осторонь ефективність (досліджується лише кілька областей зображення) та біологічну правдоподібність, все ще здається цікавим формально вивести математичні моделі, які могли б оптимально збалансувати ефективність та точність, послідовно інтегруючи докази принципово. Завдання полягає в тому, щоб

мати можливість працювати з відстроченими винагородами, які виключають нагляд на кожному етапі [17]-[20]. У той же час уникайте необхідності повністю попередньо вказувати середовище, що для візуальних сцен було б неможливо - враховуючи складність зображень і категорій візуальних об'єктів, моделі повинні бути ефективно навчені [17]-[20].

Формулюючи послідовний пошук, як рушій вивчення категорії і політику пошуку, залежить від зображення, включаючи умови зупинки, слід розробляти повністю навчання методи, які можуть явно збалансувати суперечливі цілі дослідження [17]-[20]:

- вибірка більшої кількості областей зображення для підвищення точності і експлуатації;
- ефективна зупинка пошуку при достатньої впевненості в розташування цілі.

Методологія є загальною, застосовною до будь-якої функції відгуку детектора і дозволяє вивчати стратегії пошуку і умови зупинки, які залежать від зображення і візуальної категорії [17]-[20].

Один клас ефективних детекторів зосереджується на використанні евристики гілок та меж для визначення пріоритетності дослідження простору пошуку в напрямку перспективних областей зображення [17]-[20]. На відміну від цієї роботи, такі методи застосовуються лише до класів довірчих функцій, для яких доступні суворі межі. Крім того, при відсутності мети на зображенні методи цього класу вироджуються в вичерпний пошук. Керуючись цими обмеженнями [17]-[20], інші автори запропонували використовувати каскади класифікаторів, для поступового звуження простору пошуку, де слабкі, але швидкі класифікатори застосовуються на ранній стадії, щоб виключити області зображення, які навряд чи містять ціль, при одночасному інвестуванні обчислювальних ресурсів для запуску більш складних класифікаторів в перспективних областях [17]-[20]. Такі методи значно знижують вартість обчислень, але класифікатори на ранній стадії каскаду все ще повинні бути застосовані вичерпно до всіх областей зображення. Замість

того, щоб зосередитись на стратегії дослідження регіону, інші методи були спрямовані на оптимізацію оцінки функції довіри. Це включає в себе спільне використання обчислень між сусідніми областями зображення бо серед різних класифікаторів для багатокласових завдань виявлення [17]-[20].

Останні тенденції у виявленні об'єктів зосереджені на швидкому скороченні набору кандидатів на основі вмісту – у попередніх методах вікна, обрізані в різних положеннях і з різними співвідношеннями сторін на зображенні - до меншого набору (у більшості методів все ще тисячі гіпотез), який демонструє статистичні закономірності об'єкти, знайдені в реальному світі. Типові методології включають параметричну сегментацію за фігурами за допомогою установок: об'єктно-подібній фільтрації та сигналів на основі кордонів [17]-[20].

Загалом переваги використання навчання з підкріпленням при класифікації зображень зводяться до:

- навчання з підкріпленням може бути використано для вирішення дуже складних проблем, які не можуть бути вирішені звичайними методами, коли об'єми вхідних даних занадто великі [17]-[20];
- цей підхід є кращим для досягнення високих показників адаптивності та універсальності рішень [17]-[20];
- надає можливість виправляти помилки, що виникли в процесі навчання [17]-[20];
- як тільки помилка виправлена моделлю, ймовірність виникнення такої ж помилки дуже мала;
- за відсутності навчального набору даних підхід зобов'язаний вчитися на своєму досвіді [17]-[20];
- моделі навчання підкріплення можуть перевершити людей у багатьох завданнях [17]-[20];
- навчання з підкріпленням призначене для досягнення ідеальної поведінки моделі в певному контексті, щоб максимізувати її продуктивність [17]-[20];

– методи навчання з підкріпленням підтримують баланс між дослідженням та експлуатацією. Дослідження – це процес випробування різних речей, щоб побачити, чи вони кращі за те, що було випробувано раніше. Експлуатація-це процес випробування того, що найкраще працювало в минулому. Інші алгоритми навчання не забезпечують такого балансу [17]-[20].

2.3 Класифікація зображень нейромоделями на базі МНМ з навчанням із підкріпленням

Щоб побудувати систему МНМ, потрібно вибрати відповідний розмір мереж для кожного модуля, а потім знайти середню ймовірність softmax для групування категорій. Система МНМ побудована за принципом знизу вгору. Процес побудови навченої системи МНМ описана в форматі псевдокоду на рисунку 2.3. Вхідними даними для методу є навчальний набір даних з вибірки і гіперпараметр t для вибору розміру кожного модуля. Результатом роботи методу є навчена система МНМ [17]-[20]. Спочатку починають з однієї супергрупи, яка містить усі категорії набору даних. Потім обирається розмір мережі. Вихідна матриця softmax, подібна для навченої мережі. Після чого середня ймовірність softmax використовується як показник подібності для групових категорій. Якщо формуються нові дочірні супергрупи, модуль навчається класифікації між новими супергрупами. Навчання модулів МНМ описано далі. Цикл повторюється до тих пір, поки не будуть навчені модулі всіх супергруп [17]-[20].

Вибір розмірів нейронної мережі і гіперпараметрів. Необхідно вибрати кількість шарів і розмір (конфігурацію) кожного модуля. Слід використовувати нову метрику, яка називається щільністю зміни точності [17]-[20] (ΔAD). Вона вимірює приріст точності на одиницю збільшення розміру моделі між двома моделями мережі, формула (2.7):

$$\Delta AD_{ij} = \frac{VA_i - VA_j}{MS_i - MS_j}, \quad (2.7)$$

де VA – точність перевірки, отримана під час класифікації зображень;

MS – розмір мережі блоку;

i, j – посилання на кількість шарів у двох моделях мереж.

```

inputs: Training dataset  $(x, y)$   $x$ : images and  $y$ : labels; threshold  $\tau$ .
output: MNN-Tree structure  $T$ .
CATEGORIES =  $\{c: c \text{ is a unique label} \in y\}$  // set of all categories
TREE =  $\{\text{untrained root DNN}\}$  // set holding the tree structure
 $\mathbb{S} = \text{CATEGORIES}$  // all categories of the dataset come under the root
while ( $\exists$  an untrained DNN in TREE)

    (1)  $lc = 0$  // layer_count: keeps track of the number of layers in the DNN
    (2) do
         $lc = lc + 1$ 
        initialize a DNN  $D_{lc}$ , with  $lc$  convolutional layers.
        train  $D_{lc}$  to classify all categories of  $\mathbb{S}$ .
        if  $lc \neq 1$  then calculate  $\Delta AD_{lc, lc-1} = \frac{VA_{lc} - VA_{lc-1}}{MS_{lc} - MS_{lc-1}}$ 
        while ( $lc = 1$  or  $\Delta AD_{lc, lc-1} > \tau$ )
    (3)  $DNN_{config} = D_{lc-1}$  // select the DNN configuration for the module
    (4)  $\text{SOFTMAX\_MATRIX} = \text{softmax output of } DNN_{config} \forall c \in \mathbb{S}$  // softmax output matrix
    (5)  $\text{CHILD\_GROUPS} = \phi$  // keeps track of newly formed children super-groups
    (6) for each  $c \in \mathbb{S}$  //use the ASL
        find set  $\mathbb{V}$ , s.t.  $\{\mathbb{V} \subset \mathbb{S} : \text{prob}(c \sim v) = \text{sgmf}(\text{SOFTMAX\_MATRIX}[c, v]), \forall v, c \in \mathbb{V}\}$ 
        add untrained DNN corresponding to  $\mathbb{V}$  into TREE
         $\text{CHILD\_GROUPS} = \text{CHILD\_GROUPS} \cup \mathbb{V}$  // similar categories added to set of children
    (7) train  $DNN_{config}$  to classify between elements of  $\text{CHILD\_GROUPS}$ 
    (8)  $\mathbb{S} = \text{children of next untrained DNN in TREE}$ 

return tree

```

Рисунок 2.3 – Фрагмент псевдокоду розробленого методу

Якщо розглядати ефективність використання ΔAD як показника, слід зазначити, що при збільшенні розмірів моделі помітного підвищення точності не відбувається. Точність обчислюється для кожного модуля МНМ до того, як інші мережі будуть згруповані в супергрупи [17]-[20]. Наприклад, ΔAD для кореневого модуля МНМ обчислюється на основі точності, отриманої для класифікації між усіма категоріями набору даних, а ΔAD для будь-якого іншого модуля в МНМ обчислюється при класифікації між усіма його дочірніми категоріями [17]-[20]. Використовуючи ΔAD , можна визначити ефективні конфігурації мереж і з високою точністю розрізнити

категорії. Такий підхід може призвести до незначної втрати точності порівняно з великими монолітними нейронними мережами, оскільки малим мережам з МНМ потрібно класифікувати лише кілька груп візуально подібних категорій, а не всі категорії набору даних.

2.4 Висновки за розділом 2

У другому розділі проаналізовано базову реалізацію МНМ та методи їх машинного навчання..

Так МНМ було обрано для синтезу моделей для автоматизації класифікації зображень. Такий підхід реалізує усі переваги навчання з підкріпленням для автоматизації завдання класифікації.

Використовуючи нові показники та критерії, можна визначити ефективні конфігурації мереж і з високою точністю розрізняти категорії. Такий підхід може призвести до незначної втрати точності порівняно з великими монолітними нейронними мережами, оскільки малим мережам з МНМ потрібно класифікувати лише кілька груп візуально подібних категорій, а не всі категорії набору даних.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Другий розділ було присвячено дослідженню МНМ: їх базових реалізацій, можливих модифікацій та особливостей застосування для класифікації зображень. На початку було розглянуто роботу в області навчання з підкріпленням, потім було детально описано компоненти базової реалізації пропонованого підходу комбінації МНМ та навчання з підкріпленням..

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, фреймворку машинного навчання, інтегрованого середовища розробки, тощо.

3.1 Опис основних вимог до програми

Об'єктом розробки є ПЗ для реалізації навчання МНМ з використанням методів навчання з підкріпленням для подальшої автоматизації процесу класифікації зображень з великих набірів даних. Мета роботи – розробити програмний застосунок, що в повній мірі дозволить застосувати та дослідити роботу МНМ, що пройшли навчання на основі розробленого методу.

Розробка ПЗ зводиться до реалізації основних модулів системи. Так, перший модуль представляє частину, що розгортається та доступна користувачу з метою певної авторизації та підвантаження вибірок даних. Інший модуль має розгортатися на сервері та забезпечувати можливість віддаленої обробки даних використовуючи програмні бібліотеки. З метою зберігання та представлення результатів роботи необхідно реалізувати третій модуль, що буде відповідати за зв'язок із базою даних (БД) та інтерпретацію підвантажених з нею даних.

Функціональні вимоги до розроблюваного ПЗ формують такий перелік:

- авторизація (верифікація) користувача;
- підвантаження вибірок вхідних даних;
- пересилка даних на серверну частину;
- виконання побудови та навчання МНМ;
- представлення результатів;
- збереження результатів роботи (реалізація експорту та імпорту даних про результати обчислень).

Для забезпечення надійності програми, її складові частини мають бути розгорнуті між серверною станцією та персональним комп'ютером (ПК) користувача. Протягом сеансу роботи застосунок не повинен збоїти або зависати. Після внесення оновлень за стосунок має перзавантажитися та знову бути доступним.

БД яка формується на основі даних за стосунку та проміжних результатів його роботи має відповідати наступним умовам:

- зміни до бази вносяться тільки від верифікованих користувачів;
- напряму із БД може контактувати виключно серверна частина ПЗ або адміністратор;
- оновлення БД виконується виключно на основі внутрішніх правил.

3.2 Вибір мови програмування

Проєкти штучного інтелекту відрізняються від традиційних програмних проєктів [21]-[22]. Відмінності полягають у технологічному стеку, навичках, необхідних для проєкту на основі штучного інтелекту, та необхідності глибоких досліджень. Щоб реалізувати свої прагнення до штучного інтелекту, слід використовувати мову програмування, яка є стабільною, гнучкою та має доступні інструменти [21]-[22]. Python пропонує

все це, саме тому він один з найпопулярніших для проєктів штучного інтелекту на Python [21]-[22].

Починаючи з розробки і закінчуючи розгортанням і обслуговуванням, Python допомагає розробникам бути продуктивними і впевненими в ПЗ, яке вони створюють [21]-[22]. Переваги, які роблять Python найкращим чином придатним для машинного навчання та проєктів на основі штучного інтелекту, включають простоту та послідовність, доступ до чудових бібліотек та фреймворків для штучного інтелекту та машинного навчання, гнучкість, незалежність від платформи та широке співтовариство. Це збільшує загальну популярність мови [21]-[22].

Простий і послідовний. Python пропонує короткий і читабельний код [21]-[22]. У той час як за машинним навчанням і штучним інтелектом стоять складні алгоритми і універсальні робочі процеси, простота Python дозволяє розробникам створювати надійні системи. Розробники можуть докласти всіх зусиль до вирішення проблеми машинного навчання, замість того, щоб зосереджуватися на технічних нюансах мови [21]-[22].

Крім того, Python приваблює багатьох розробників, оскільки його легко освоїти. Код на Python зрозумілий людям, що полегшує побудову моделей для машинного навчання [21]-[22].

Багато програмістів кажуть, що Python більш інтуїтивно зрозумілий, ніж інші мови програмування [21]-[22]. Інші вказують на безліч фреймворків, бібліотек та розширень, які полегшують реалізацію різних функціональних можливостей. Загальновизнано, що Python підходить для спільної реалізації, коли задіяно кілька розробників. Оскільки Python є мовою загального призначення, він може виконувати ряд складних завдань машинного навчання і дозволяє швидко створювати прототипи, що дозволяють протестувати ваш продукт для цілей машинного навчання [21]-[22].

Широкий вибір бібліотек та фреймворків. Впровадження алгоритмів штучного інтелекту та ML може бути складним і трудомістким [21]-[22]. Життєво важливо мати добре структуроване та добре перевірене середовище,

що дозволяє розробникам пропонувати найкращі рішення для кодування [21]-[22].

Щоб скоротити час розробки, програмісти звертаються до ряду фреймворків та бібліотек Python [21]-[22]. Бібліотека програмного забезпечення-це попередньо написаний код, який розробники використовують для вирішення звичайних завдань програмування. Python, з його багатим технологічним стеком, володіє великим набором бібліотек для штучного інтелекту і машинного навчання. Ось деякі з них [21]-[22]:

- Keras, TensorFlow та Scikit-learn для машинного навчання;
- NumPy для високопродуктивних наукових обчислень та аналізу даних;
- SciPy для просунутих обчислень;
- Pandas для аналізу даних загального призначення;
- Сіборн для візуалізації даних;
- Scikit-learn містить різні алгоритми класифікації, регресії та кластеризації, включаючи опорні векторні машини, випадкові ліси, градієнтне підвищення, k-means, і призначений для роботи з числовими та науковими бібліотеками Python NumPy та SciPy [21]-[22].

За допомогою цих рішень розробник зможе швидше розробляти свій продукт. А його команді розробників не доведеться винаходити нові розширення та надбудови, і вона зможе використовувати існуючу бібліотеку для реалізації необхідних функцій [21]-[22].

Кроссплатформність. Незалежність платформи відноситься до мови програмування або фреймворку, що дозволяє розробникам реалізовувати що-небудь на одній машині та використовувати їх на іншій машині без будь-яких (або мінімальних) змін [21]-[22]. Одним з ключових факторів популярності Python є те, що це незалежна від платформи мова. Python підтримується багатьма платформами, включаючи Linux, Windows та macOS [21]-[22]. Код Python може бути використаний для створення автономних виконуваних програм для більшості поширених операційних систем, що означає, що

програмне забезпечення Python можна легко розповсюджувати та використовувати в цих операційних системах без інтерпретатора Python [21]-[22].

Більше того, розробники зазвичай використовують такі сервіси, як Google або Amazon, для своїх обчислювальних потреб. Однак часто можна зустріти компанії та науковців даних, які використовують власні машини з потужними GPU для навчання своїх моделей машинного навчання [21]-[22]. І той факт, що Python не залежить від платформи, робить це навчання набагато дешевшим і простішим [21]-[22].

Велика спільнота та популярність. В опитуваннях розробників Stack Overflow 2020-2022 року Python увійшов до 5 найпопулярніших мов програмування, що в кінцевому підсумку означає, що можна знайти та найняти компанію-розробника з необхідним набором навичок для створення вашого проєкту на основі штучного інтелекту [21]-[22].

У огляді розробників Python 2020 зазначено, що Python зазвичай використовується для веб-розробки. На перший погляд, переважає веб-розробка, на яку припадає понад 26% випадків використання, показаних на зображенні нижче [21]-[22]. Однак, якщо ви поєднаєте науку про дані та машинне навчання, вони становлять приголомшливі 27% [21]-[22].

Інтернет-сховища містять понад 140000 користувацьких програмних пакетів Python. Наукові пакети Python, такі як Numpy, Scipy та Matplotlib, можуть бути встановлені в програмі, що працює на Python [21]-[22]. Ці пакети призначені для машинного навчання та допомагають розробникам виявляти закономірності у великих наборах даних. Python настільки надійний, що Google використовує його для обходу веб-сторінок, Pixar використовує його для створення фільмів, а Spotify використовує його для рекомендації пісень [21]-[22].

Загальновідомо, що спільнота штучного інтелекту Python виросла по всьому світу [21]-[22]. Існують форуми Python та активний обмін досвідом, пов'язаним із рішеннями машинного навчання. Для будь-якого завдання, яке

може виникнути у вас, існує досить велика ймовірність того, що хтось інший стикався з такою ж проблемою [21]-[22]. Кожен може знайти поради та рекомендації від розробників. Будь-хто не буде самотні і обов'язково знайдете найкраще рішення для власних конкретних потреб, якщо звернеться до спільноти Python [21]-[22].

Python як найкраща мова для розробки штучного інтелекту [21], [22]. Фільтри спаму, системи рекомендацій, пошукові системи, особисті помічники та системи виявлення шахрайства – все це стало можливим завдяки штучному інтелекту та машинному навчанню, і, безумовно, попереду ще багато чого. Власники продуктів хочуть створювати додатки, які добре працюють. Це вимагає розробки алгоритмів, які розумно обробляють інформацію, змушуючи програмне забезпечення діяти як людина [21]-[22].

Проведемо порівняння мов програмування для реалізації ПЗ, результати порівняння представимо в табличній формі (табл. 3.1). У якості критеріїв будемо використовувати наступні:

- парсинг вибірок даних – реалізація вбудованих методів (чи доступних із встановлених бібліотек) для парсингу вибірок даних;
- препроцесинг та обробка даних – наявність методів здатних швидко виконати найпростіші дії з попередньої обробки даних (фільтрація, сортування, видалення дублікатів, тощо);
- масштабування за стосунків – можливість та легкість подальшого масштабування розробленого ПЗ;
- програмні надбудови для машинного навчання – наявність та загальна оцінка фреймворків, бібліотек, плагінів, рушіїв та інших розширень для підтримки машинного навчання;
- підтримка спільноти – активність підтримки розробників учасниками спільноти, їх активність та кількість.

Тож основне порівняння зосереджувалося на можливостях використання мов програмування для машинного навчання.

Таблиця 3.1 – Порівняння мов програмування

Критерій	Python	Java	Scala	R
Парсинг вибірок даних	Високий рівень	Середній рівень	Високий рівень	Високий рівень
Препроцесинг та обробка даних	Достатньо методів	Мала кількість методів	Достатньо методів	Достатньо методів
Масштабування застосунків	Високий рівень	Високий рівень	Середній рівень	Середній рівень
Програмні надбудови для машинного навчання	Велика кількість	Недостатньо	Середня кількість	Середня кількість
Підтримка спільноти	Високий рівень	Високий рівень	Середній рівень	Середній рівень

За результатами порівняння можна зробити висновок, що найбільш з порівняння можна зробити висновок, що Python є найбільш перспективною мовою програмування для розробки за стосунків з використанням машинного навчання, адже велика кількість програмних надбудов для реалізації різних завдань машинного навчання, та вбудовані і розгорнуті через зовнішні бібліотеки методів обробки даних – забезпечують високий рівень реалізації такого ПЗ.

3.3 Вибір фреймворку

TensorFlow – це дружня до Python бібліотека з відкритим кодом для чисельних обчислень, яка прискорює та спрощує машинне навчання та розробку нейронних мереж [23]-[24].

Машинне навчання є складною дисципліною, але впровадження моделей машинного навчання набагато менш складне, ніж раніше, завдяки фреймворкам машинного навчання, таким як Google TensorFlow, які полегшують процес збору даних, навчання моделей, надання прогнозів та уточнення майбутніх результатів [23]-[24].

Створена командою Google Brain і спочатку випущена для широкої публіки в 2015 році, TensorFlow-це бібліотека з відкритим кодом для чисельних обчислень та великомасштабного машинного навчання. TensorFlow об'єднує безліч моделей та алгоритмів машинного навчання та глибокого навчання (вони ж нейронні мережі) та робить їх корисними за допомогою загальних програмних метафор [23]-[24]. Він використовує Python або JavaScript, щоб забезпечити зручний інтерфейсний API для створення додатків, одночасно виконуючи ці програми на високопродуктивному C++ [23]-[24].

TensorFlow, який конкурує з такими фреймворками, як PyTorch і Apache MXNet, може навчати і запускати глибокі нейронні мережі для класифікації рукописних цифр, розпізнавання зображень, вбудовування слів, рекурентних нейронних мереж [23]-[24], послідовних моделей для машинного перекладу, обробки природної мови та моделювання на основі рівняння з частинними похідними. Найкраще, TensorFlow підтримує масштабне прогнозування виробництва, використовуючи ті самі моделі, що використовуються для навчання [23]-[24].

TensorFlow також має велику бібліотеку попередньо навчених моделей, які можна використовувати у ваших власних проектах. Ви також можете використовувати код із Tensorflow Model Garden як приклади найкращих практик для навчання власних моделей [23]-[24].

TensorFlow дозволяє розробникам створювати графіки потоків даних-структури, які описують, як дані переміщуються по графіку або серії вузлів обробки. Кожен вузол на графіку являє собою математичну операцію, а кожне з'єднання або ребро між вузлами являє собою багатовимірний масив даних, або тензор [23]-[24].

Програми TensorFlow можна запускати практично на будь-якій зручній цілі: локальному комп'ютері, кластері в хмарі, пристроях iOS і Android, процесорах або графічних процесорах. Якщо ви використовуєте власну хмару Google, ви можете запустити TensorFlow на спеціальному

процесорі Google TensorFlow Processing Unit (TPU) silicon для подальшого прискорення. Однак отримані моделі, створені TensorFlow, можуть бути розгорнуті практично на будь-якому пристрої, де вони будуть використовуватися для надання прогнозів [23]-[24].

TensorFlow 2.0, випущений у жовтні 2019 року, значною мірою оновив фреймворк на основі відгуків користувачів, щоб полегшити роботу з ним (наприклад, використовуючи відносно простий API Keras для навчання моделей) та підвищити продуктивність [23], [24]. Розподілене навчання стало простіше запускати завдяки новому API, а підтримка TensorFlow Lite дозволяє розгорнути моделі на більшій різноманітності платформ. Однак код, написаний для попередніх версій TensorFlow, повинен бути переписаний — іноді незначно, іноді значно — щоб максимально використати нові можливості TensorFlow 2.0 [23]-[24].

Навчена модель може бути використана для надання прогнозів у вигляді Сервісу через контейнер Docker з використанням rest або gRPC API. Для більш просунутих сценаріїв обслуговування ви можете використовувати Kubernetes [23]-[24].

Використання TensorFlow з Python. TensorFlow надає все це програмісту за допомогою мови Python. Python простий у вивченні та роботі з ним, і він забезпечує зручні способи вираження того, як абстракції високого рівня можуть бути з'єднані між собою. TensorFlow підтримується у версіях Python з 3.7 по 3.10, і хоча він може працювати в попередніх версіях Python, це не гарантується [23]-[24].

Вузли та тензори в TensorFlow є об'єктами Python, а програми TensorFlow самі по собі є програмами Python [23]-[24]. Однак фактичні математичні операції не виконуються в Python. Бібліотеки перетворень, доступні через TensorFlow, написані у вигляді високопродуктивних двійкових файлів C++. Python просто спрямовує трафік між частинами та надає Абстракції програмного забезпечення високого рівня для їх з'єднання [23]-[24].

Робота на високому рівні в TensorFlow – створення вузлів і шарів і зв'язування їх разом – використовує бібліотеку Keras [23]-[24]. API Keras зовні простий; базова модель з трьома рівнями може бути визначена менш ніж за 10 рядків коду, а навчальний код для того ж самого займає всього кілька рядків коду. Але якщо треба підняти капот і виконати більш дрібнозернисту роботу, наприклад, написати власний цикл навчання, ви можете це зробити [23]-[24].

Використання TensorFlow з JavaScript. Python – найпопулярніша мова для роботи з TensorFlow та машинним навчанням загалом. Але JavaScript тепер також є першокласною мовою для TensorFlow, і однією з величезних переваг JavaScript є те, що він запускається скрізь, де є веб-браузер [23]-[24].

TensorFlow.js, як називається бібліотека JavaScript TensorFlow, використовує WebGL API для прискорення обчислень за допомогою будь-яких графічних процесорів, доступних у системі [23]-[24]. Також можна використовувати серверну частину WebAssembly для виконання, і це швидше, ніж звичайна серверна частина JavaScript, якщо ви працюєте тільки на процесорі, хоча по можливості краще використовувати графічні процесори. Готові моделі дозволяють розпочати роботу з простими проектами, щоб дати вам уявлення про те, як все працює [23]-[24].

Тензорний потік Lite. Навчені моделі TensorFlow також можуть бути розгорнуті на периферійних обчисленнях або мобільних пристроях, таких як системи iOS або Android [23], [24]. Набір інструментів TensorFlow Lite оптимізує моделі TensorFlow для ефективної роботи на таких пристроях, дозволяючи вам вибирати між розміром моделі і точністю. Менша Модель (тобто 12 МБ проти 25 МБ або навіть 100+ МБ) менш точна, але втрата точності, як правило, невелика і з лишком компенсується швидкістю та енергоефективністю моделі [23]-[24].

Навіщо використовувати TensorFlow. Єдиною найбільшою перевагою, яку TensorFlow надає для розробки машинного навчання, є абстракція [23]-[24]. Замість того, щоб розбиратися з найдрібнішими деталями реалізації

алгоритмів або з'ясовувати правильні способи прив'язки вихідних даних однієї функції до вхідних даних іншої, розробник може зосередитися на загальній логіці програми. TensorFlow піклується про деталі за лаштунками [23]-[24].

TensorFlow пропонує додаткові зручності для розробників, яким потрібно налагодити та поглибити програми TensorFlow [23]-[24]. Кожна операція графа може бути оцінена і змінена окремо і прозоро, замість того, щоб будувати весь графік як єдиний непрозорий об'єкт і оцінювати його все відразу. Цей так званий режим нетерплячого виконання, що надається як опція у старих версіях TensorFlow, тепер є стандартним [23]-[24].

Пакет візуалізації TensorBoard дозволяє перевіряти та профілювати спосіб роботи графіків за допомогою інтерактивної веб-інформаційної панелі. Сервіс Tensorboard.dev (розміщений Google) дозволяє розміщувати експерименти з машинного навчання, написані на TensorFlow, і ділитися ними. Його можна безкоштовно використовувати зі сховищем до 100 млн скалярів, 1 ГБ тензорних даних і 1 ГБ двійкових об'єктних даних [23]-[24].

TensorFlow також отримує багато переваг від підтримки комерційного підрозділу a-list в Google. Google сприяла швидкому темпу розробки проєкту і створила безліч важливих пропозицій, які спрощують розгортання і використання TensorFlow [23]-[24]. Вищезгаданий TPU-кремній для підвищення продуктивності в хмарі Google-це лише один приклад. [23]-[24]

Навчання детермінованої моделі за допомогою TensorFlow. Кілька деталей реалізації TensorFlow ускладнюють отримання повністю детермінованих результатів навчання моделі для деяких навчальних завдань. Іноді модель, навчена в одній системі, буде дещо відрізнятися від моделі, навченої в іншій, навіть якщо їм передаються однакові дані [23]-[24]. Причини такої різниці неоднозначні-одна з причин полягає в тому, як і де висіваються випадкові числа; інша пов'язана з певною недетермінованою поведінкою при використанні графічних процесорів. Гілка TensorFlow 2.0 має можливість увімкнути детермінізм у всьому робочому процесі за

допомогою декількох рядків коду. Однак ця функція вимагає витрат на продуктивність і повинна використовуватися лише при налагодженні робочого процесу [23]-[24].

TensorFlow проти PyTorch, CNTK та MXNet. TensorFlow конкурує з багатьма іншими фреймворками машинного навчання. PyTorch, CNTK та MXNet – це три основні рамки, які задовольняють багато однакових потреб. Завершимо швидким переглядом на те, де вони виділяються і не дотягують до TensorFlow [23]-[24].

PyTorch побудований на Python і має багато інших подібностей з TensorFlow: апаратно-прискорені компоненти під капотом, високоінтерактивна модель розробки, що дозволяє розробляти по ходу роботи, і безліч корисних компонентів, вже включених. PyTorch, як правило, є найкращим вибором для швидкої розробки проектів, які потрібно запустити за короткий час, але TensorFlow виграє для більших проектів та більш складних робочих процесів [23]-[24].

CNTK, Microsoft Cognitive Toolkit, схожий на TensorFlow у використанні графічної структури для опису потоку даних, але в основному фокусується на створенні нейронних мереж глибокого навчання. CNTK швидше справляється з багатьма завданнями нейронної мережі і має більш широкий набір API (Python, C++, C#, Java). Але в даний час це не так просто вивчити або розгорнути, як TensorFlow. Він також доступний лише за ліцензією GNU GPL 3.0, тоді як TensorFlow доступний за більш ліберальною ліцензією Apache. І CNTK розробляється не так агресивно; останній великий реліз був у 2019 році [23]-[24].

Проведемо порівняння фреймворків машинного навчання для реалізації ПЗ, результати порівняння представимо в табличній формі (табл. 3.2). У якості критеріїв будемо використовувати наступні:

- топології нейронних мереж – підтримка кількості та різноманітності нейронних мереж;

- організація паралельних обчислень – наявність інструментарію для організації паралельних обчислень;
- швидкість розробки – оцінка швидкості розробки за стосунку з використанням фреймворку;
- простота розгортання за стосунків – простота розгортання розроблених за стосунків;
- оновлення та підтримка – якість оновлень та оцінка активності підтримки від спільноти.

Таблиця 3.2 – Порівняння мов програмування

Критерій	TensorFlow	PyTorch	CNTK	Theano
Топології нейронних мереж	Усі сучасні топології	Обмежена кількість	Обмежена кількість	Обмежена кількість
Організація паралельних обчислень	Можливе використання CPU та GPU	Обмеження	Можливе використання CPU	Обмеження
Швидкість розробки	Висока	Висока	Середня	Середня
Простота розгортання застосунків	Просто	Середня складність	Обмежені можливості	Обмежені можливості
Оновлення та підтримка	Комплексна, середня частота	Комплексна, середня частота	Недостатня, середня частота	Низька, низька частота

За результатами порівняння можна зробити висновок, що найбільш оптимальним є TensorFlow. Цей фреймворк машинного навчання активно підтримується спільнотою розробників, а його оновлення мають саме комплексний характер. Більш того, саме цей фреймворк відрізняється найповнішою кількістю різновидів сучасних топологій штучних нейронних мереж та забезпечує автоматизацію деяких процесів з розпаралелювання обчислень.

3.4 Вибір середовища розробки

Anaconda – це дистрибутив Найважливішим компонентом Anaconda є conda, менеджер пакетів Anaconda [25]-[26].

Менеджер пакетів (у контексті Python) – це ПЗ, яке керує зовнішніми пакетами поверх вашої локальної інсталяції Python всередині коробки під назвою (віртуальне) середовище. Якщо встановлюються зовнішні пакети, вони додаються до середовища [25]-[26]. За замовчуванням користувач завжди має повний доступ до всіх пакетів, які знаходяться у його (глобальному) середовищі. Менеджер пакетів тепер створює нові середовища, що дозволяє розміщувати певні пакети лише в одному конкретному середовищі. Наприклад, можна створити середовище для веб-розробки та середовище для машинного навчання [25]-[26]. Потім потрібно активувати середовище, щоб використовувати пакети всередині нього. Буде неможливо використовувати свої пакети веб-розробки в середовищі машинного навчання [25]-[26].

Conda – саме такий менеджер пакетів. По суті, це альтернатива рір, вбудованому менеджеру пакетів Python, який поставляється з попередньо встановленою версією більшості сучасних Python [25]-[26].

Рір поставляється з попередньо встановленими версіями Python. Оскільки conda – це лише альтернатива рір, є причини як за, так і проти використання conda замість рір [25]-[26]. Один з них не є однозначно кращим за інший, вони обидва чудові, і ви не можете помилитися з будь-яким із них. З огляду на це, давайте подивимося на плюси і мінуси conda [25]-[26].

Однією з переваг conda є те, що conda може керувати віртуальними середовищами, тоді як рір не може. Насправді це не так вже й складно, оскільки інструмент venv, який також керує віртуальними середовищами, попередньо встановлений з усіма сучасними версіями Python, а також є рекомендованим способом управління віртуальними середовищами [25]-[26].

Починаючи з версії Python 3.5, документи офіційно рекомендують `venv` для управління середовищами [25]-[26].

Єдина перевага `conda` все ще полягає в тому, що можна встановлювати свої пакети та керувати своїми середовищами за допомогою того самого інструменту – `conda`, тоді як з `pip` вам доводиться використовувати зовнішню утиліту, таку як `venv`, для управління своїми середовищами [25]-[26].

Ще однією перевагою Anaconda є те, що можна використовувати так званий Anaconda Navigator, який є програмою графічного інтерфейсу для запуску популярних програм програмування або завдань, пов'язаних з наукою про дані, таких як VSCode, Jupyter або RStudio. Також можна керувати своїми середовищами `conda` всередині навігатора Anaconda [25]-[26].

Третя перевага Anaconda полягає в тому, що за замовчуванням `conda` постачається з багатьма попередньо встановленими пакетами машинного навчання. Conda постачається з NumPy, Matplotlib, Pandas, Scikit-Learn та багатьма іншими поза коробкою [25]-[26]. Це означає, що не потрібно встановлювати ці пакети вручну; натомість можна просто встановити Anaconda і бути готовими розпочати роботу над наступним проектом з науки про дані або машинного навчання. Якщо немає необхідності мати всі ці додаткові утиліти та пакети, можна просто встановити Miniconda, яка є мінімалістичною версією Anaconda, яка постачається без Anaconda Navigator і не вимагає попередньої установки такої кількості пакетів у вашу систему [25]-[26].

Четверта і остання перевага `conda` полягає в тому, що це рекомендований спосіб встановлення та використання Spyder [25]-[26]. Spyder – це середовище розробки, яке було розроблено спеціально для цілей машинного навчання та науки про дані і використовується багатьма людьми, які працюють у цих галузях, хоча воно не настільки популярне, як сучасні альтернативи, такі як Vscode [25]-[26].

У таблиці 3.3 наведено порівняння найбільш популярних середовищ розробки для Python. У якості критеріїв будемо використовувати наступні:

- доступність – вільне чи платне розповсюдження ПЗ;
- модулі та надбудови для машинного навчання – наявність та простота встановлення/налаштування модулів, плагінів та надбудов для машинного навчання;
- спільнота та підтримка оновлень – як часто виходять оновлення середовища та наскільки активна підтримка спільноти розробників;
- робота із засобами TensorFlow – наскільки просто налагодити роботу середовища із фреймворком TensorFlow.

Таблиця 3.3 – Порівняння середовищ розробки для Python

Критерії	Anaconda	PyCharm	Atom	PhpStorm
Доступність	Безкоштовна	Платний	Безкоштовне	Платний
Модулі та надбудови для машинного навчання	Присутні (деякі необхідно встановлював окремо)	Доступні, відрізняються за вартістю	Присутні, ускладнене встановлення	Доступні, відрізняються за вартістю
Спільнота та підтримка оновлень	Активно оновлюється, активна спільнота	Активно оновлюється, активна спільнота	Оновлюється поступово, спільнота малоактивна	Активно оновлюється, активна спільнота
Робота із засобами TensorFlow	Налагоджена	Налагоджена	Ускладнена	Налагоджена

За результатами порівняння можна прийти до висновку, що Anaconda хороший вибір у якості основного середовища розробки, адже наявна більшість налаштувань для розробки за стосунків машинного навчання та є активна підтримка засобів фреймворку TensorFlow.

3.5 Проектування структури програмної системи

При проектуванні структури майбутньої програми всю структуру можна звести до 4 основних рівнів майбутнього ПЗ, що продемонстровано на рис. 3.1. У якості апаратної платформи було обрано саме процесор – CPU.

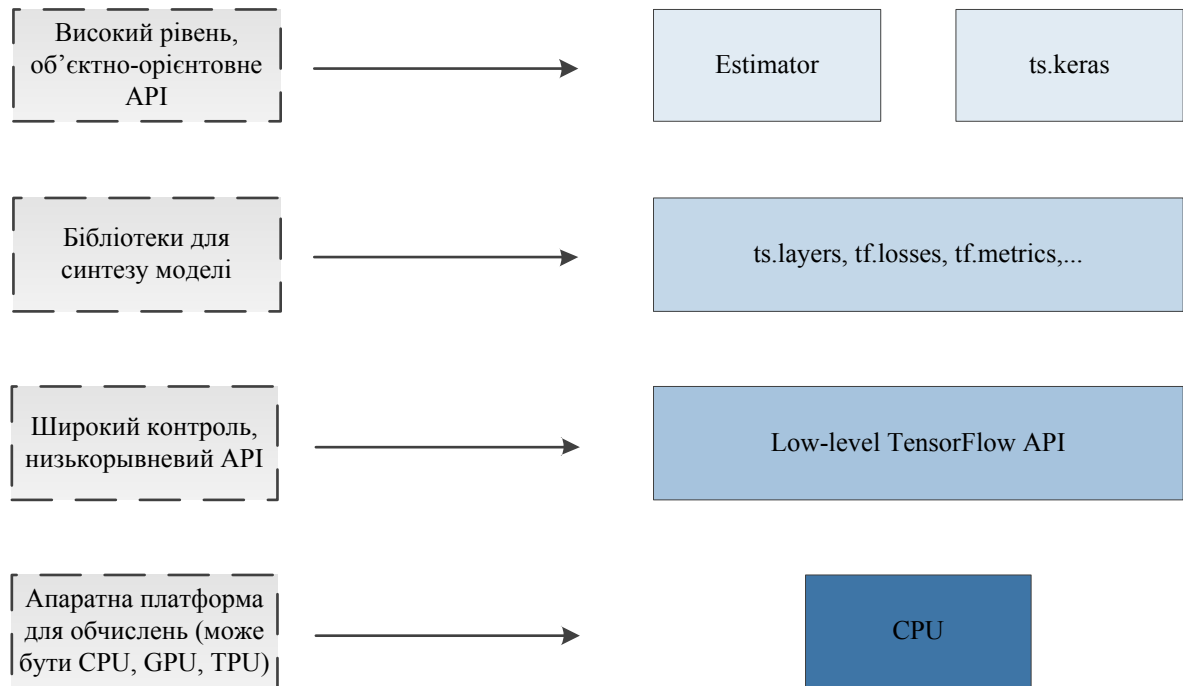


Рисунок 3.1 – Рівні розроблюваного ПЗ

Розробку ж в контексті описаних рівнів можна аргументувати способом розподілення основного функціоналу ПЗ. Згідно з обраним принципом в рамках блоків програми формуються функціонально завершені модулі. Тож такий варіант розподілення представлено на рис. 3.2.

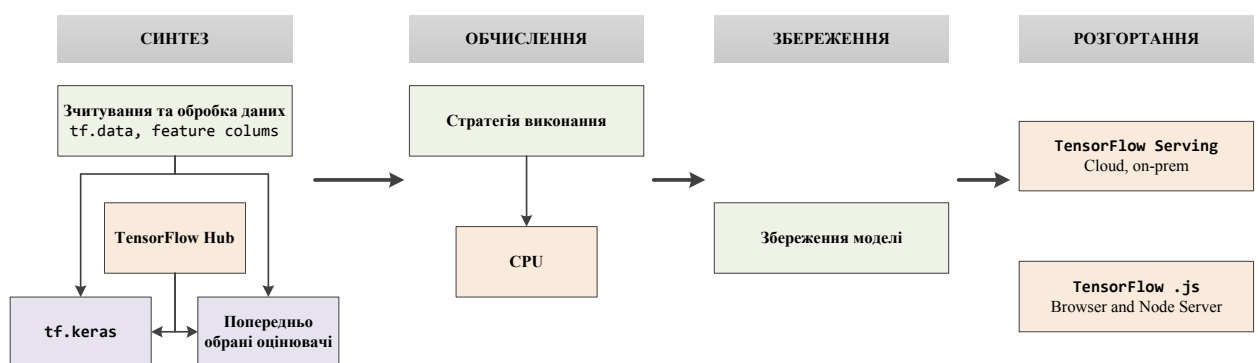


Рисунок 3.2 – Модулі розроблюваного ПЗ

3.6 Вимоги до технічних засобів

Для нормальної розробки програми, повинні виконуватися наступні умови до апаратної частини:

- наявність персонального комп'ютера (ПК), на якому буде працювати застосунок;
- CPU із тактовою частотою 3,0 GHz та більше;
- оперативна пам'ять (ОЗУ) 4,0 ГБ та більше;
- вільний простір на жорсткому диску (формат HDD або SSD – не принциповий) не менше, ніж 1 ГБ (для збереження вибірок – можливо розширення).

Вимоги до програмної частини:

- встановлений та розгорнутий дистрибутив Python.

3.7 Висновки за розділом 3

Під час роботи над розділом було спроектовано рівневу архітектуру та структуру майбутнього ПЗ. Були розроблені структуру рівнів та модулі майбутнього застосунку. Було проаналізовано найбільш популярні мови програмування та після ретельного порівняння перевага була віддана Python, а в якості фреймворку машинного навчання було обрано TensorFlow.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У третьому розділі було досліджено питання проєктування ПЗ, що буде розроблятися з метою повної реалізації запропонованого методу та можливих модифікацій. Особливу увагу було присвячено вибору технічних засобів розробки.

Четвертий розділ буде присвячено висвітленню основних рішень щодо реалізації ПЗ. Будуть проаналізовані розроблені модулі та загальна архітектура ПЗ.

4.1 Представлення основних рішень розробки

На рисунку 4.1 представлено модульний розподіл розробленого застосунку.

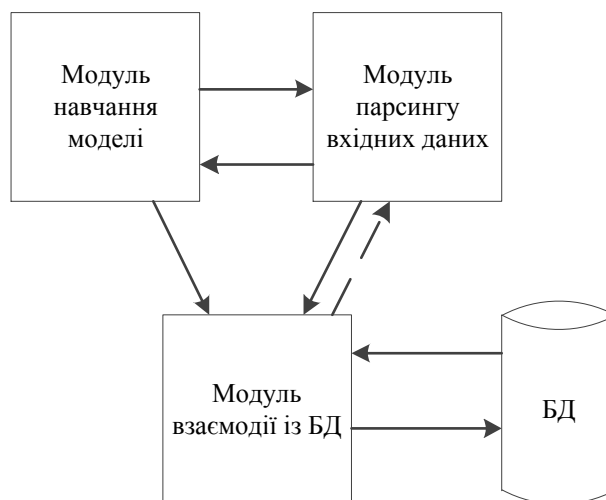


Рисунок 4.1 – Модульне представлення розробленого застосунку

На рисунку 4.2 представлено діаграму класів в нотації UML основних класів у модулі, що присвячено машинному навчанню моделі. Стрілки з порожнистими трикутними наконечниками представляють успадкування, а чорні заповнені ромбоподібні наконечники представляють відношення

Шаблон дизайну композиції був використаний для шарів верхнього і нижнього рівнів. Складні класи отримують як параметри для ініціалізації простіші шари і, таким чином, асоціюються з ними на діаграмі UML. Класи вищого рівня успадковуються від базового класу модуля, такі як Lissom, LGN та ReducedLissom, і пов'язані з лінійними модулями та класом PiecewiseSigmoid, який реалізує активацію Лісома.

Наприклад, ReducedLissom – це назва, дана творцями LISSOM мережі V1, який має аферентний, збудливий і гальмівний модулі, які є класами мережі. А клас Lissom – це V1 плюс дві карти LGN, що представляють канали зорових шарів, включені та вимкнені. На малюнку також показано клас OrientationMap та клас Pipeline.

На наступному рисунку 4.3 є класами dataset, configuration і optimizers.

Читаючи зліва, видно абстрактний клас Dataset, що представляє вхідні дані моделей нейронної мережі. Dataset є ітеративним у фреймворку, тому він не має жодних атрибутів чи функцій. Класам OrientedGaussians та ThreeDotFaces надаються набори даних та інші утиліти, якими я користуюся, які дослідник Лісома вважатиме корисними як відправну точку, представляючи загальні стимули для мережі Лісома.

Поруч праворуч видно класи, які вирішують проблему конфігурації Лісома. EvalConfigObj та EvalConfigYaml – це два класи, надаються кінцевому користувачеві, розширюючи бібліотеки yaml та ConfigObj, і вони використовуються за замовчуванням у пакеті консервованих моделей, який я також надаю.

І, нарешті, класи оптимізатора реалізують навчання Лісома. Абстрактний базовий клас Optimizer – це інтерфейс для навчання на нейронному рівні. Клас CortexHebbian є найбільш важливим, він реалізує навчання Лісома за правилом Хебба для набору нейронних зв'язків, таких як аферентні, збуджуючі або гальмівні.

Отже, за допомогою трьох CortexHebbian можна отримати оптимізатор ReducedLissom (V1), який називається ReducedLissomHebbian.

ConnectionDeath та Neighboursdecay – це інші оптимізатори, які я надаю, які є загальними для LISSOM.

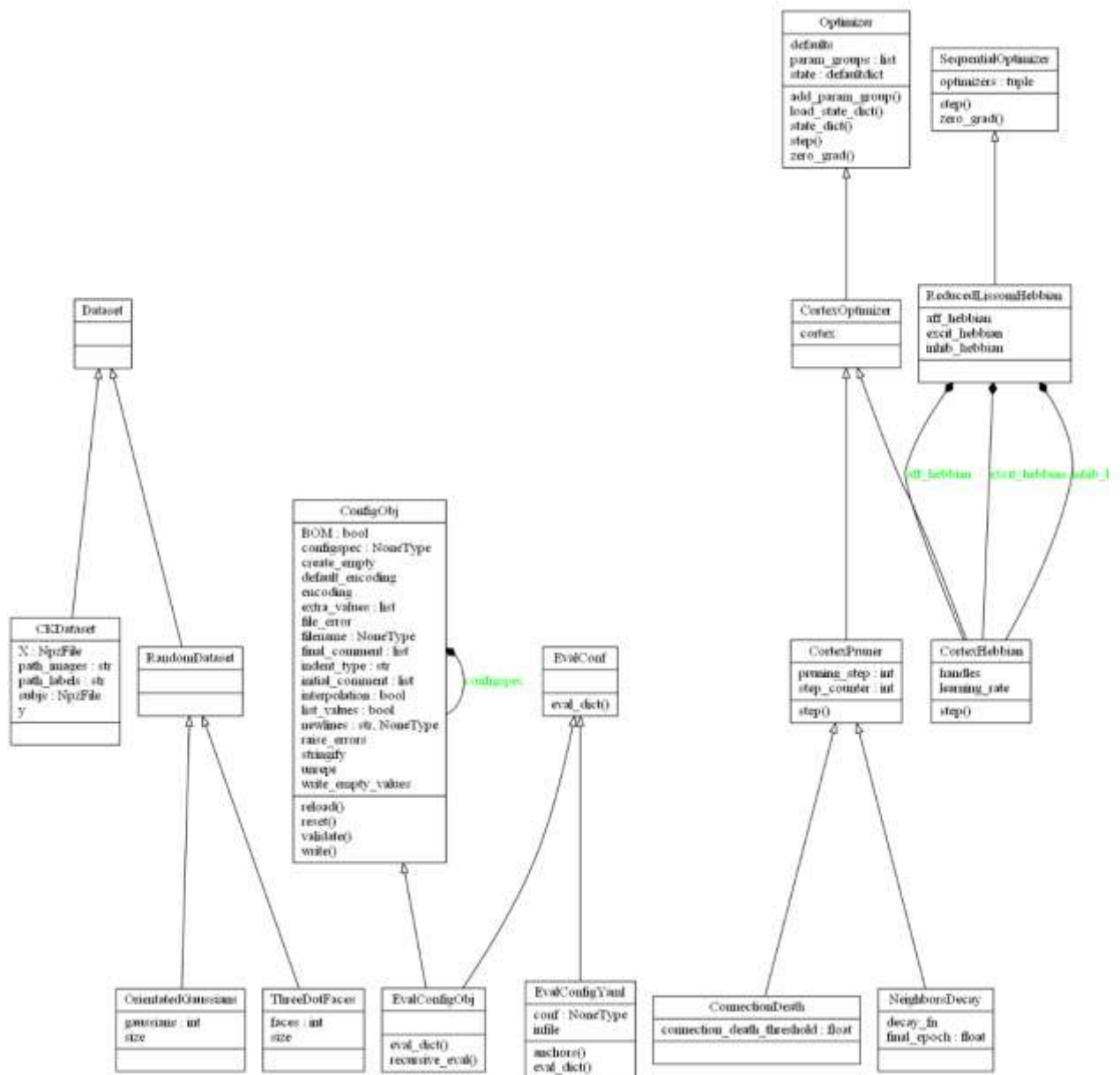


Рисунок 4.3 – UML діаграма класів модулю парсингу даних та початкових налаштувань

На рисунку 4.4 представлено загальний варіант комунікації між модулями МНМ під час роботи. Так виділяється головний модуль який відповідає за розподіл операцій та подальшу синхронізацію їх результатів.

На рисунку 4.5 наведено схематичний приклад синтезованих моделей.

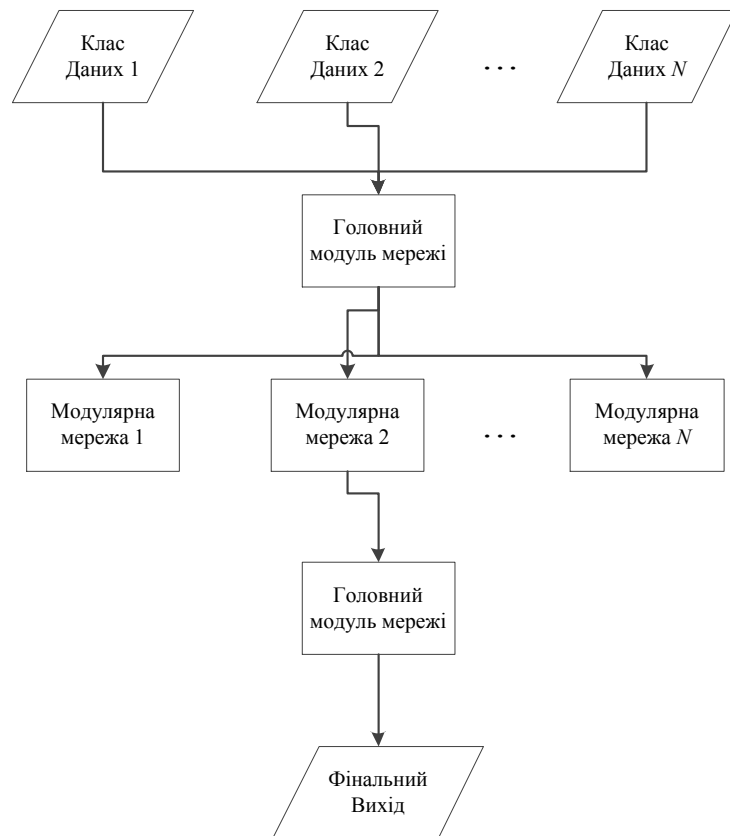


Рисунок 4.4 – Загальне представлення комунікації між модулями МНМ під час роботи

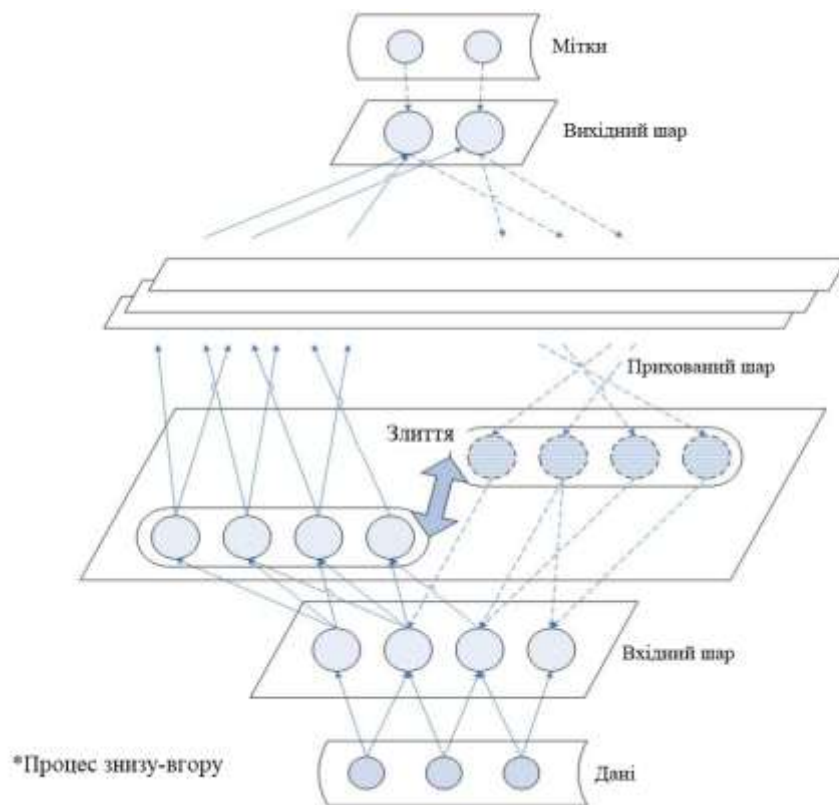


Рисунок 4.5 – Схематичний приклад синтезованих мереж

4.2 Висновки за розділом 4

Розроблено ПЗ для реалізації побудови та навчання систем на базі МНМ з метою розпізнавання зображень. Обрано програмні механізми реалізації відповідних функціональних вимог до за стосунку у вигляді методів та функцій. Представлена архітектура ПЗ засвідчила прийнятні результати масштабування та розгортання на ПК.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У попередніх розділах було розглянуто питання розроблення за стосунку, що реалізує новий метод класифікації зображень на базі МНМ та навчання з підкріпленням.

Цей розділ буде присвячено опису тестування розробленого ПЗ. Буде описано метрики для оцінки точності класифікації. Окремо буде проаналізовано отримані результати.

5.1 Вхідні дані для тестування

У експериментах було використано кілька наборів даних зображень. CEFAR [27]-[29], SVHN [30] та EMNIST [31], які містять зображення з центром та фіксованим розміром, лише з одним об'єктом на кожному зображенні. Зображення у великих наборах даних, таких як ImageNet 2012 [32] та Caltech-256 [33], мають різні розміри та точніше відображають реальні зображення. Приклади зображень з цих наборів даних наведені на рисунку 5.1.



Рисунок 5.1 – Приклади та порівняння різниць зображень з різних вибірок
даних

Набори даних CEFAR [27]–[29] складаються з кольорових зображень розміром 32×32 різних категорій. Навчальний і тестовий набори містять 50

тис. і 10 тис. зображень відповідно. Під час експериментів було дотримано загальноприйнятої практики використання 5000 зображень з навчального набору для формування перевірного набору [27]-[29]. Набір даних SVHN містить 73257 кольорових зображень розміром 32×32 пікселі в навчальному наборі та 531131 зображення для додаткового навчання [30]. Повідомляючи про результати набору даних SVHN, було дотримано загальноприйнятої практики використання всіх навчальних даних без будь-якого збільшення даних. Набір із 6000 зображень використовується для перевірки результатів навчання. EMNIST – це розширення популярного набору даних MNIST [31]. Існує шість конфігурацій набору даних EMNIST, і було використано конфігурацію, збалансовану EMNIST. Вона містить 131600 зображень у відтінках сірого розміром 28×28 пікселів, що належать до 47 категорій. Навчальний набір ImageNet містить 1000 категорій приблизно по 1000 зображень у кожній [33]. ImageNet також містить набір для перевірки та набір для тестування. Також було використано підмножину набору даних ImageNet з 20 категоріями, щоб легко візуалізувати МНМ та повністю зрозуміти деталі та властивості ієрархії. Набір даних Caltech також використовується в експериментах [32]. Як було запропоновано в [32], було використано підмножину з 11 категорій з набору даних Caltech, щоб забезпечити справедливе порівняння з існуючою роботою. Кожна категорія набору даних Caltech містить приблизно 100 навчальних зображень та 20 тестових зображень. Параметри наборів даних описані в таблиці 5.1.

Таблиця 5.1 – Детальна інформація про вибірки даних для тестування

Вибірка	Розмір зображень	Кількість зображень для навчання	Кількість зображень для тестування	Кількість класів зображень
1	2	3	4	5
CIFAR 10	$32 \times 32 \times 3$	50000	10000	10
CIFAR 100	$32 \times 32 \times 3$	50000.	10000	100

Продовження таблиці 5.1

1	2	3	4	5
SVHN	$32 \times 32 \times 3$	604388	26032	10
EMNIST	$28 \times 28 \times 1$	112800	18800	47
ImageNet 2012	Варіюється	1200000	75000	1000
ImageNet 2012 (підвибірка)	Варіюється	26000	2000	20
Caltech (підвибірка)	Варіюється	2000	400	11

5.2 Метрики оцінювання

Моделі класифікації зображень повинні бути оцінені, щоб визначити, наскільки добре вони працюють порівняно з іншими моделями [7]-[8]. Ось деякі добре відомі показники, які використовуються при класифікації зображень [7]-[8].

Точність. Точність – це показник, який визначається для кожного класу. Точність у класі говорить нам, яка частка даних, передбачених моделлю машинного навчання для приналежності до класу, насправді була частиною класу в даних перевірки. Проста формула (5.1) може продемонструвати це [7]-[8]:

$$Precision = \frac{TP}{TP + FP}, \quad (5.1)$$

де TP – представляє справжні позитивні результати: кількість зразків, які, як було передбачено, належать до класу і правильно належать до класу;

FP – представляє хибнопозитивні результати: кількість зразків, які, як було передбачено, належать до класу, тоді як вони взагалі не належать до класу.

Повнота. Повнота, подібне до точності, визначається для кожного класу [7]-[8].

Повнота повідомляє нам, яка частка даних із набору перевірки, що належать до класу, була правильно визначена (як належна до класу) [7]-[8].

Повнота може бути представлена у вигляді [7]-[8], формула (5.2):

$$Recall = \frac{TP}{TP + FN}, \quad (5.2)$$

де, TP – представляє справжні позитивні результати: кількість зразків, які, як було передбачено, належать до класу і правильно належать до класу;

FN – представляє хибнонегативні результати: кількість зразків, які модель передбачала як такі, що не належать до класу, тоді як вони фактично належать до цього конкретного класу [7]-[8].

Оцінка F1. Оцінка F1 допомагає досягти балансу між точністю та повнотою, щоб отримати середнє уявлення про те, як працює модель [7], [8]. Оцінка F1 як показник розраховується наступним чином за формулою (5.3):

$$F1\ Score = \frac{2 \cdot (Precision \times Recall)}{Precision + Recall}. \quad (5.3)$$

Відносна помилка. Відносна помилка в цьому випадку буде обчислюватися за формулою (5.4), як відношення помилки класифікування до загального розміру вибірки (кількості її екземплярів):

$$E = \frac{error_{class}}{Number_{sampl}} \cdot 100\%, \quad (5.4)$$

де E – відносна помилка;

$error_{class}$ – помилка класифікування;

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

5.3 Результати тестування

Порівняння результатів тестування МНМ із іншими нейронними мережами під час класифікації обраних вибірок даних наведено в таблиці 5.2.

Таблиця 5.2 – Загальні результати тестування

Dataset	Мережа	Розмір моделі в пам'яті (KB)	Кількість ітерацій	Помилка моделі на тестових даних
CIFAR-10	VGG-16	78,410	313 М	0,067
	VGG-Pruned	28,200	206 М	0,066
	MHM	806	28 М	0,079
CIFAR-100	VGG-16	78,590	207 М	0,295
	VGG-Pruned	28,910	210 М	0,252
	ResNet	141,100	25,800 М	0,192
	MHM	832	34 М	0,209
SVHN	ResNet	11,000	1,935 М	0,016
	MHM	522	30 М	0,018
	EDEN	—	—	0,117
EMNIST	MHM	363	4М	0,078
	VGG-16	528,000	15,300 М	0,076
ImageNet 2012 (subset)	ResNet	84,000	3,640 М	0,081
	MHM	1,872	605 М	0,124
	VGG-16	528,120	15,300 М	0,295
ImageNet 2012	ResNet-34	84,100	3,640 М	0,276
	MHM	2,515	713 М	0,313

Конкуруючі моделі були використані зі стандартних модулів фреймворку TensorFlow [23]-[24].

5.4 Аналіз результатів

Було порівняно архітектуру МНМ з кількома існуючими архітектурами.

Було доведено, що МНМ має найменші вимоги до пам'яті і кількості ітерацій. Цей приріст продуктивності досягається незначними витратами на помилку тестування. ResNet і VGG містять 54 і 16 шарів відповідно. В деяких тестах було рекомендовано використовувати також скорочену та квантову версією архітектури VGG: VGG-Pruned. Також порівнялася МНМ з ResNet. Для набору даних EMNIST ми використовуємо EDEN [88] для порівняння.

Ці архітектури містять великі глибокі нейронні мережі з перевернутими фільтрами вузьких місць для зменшення кількості операцій.

У таблиці 5.2 видно, що дерево МНМ має найменший розмір моделі. У порівнянні з VGG-Pruned на CIFAR – 100, для МНМ потрібна модель на 97,12% менше. Моделі меншого розміру вимагають меншої кількості звернень до пам'яті, досягають більш швидкої збіжності і зниження енергоспоживання. У таблиці показано кількість множень і додавань з плаваючою комою, виконаних під час виведення одного зображення. Зазначений розмір моделі та кількість операцій МНМ – це сума значень модулів уздовж найдовшого шляху виконання.

Існує незначна різниця у вимогах до пам'яті та кількості операцій при порівнянні МНМ для ImageNet 2012 (підмножина) та всього набору даних ImageNet 2012. Це показує масштабованість запропонованої методики при побудові з використанням методів, представлених у розділі 1. Таблиця не повідомляє розмір моделі та кількість операцій для EDEN, оскільки дані та вихідний код не є загальнодоступними.

З таблиці 5.2 видно, що МНМ досягає найменшої помилки 7,8% для набору даних EMNIST. Точність МНМ порівнянна з рівнем техніки для наборів даних CIFAR-10 та SVHN. Варто зазначити, що помилка тестування, досягнута в сучасних монолітних архітектурах глибоких нейромереж, досягається після значних зусиль з налаштування гіперпараметрів.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи магістра, було запропоновано новий метод класифікації зображень на основі моделі на базі МНМ, що навчається використовуючи підход машинного навчання з підкріпленням.

Для цього були поставлені і вирішені такі задачі:

- ознайомлення та аналіз проблемної області класифікації зображень;
- розгляд мови програмування Python, фреймворку машинного навчання TensorFlow та середовищ розробки для цих мови та фреймворку;
- розроблення ПЗ для реалізації та тестування запропонованого методу.

Розроблений програмний комплекс, реалізує основні функції, а саме:

- авторизація (верифікація) користувача;
- підвантаження вибірок вхідних даних;
- пересилка даних на серверну частину;
- виконання побудови та навчання МНМ;
- представлення результатів;
- збереження результатів роботи (реалізація експорту та імпорту даних про результати обчислень).

Запропонований метод використовує найсучасніші засади навчання з підкріпленням та МНМ у якості базису для моделей.

Розроблений програмний комплекс має практичну спрямованість та готовий до використання.

Під час розробки програми та аналізу розробленого програмного комплексу з'явилися ідеї з удосконалення, такі як: використання паралельних потужностей обчислювальних систем на базі GPU для організації на їх базі безпосередньо модульних нейромереж.

Нові можливості зможуть розширити коло вирішення практичних задач методом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kuo, C. Transfer Learning for Image Classification: With Python Examples / C. Kuo. – New York : Kindle, 2022. – 127 p.
2. Borra, S. Satellite Image Analysis: Clustering and Classification (SpringerBriefs in Applied Sciences and Technology) / S. Borra, R. Thanki, N. Dey. – Cham : Springer, 2019. – 113 p.
3. Siahaan, V. Step by Step Tutorial image classification Using Scikit-Learn, Keras, And TensorFlow with Python GUI / V. Siahaan. – New York : Kindle, 2021. – 141 p.
4. Canty, M.J. Image Analysis, Classification and Change Detection in Remote Sensing: With Algorithms for Python / M. J. Canty. – New York: CRC Press, 2019. – 530 p.
5. Image Classification [Electronic resource]. – Access mode: <https://www.sciencedirect.com/topics/engineering/image-classification>.
6. Image Classification Explained [+V7 Tutorial] [Electronic resource]. – Access mode: <https://www.v7labs.com/blog/image-classification-guide#h4>.
7. Image Classification in AI: How it works [Electronic resource]. – Access mode: <https://levity.ai/blog/image-classification-in-ai-how-it-works>.
8. Image Classification [Electronic resource]. – Access mode: <https://huggingface.co/tasks/image-classification>.
9. Dayhoff J. Neural Network Architectures: An Introduction / J. Dayhoff. – New York : Van Nostrand Reinhold Company, 1990. – 259 p.
10. Aggarwal C.C. Neural Networks and Deep Learning: A Textbook / C.C. Aggarwal. – Cham : Springer, 2018. – 520 p.
11. Rashid T. Make Your Own Neural Network / T. Rashid. – Scotts Valley : CreateSpace Independent Publishing Platform, 2016. – 222 p.
12. Jones, H. Neural Networks: An Essential Beginners Guide to Artificial Neural Networks and their Role in Machine Learning and Artificial

Intelligence / H. Jones. – Scotts Valley : CreateSpace Independent Publishing Platform, 2018. – 76 p.

13. Theobald, O. Machine Learning for Absolute Beginners: A Plain English Introduction (Machine Learning with Python for Beginners) / O. Theobald. – London : Scatterplot Press, 2020. – 181 p.

14. Sebastian, C. Machine Learning for Beginners: Absolute Beginners Guide, Learn Machine Learning and Artificial Intelligence from Scratch / C. Sebastian. – New York : Kindle, 2018. – 102 p.

15. Julian, D. Designing Machine Learning Systems with Python / D. Julian. – Birmingham : Packt Publishing, 2016. – 232 p.

16. Burkov, A. The Hundred-Page Machine Learning Book / A. Burkov. – New York : Kindle, 2022. – 160 p.

17. Modular Neural Networks for Low-Power Image Classification on Embedded Devices [Electronic resource]. – Access mode: <https://dl.acm.org/doi/10.1145/3408062>.

18. Reasoning Using Modular Neural Networks [Electronic resource]. – Access mode: <https://towardsdatascience.com/reasoning-using-modular-neural-networks-f003cb6109a2>.

19. What is a Modular Neural Network? Weak vs. Strong Learners [Electronic resource]. – Access mode: <https://caseguard.com/articles/what-is-a-modular-neural-network-weak-vs-strong-learners/>.

20. Modular neural networks using multiple gradients [Electronic resource]. – Access mode: <http://essay.utwente.nl/82225/>.

21. Machine Learning with Python [Electronic resource]. – Access mode: <https://www.freecodecamp.org/learn/machine-learning-with-python/>.

22. Python for Machine Learning [Electronic resource]. – Access mode: <https://machinelearningmastery.com/python-for-machine-learning/>.

23. Machine learning education [Electronic resource]. – Access mode: <https://www.tensorflow.org/resources/learn-ml>.

24. Machine Learning with TensorFlow [Electronic resource]. – Access mode: <https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit>.
25. The Only Post you Need to Start Using Anaconda [Electronic resource]. – Access mode: https://machinelearningcompass.com/development_environment/anaconda/
26. Anaconda [Electronic resource]. – Access mode: <https://www.anaconda.com/products/individual>.
27. The CIFAR-10 dataset [Electronic resource]. – Access mode: <https://www.cs.toronto.edu/~kriz/cifar.html>.
28. The CIFAR-100 dataset [Electronic resource]. – Access mode: <https://www.cs.toronto.edu/~kriz/cifar.html>.
29. CIFAR-100 [Electronic resource]. – Access mode: <https://paperswithcode.com/dataset/cifar-100>.
30. The Street View House Numbers (SVHN) Dataset [Electronic resource]. – Access mode: <http://ufldl.stanford.edu/housenumbers/>.
31. The EMNIST Dataset [Electronic resource]. – Access mode: <https://www.nist.gov/itl/products-and-services/emnist-dataset>.
32. ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [Electronic resource]. – Access mode: <https://www.image-net.org/challenges/LSVRC/2012/>.
33. Caltech 256 Image Dataset [Electronic resource]. – Access mode: <https://www.kaggle.com/datasets/jessicali9530/caltech256>.
34. Мошек, Г.Є. Менеджмент організації. Теорія та практика [Текст] : навч. посіб. XXI ст. / [Г. Є. Мошек, В. Л. Федоренко, А. С. Соломко ; за заг. ред. Мошека Г. Є.]; Національна академія державного управління при Президентові України, Науково-дослідному центрі судової експертизи з питань інтелектуальної власності Міністерства юстиції України. – Київ : Ліра-К, 2020. – 419 с.

35. Єрмошенко, М.М. Інформаційні технології аналізу даних у маркетингу [Текст] : навч. посіб. / Єрмошенко М. М., Нестеренко О. В., Штулер І. Ю.; Національна академія державного управління. – Київ: Національна академія управління, 2021. – 141 с.

ДОДАТОК А
Технічне завдання

A.1 Вступ

A.1.2 Призначення та область застосування

Розроблений ПЗ може використовуватися для реалізації машинного навчання з підкріпленням на базі МНМ. Така система може використовуватися системах класифікації та розпізнавання зображень в технічних, медичних та військових цілях.

Програма представляє графічний інтерфейс користувача для зручної роботи із реалізованими функціями.

A.2 Підстави для розробки

Підставою для розробки є завдання на магістерську роботу, яке було визначене за темою «Дослідження та програмна реалізація методів класифікації зображень на основі навчання з підкріпленням» і затверджене наказом №363 від 04 листопада 2022 р. за Національним університетом «Запорізька політехніка».

A.3 Призначення розробки

Розроблений ПЗ може використовуватися для автоматизації класифікації зображень.

Розроблений застосунок має реалізовувати основні функції, а саме:

- авторизація (верифікація) користувача;
- підвантаження вибірок вхідних даних;
- пересилка даних на серверну частину;
- виконання побудови та навчання МНМ;
- представлення результатів;
- збереження результатів роботи (реалізація експорту та імпорту даних про результати обчислень).

А.4 Вимоги до програми

А.4.1 Вимоги до функціональних характеристик

ПЗ повинне забезпечувати виконання нижче перерахованих функцій:

- авторизація (верифікація) користувача;
- підвантаження вибірок вхідних даних;
- пересилка даних на серверну частину;
- виконання побудови та навчання МНМ;
- представлення результатів;
- збереження результатів роботи (реалізація експорту та імпорту даних про результати обчислень).

А.4.2 Вимоги до надійності

Для забезпечення надійності програми, її складові частини мають бути розгорнуті між серверною станцією та персональним комп'ютером (ПК) користувача. Протягом сеансу роботи застосунок не повинен збоїти або зависати. Після внесення оновлень за стосунок має перезавантажитися та знову бути доступним.

БД яка формується на основі даних за стосунку та проміжних результатів його роботи має відповідати наступним умовам:

- зміни до бази вносяться тільки від верифікованих користувачів;
- напряму із БД може контактувати виключно серверна частина ПЗ або адміністратор;
- оновлення БД виконується виключно на основі внутрішніх правил.

А.4.3 Вимоги до технічних та програмних засобів

Для нормальної розробки програми, повинні виконуватися наступні умови до апаратної частини:

- наявність ПК, на якому буде працювати застосунок;
- CPU із тактовою частотою 3,0 GHz та більше;
- ОЗУ 4,0 ГБ та більше;
- вільний простір на жорсткому диску (формат HDD або SSD – не принциповий) не менше, ніж 1 ГБ (для збереження вибірок – можливо розширення).

Вимоги до програмної частини:

- встановлений та розгорнутий дистрибутив Python.

A.5 Вимоги до програмної документації

Склад програмної документації повинен включати в себе [34]-[35]:

- 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 [34]-[35].

На стадії робочого проєктування повинні бути виконані перераховані нижче етапи робіт [34]-[35]:

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

На стадії впровадження повинен бути виконаний етап розробки підготовка і передача програми [34]-[35].

A.6.3 Зміст робіт по етапах

На етапі розробки технічного завдання повинні бути виконані перераховані нижче роботи [34], [35]:

- постановка задачі;
- визначення і уточнення вимог до технічних засобів [34]-[35];
- визначення вимог до програми;
- визначення стадій, етапів і термінів розробки програми та документації на неї [34]-[35];
- узгодження і затвердження технічного завдання.

На етапі розробки програми повинна бути виконана робота з програмування (кодування) і налагодженні програми [34]-[35].

На етапі розробки програмної документації повинна бути виконана розробка програмних документів відповідно до вимог до складу документації [34], [35].

На етапі випробувань програми повинні бути виконані перераховані нижче види робіт [34]-[35]:

- розробка, узгодження та затвердження та методики випробувань [34]-[35];
- проведення приймально-здавальних випробувань; коректування програми і програмної документації за результатами випробувань [34]-[35].

ДОДАТОК Б
Опис програми

Б.1 Загальні свідчення

ПЗ розроблено з використанням мови програмування Python, у інтегрованому середовищі розробки Anaconda та на базі засобів фреймворку машинного навчання – TensorFlow. Тобто для реалізації основної бізнес-логіки застосунку використовується мова програмування Python.

Б.2 Функціональне призначення

ПЗ повинне забезпечувати виконання нижче перерахованих функцій:

- авторизація (верифікація) користувача;
- підвантаження вибірок вхідних даних;
- пересилка даних на серверну частину;
- виконання побудови та навчання МНМ;
- представлення результатів;
- збереження результатів роботи (реалізація експорту та імпорту даних про результати обчислень).

Б.3 Опис логічної структури

При проектуванні структури майбутньої програми всю структуру можна звести до 4 основних рівнів майбутнього ПЗ, що продемонстровано на рис. Б.1. У якості апаратної платформи було обрано саме процесор – CPU.

Розробку ж в контексті описаних рівнів можна аргументувати способом розподілення основного функціоналу ПЗ. Згідно з обраним принципом в рамках блоків програми формуються функціонально завершені модулі. Тож такий варіант розподілення представлено на рис. Б.2.

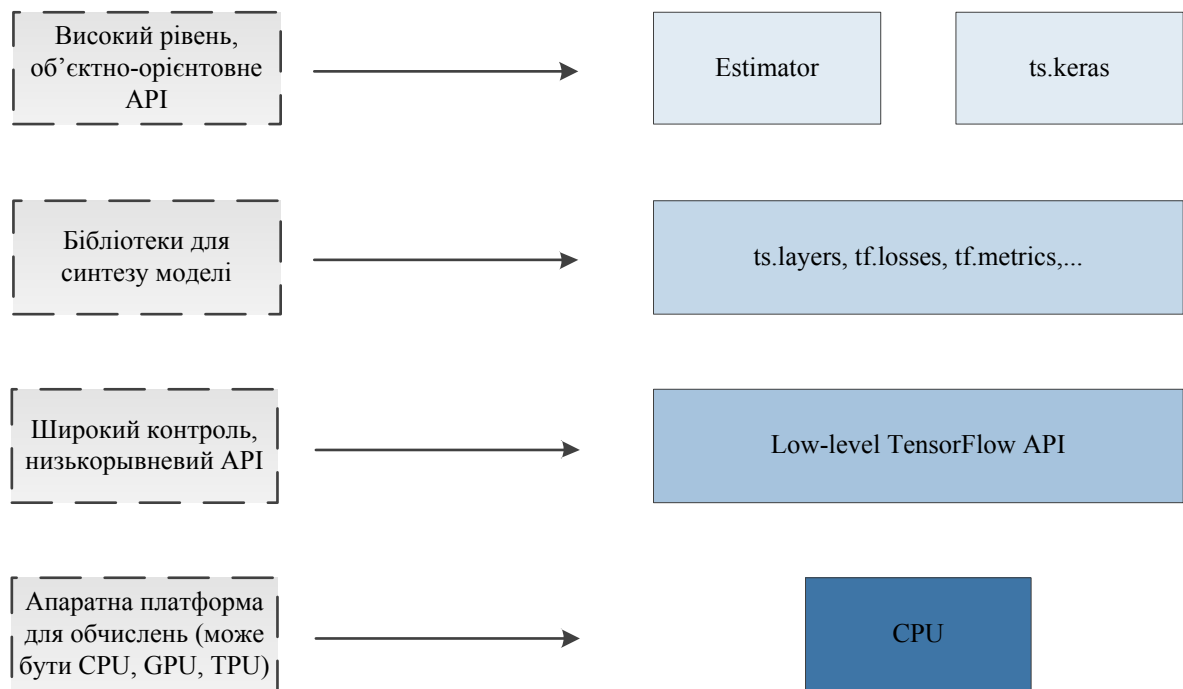


Рисунок Б.1 – Рівні розроблюваного ПЗ

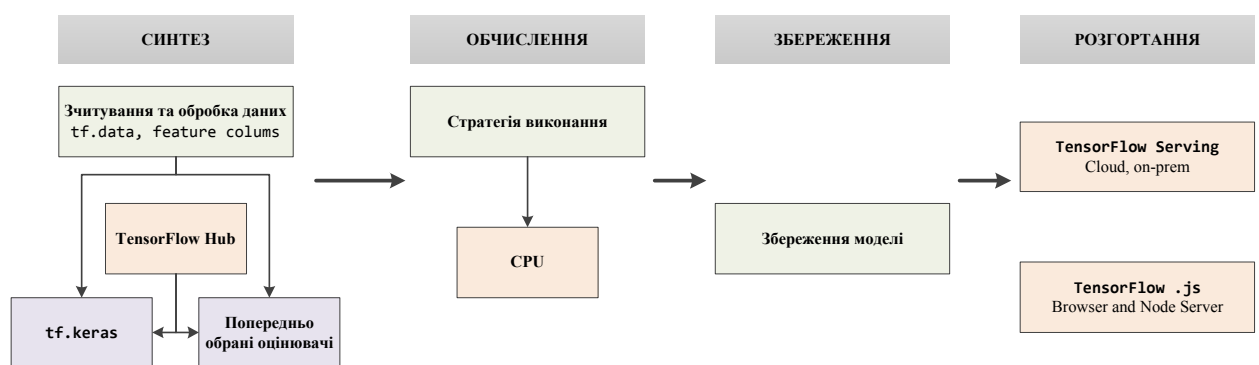


Рисунок Б.2 – Модулі розроблюваного ПЗ

Б.4 Використані технічні засоби

Для нормальної розробки програми, повинні виконуватися наступні умови до апаратної частини:

- наявність ПК, на якому буде працювати застосунок;
- CPU із тактовою частотою 3,0 GHz та більше;
- ОЗУ 4,0 ГБ та більше;

- вільний простір на жорсткому диску (формат HDD або SSD – не принциповий) не менше, ніж 1 ГБ (для збереження вибірок – можливо розширення).

Вимоги до програмної частини:

- встановлений та розгорнутий дистрибутив Python.

ДОДАТОК В
Текст програми

Сирцевий код розробленого застосунку:

```
import numpy as np
import random
import time

class ANN:

    def __init__(self, PolTopo, training, testing, maxmTime, minmPer):
        self.PolTop = PolTopo # NN topology [inp, hddn, otpt]
        self.maxm = maxmTime # max epocs
        self.trainingSampDat = training
        self.testingSampDat = testing
        self.NumDTSamps = training.form[0]

        self.ANN_rate = 0

        self.temp_ANNRate = 0
        self.useKapelMomen = 0 #use
        self.minRetfp = minmPer
            #
            AnnPTopmLay.random.seed()
            self.WightAnnFrst =
AnnPTopmLay.random.randn(self.PolTop[0] , self.PolTop[1]) /
AnnPTopmLay.sqrt(self.PolTop[0] )
            self.BiasAnnFrst = AnnPTopmLay.random.randn(1 , self.PolTop[1])
/ AnnPTopmLay.sqrt(self.PolTop[1] )
            self.BestBiasAnnFrst = self.BiasAnnFrst
            self.BestWightAnnFrst = self.WightAnnFrst
            self.WightAnnScnd = AnnPTopmLay.random.randn(self.PolTop[1] ,
self.PolTop[2]) / AnnPTopmLay.sqrt(self.PolTop[1] )
```

```

        self.BiasAnnScnd = AnnPTopmLay.random.randn(1 , self.PolTop[2])
/ AnnPTopmLay.sqrt(self.PolTop[1] )
        self.BestBiasAnnScnd = self.BiasAnnScnd
        self.BestWightAnnScnd = self.WightAnnScnd
        self.HiddenANNLay = AnnPTopmLay.zeros((1, self.PolTop[1] ))
        self.OutputANNLay = AnnPTopmLay.zeros((1, self.PolTop[2]))

def FuncActivSigm(self,x):
    return 1 / (1 + AnnPTopmLay.exp(-x))
def printNet(self):
    print self.PolTop
    print self.WightAnnFrst

def DTSampEr(self,actualOutputANNLay):
    error          =          AnnPTopmLay.subtract(self.OutputANNLay,
actualOutputANNLay)
    SDErr=
AnnPTopmLay.sum(AnnPTopmLay.square(error))/self.PolTop[2]
    #print SDErr
    return SDErr

def ForwardRunning(self, X ):
    BufZ1 = X.dot(self.WightAnnFrst) - self.BiasAnnFrst
    self.HiddenANNLay = self.FuncActivSigm(BufZ1) # otpt of first
hddn layer
    BufZ2      =      self.HiddenANNLay.dot(self.WightAnnScnd)      -
self.BiasAnnScnd
    self.OutputANNLay = self.FuncActivSigm(BufZ2) # otpt second
hddn layer

```

```

def BackwardRunningMomentum(self, inp, PlanModlANN,
BuffCocaVanill):
    OutputANNNLay_btwnRes = (PlanModlANN -
self.OutputANNNLay)*(self.OutputANNNLay*(1-self.OutputANNNLay))
    hid_btwnRes =
OutputANNNLay_btwnRes.dot(self.WightAnnScnd.T) * (self.HiddenANNNLay * (1-
self.HiddenANNNLay))

    if BuffCocaVanill == 1:
        self.WightAnnScnd+=
(self.HiddenANNNLay.T.dot(OutputANNNLay_btwnRes) * self.lrate)
        self.BiasAnnScnd+= (-1 * self.lrate * OutputANNNLay_btwnRes)
        self.WightAnnFrst += (inp.T.dot(hid_btwnRes) * self.lrate)
        self.BiasAnnFrst+= (-1 * self.lrate * hid_btwnRes)

    else:

        self.WightAnnScnd += ( self.WightAnnScnd *self.momenRate)
+ (self.HiddenANNNLay.T.dot(OutputANNNLay_btwnRes) * self.lrate) #
velocity update
        self.WightAnnFrst += ( self.WightAnnFrst *self.momenRate) +
(inp.T.dot(hid_btwnRes) * self.lrate)
        self.BiasAnnScnd += ( self.BiasAnnScnd *self.momenRate) + (-
1 * self.lrate * OutputANNNLay_btwnRes) # velocity update
        self.BiasAnnFrst += ( self.BiasAnnFrst *self.momenRate) + (-1
* self.lrate * hid_btwnRes)

```

```

def testingANN(self, SampDat, ThroughErrTr):
    inp = AnnPTopmLay.zeros((1, self.PolTop[0])) # temp hold inp
    Desired = AnnPTopmLay.zeros((1, self.PolTop[2]))
    notpt = AnnPTopmLay.zeros((1, self.PolTop[2]))
    clasPerf = 0
    sse = 0
    testingSize = SampDat.shape[0]
    self.WightAnnFrst = self.BestWightAnnFrst
    self.WightAnnScnd = self.BestWightAnnScnd #load best knowledge
    self.BiasAnnFrst = self.BestBiasAnnFrst
    self.BiasAnnScnd = self.BestBiasAnnScnd #load best knowledge

    for s in AnnXRang(0, testingSize):

        inp[:] = SampDat[s,0:self.PolTop[0]]
        Desired[:] = SampDat[s,self.PolTop[0]:]

        self.ForwardRunning(inp )
        sse = sse+ self.DTSampEr(Desired)

        if(AnnPTopmLay.isclose(self.OutputANNLay, Desired,
atol=ThroughErrTr).any()):
            clasPerf = clasPerf +1

    return ( sse/testingSize, float(clasPerf)/testingSize * 100 )

```

```

def saveKnowledge(self):
    self.BestWightAnnFrst = self.WightAnnFrst
    self.BestWightAnnScnd = self.WightAnnScnd
    self.BestBiasAnnFrst = self.BiasAnnFrst
    self.BestBiasAnnScnd = self.BiasAnnScnd

def BP_GD(self, trainValRate, mRate,    stohCastVal, BuffCocaVanill):
#
    self.lrate = trainValRate
    self.momenRate = mRate

    inp = AnnPTopmLay.zeros((1, self.PolTop[0])) # temp hold inp
    Desired = AnnPTopmLay.zeros((1, self.PolTop[2]))
    #Er = []#AnnPTopmLay.zeros((1, self.maxm))
    iterEpoch = 0
    bestResMSE = 100
    besttraining = 0
    while iterEpoch < self.maxm and besttraining < self.minRetfp :

        sse = 0
        for s in AnnXRang(0, self.NumDTSamps):

            if(stohCastVal):
                pat = random.randint(0, self.NumDTSamps-1)
            else:
                pat = s

            inp[:] = self.trainingSampDat[pat,0:self.PolTop[0]]

```

```

Desired[:] = self.trainingSampDat[pat,self.PolTop[0]:]

self.ForwardRunning(inp )
self.BackwardRunningMomentum(inp , Desired,
BuffCocaVanill)
sse = sse+ self.DTSampEr(Desired)

mse = AnnPTopmLay.sqrt(sse/self.NumDTSamps*self.PolTop[2])

if mse < bestResMSE:
    bestResMSE = mse
    self.saveKnowledge()
    (x,besttraining) = self.testingANN(self.trainingSampDat, 0.2)

#Er = AnnPTopmLay.append(Er, mse)

iterEpoch=iterEpoch+1

return (mse,bestResMSE, besttraining, iterEpoch)

#

class ModulTaskANN: #

    def __init__(self, mquestTaskANN, trainingSampDat, testingSampDat,
maxValTimer, minRetfp, trainValRate, nummoduleANNs, tansKnowlg):

```

```

        self.tansKnowlglCore = tansKnowlg
self.trainingSampDat = trainingSampDat
self.testingSampDat = testingSampDat
self.maxValTimer = maxValTimer
self.minCrit = minRetfp
self.nummoduleANNs = nummoduleANNs # number of network
moduleANNs (i.e tasks with shared knowledge representation)

self.mquestTaskANN = mquestTaskANN

self.trainValRate = trainValRate
self.trainingTolerance = 0.20 # [eg 0.15 otpt would be seen as 0] [
0.81 would be seen as 1]
self.testingTolerance = 0.49

def BufTransKnowlg(self, WeightPrv, WeightNxt):
    x=0
    y = 0
    WeightNxt[x:x+WeightPrv.shape[0], y:y+WeightPrv.shape[1]] =
WeightPrv                                     #(ANNList[n].WightAnnFrst ->
ANNList[n+1].WightAnnFrst)
    return WeightNxt

def mainAlg(self):

    mRate = 0.05

    stohCastVal = 1 #

```

```
BuffCocaVanill = 1 #
```

```
ANNList = [None]*10 #
```

```
depthSearch = 10
```

```
trainingPerf = AnnPTopmLay.zeros(self.nummoduleANNs)
```

```
trainingMSE = AnnPTopmLay.zeros(self.nummoduleANNs)
```

```
testingPerf = AnnPTopmLay.zeros(self.nummoduleANNs)
```

```
testingMSE = AnnPTopmLay.zeros(self.nummoduleANNs)
```

```
erPlot =
```

```
AnnPTopmLay.random.randn(self.maxValTimer/(depthSearch),self.nummoduleANNs)
```

```
for n in AnnXRang(0, self.nummoduleANNs):
```

```
    ANNList[n] = ANN(self.mquestTaskANN[n],  
self.trainingSampDat, self.testingSampDat, depthSearch, self.minCrit)
```

```
    cycles = 0
```

```
    index = 0
```

```
    while(depthSearch*self.nummoduleANNs*cycles)
```

```
<self.maxValTimer:
```

```
        cycles =cycles + 1
```

```
for n in AnnXRang(0, self.nummoduleANNs-1):
```

```
    (erPlot[index, n], trainingMSE[n], trainingPerf[n], iterEpochs)  
= ANNList[n].BP_GD(self.trainValRate, mRate, stohCastVal, BuffCocaVanill)
```

```

        if(self.tansKnowlglCore ==1):
            ANNList[n+1].WightAnnFrst =
self.BufTransKnowlg(ANNList[n].WightAnnFrst, ANNList[n+1].WightAnnFrst)
            ANNList[n+1].BiasAnnFrst =
self.BufTransKnowlg(ANNList[n].BiasAnnFrst, ANNList[n+1].BiasAnnFrst)
            ANNList[n+1].WightAnnScnd =
self.BufTransKnowlg(ANNList[n].WightAnnScnd,
ANNList[n+1].WightAnnScnd)
            ANNList[n+1].BiasAnnScnd =
self.BufTransKnowlg(ANNList[n].BiasAnnScnd, ANNList[n+1].BiasAnnScnd)
            (erPlot[index, self.nummoduleANNs-1],
trainingMSE[self.nummoduleANNs-1], trainingPerf[self.nummoduleANNs-1],
iterEpochs) = ANNList[self.nummoduleANNs-1].BP_GD(self.trainValRate,
mRate, stohCastVal, BuffCocaVanill) # BP for last moduleANN
            index = index + 1

    for n in AnnXRang(0, self.nummoduleANNs):
        (testingMSE[n], testingPerf[n]) =
ANNList[n].testingANN(self.testingSampDat, self.testingTolerance)

    return (erPlot, trainingMSE, trainingPerf, testingMSE, testingPerf)

def SampDatNorm(data, inpsize, OutputANNLaysize): #
    trainingdumTr = data[:,AnnPTopmLay.array(range(0,inpsize))]
    dumTr = AnnPTopmLay.amax(trainingdumTr, axis=0)
    TrDumSp = abs(trainingdumTr/dumTr)
    return AnnPTopmLay.concatenate(( TrDumSp[:,range(0,inpsize)],
data[:,range(inpsize,inpsize+OutputANNLaysize)]), axis=1)

```

```
def main():
```

```
    IndxTaskLev = 1 #
```

```
    if IndxTaskLev == 1:
```

```
        TrDat = AnnPTopmLay.loadumTrxt("training.csv", delimiter=',')
```

```
        TesDat = AnnPTopmLay.loadumTrxt("testing.csv", delimiter=',') #
```

```
        hddn = 2
```

```
        inp = 4
```

```
        otpt = 2
```

```
        TrDTSamps = 110
```

```
        testingSize = 40
```

```
        trainValRate = 0.1
```

```
        mRate = 0.01
```

```
        trainingSampDat = SampDatNorm(TrDat, inp, otpt)
```

```
        testingSampDat = SampDatNorm(TesDat, inp, otpt)
```

```
        maxmTime = 500
```

```
    if IndxTaskLev == 2:
```

```
        trainingSampDat = AnnPTopmLay.loadumTrxt("4bit.csv",
delimiter=',')
```

```

testingSampDat      =      AnnPTopmLay.loadumTrxt("4bit.csv",
delimiter=',') #
    hddn = 2
    inp = 4
    otpt = 1
    TrDTSamps = 16
    testingSize = 16
    trainValRate = 0.9
    mRate = 0.01
    maxmTime = 4000

if IndxTaskLev == 3:
    trainingSampDat    =      AnnPTopmLay.loadumTrxt("xor.csv",
delimiter=',') #
    testingSampDat     =      AnnPTopmLay.loadumTrxt("xor.csv",
delimiter=',') #
        hddn = 2
        inp = 2
        otpt = 1
        TrDTSamps = 4
        testingSize = 4
        trainValRate = 0.9
        mRate = 0.01
        maxmTime = 500

#print(trainingSampDat)

base = [inp, hddn, otpt] # d

```

```

#
maxmRun = 10 #

minmCrit = 95 #

nummoduleANNs = 4 #

mquestTaskANN = AnnPTopmLay.array([base, base,base,base])
for i in AnnXRang(1, nummoduleANNs):
    #mquestTaskANN[i][0] += (i*2)
    mquestTaskANN[i][1] += (i*2) # e adapt the number of hddn
neurons for each task.
print mquestTaskANN #

trainingPerf =
AnnPTopmLay.random.randn(maxmRun,nummoduleANNs)
testingPerf =
AnnPTopmLay.random.randn(maxmRun,nummoduleANNs)
mnFortraininging = AnnPTopmLay.zeros(nummoduleANNs)
stdFortraininging = AnnPTopmLay.zeros(nummoduleANNs)
mnFortestinging = AnnPTopmLay.zeros(nummoduleANNs)
stdFortestinging = AnnPTopmLay.zeros(nummoduleANNs)

trainingMSE =
AnnPTopmLay.random.randn(maxmRun,nummoduleANNs)
testingMSE =
AnnPTopmLay.random.randn(maxmRun,nummoduleANNs)
iterEpochs = AnnPTopmLay.zeros(maxmRun)
Time = AnnPTopmLay.zeros(maxmRun)

```

```

for tansKnowlg in AnnXRang(0, 2 ):
    for run in AnnXRang(0, maxmRun ):
        mt = ModulTaskANN(mquestTaskANN, trainingSampDat,
testingSampDat,
maxmTime,minmCrit,trainValRate,
nummoduleANNs,tansKnowlg)
        (erPlot, trainingMSE[run,:], trainingPerf[run:],
testingMSE[run:], testingPerf[run,:]) = mt.mainAlg()
        print testingPerf[run,:]

    for moduleANN in AnnXRang(0, nummoduleANNs ):
        mnFortraininging[moduleANN] =
AnnPTopmLay.mean(trainingPerf[:,moduleANN])
        stdFortraininging[moduleANN] =
AnnPTopmLay.std(trainingPerf[:,moduleANN])
        mnFortestinging[moduleANN] =
AnnPTopmLay.mean(trainingPerf[:,moduleANN])
        stdFortestinging[moduleANN] =
AnnPTopmLay.std(trainingPerf[:,moduleANN])

    print mnFortraininging
    print stdFortraininging
    print mnFortestinging
    print stdFortestinging

plt.figure()

```

```
plt.plot(erPlot) #  
plt.ylabel('error')  
plt.savefig('OutputANNLay.png')
```

```
if __name__ == "__main__": main()
```

ДОДАТОК Д
Слайди презентації

Міністерство освіти та науки України
Національний Університет «Запорізька політехніка»



Дослідження та програмна реалізація методів класифікації зображень на основі навчання з підкріпленням

Капленко А.А.	Коцур М.І.
КНТ-111М	к.т.н., доцент

[1] – <http://www.zn.ua.edu.ua/>
[2] – <http://www.zn.ua.edu.ua/>

Рисунок Д.1 – Слайд №1

2

Об'єкт, предмет та мета роботи

- **Об'єкт дослідження** – процес використання методів навчання з підкріпленням для вирішення задач класифікації зображень.
- **Предмет дослідження** – методи навчання з підкріпленням автоматизації класифікації зображень.

Метою роботи – є дослідити, проаналізувати та реалізувати методи навчання з підкріпленням для побудови модулярних нейронних мереж у якості основи для моделей класифікації зображень.

Рисунок Д.2 – Слайд №2

Порівняння нейромережевих основ для моделей класифікації зображень

	AlexNet	ResNet	VIT
Структурна складність	На базі CNN, більш спрощена архітектура (менше шарів, але більше нейронів)	На базі CNN, більш спрощена архітектура (менше шарів)	Більш складна, складена структура (Layer Norm + MSP+ MLP)
Використання GPU під час навчання	Можливе	Можливе	Можливе (ускладнене пересилкою даних)
Складність функцій активації	Ускладнена функція для підвищення точності (ReLU)	Ускладнена функція для підвищення точності (ReLU)	Складена функція для різних блоків (GELU + шари softmax)
Робота на великих масивах даних	Можлива (середні точність та швидкість роботи)	Можлива (середня точність та більша швидкість роботи)	Можлива (висока точність та низька швидкість роботи)

Рисунок Д.3 – Слайд №3

Порівняння підходів машинного навчання для класифікації зображень

	Навчання з учителем	Навчання без учителя	Навчання з підкріпленням
Вимоги до вхідних даних	Найвищі	Мінімальні	Середні
Вимоги до метапараметрів навчання	Високі	Високі	Середні
Складність організації	Середня	Висока	Висока
Адаптованість рішень (моделей)	Середня	Висока	Найвища

Рисунок Д.4 – Слайд №4

Постановка завдання

Нехай задано вибірку даних, яка складається з зображень, які характеризуються низкою ознак – $\{x_1, x_2, \dots, x_n\}$. Тоді використовуючи модель на основі штучної нейронної мережі – $Model = \{ANN\}$, необхідно пропускаючи через неї всю вибірку даних $DataSet = \{x_1, x_2, \dots, x_n, y\}$, визначити клас кожного з зображень – визначити залежну ознаку $\{y\}$. Умовами регулювання процесу навчання є визначення нагороди/штрафу під час навчання, з метою отримання рішення $Model = \{ANN\}$, що характеризується найбільшою точністю роботи: $Error_{ANN} \rightarrow 0$.

Рисунок Д.5 – Слайд №5

Псевдокод розробленого методу

inputs: Training dataset (x, y) : x images and y labels; threshold τ .
output: MNN-Tree structure T .
 $CATEGORIES = \{c: c \text{ is a unique label} \in y\}$ // set of all categories
 $TREE = \{\text{untrained root DNN}\}$ // set holding the tree structure
 $S = CATEGORIES$ // all categories of the dataset come under the root
while $\exists$ an untrained DNN in $TREE$
 (1) $lc = 0$ // layer_count: keeps track of the number of layers in the DNN
 (2) **do**
 $lc = lc + 1$
 initialize a DNN D_{lc} , with lc convolutional layers.
 train D_{lc} to classify all categories of S .
 if $lc \neq 1$ **then** calculate $\Delta AD_{lc, lc-1} = \frac{VA_{lc} - VA_{lc-1}}{MS_{lc} - MS_{lc-1}}$
 while ($lc = 1$ or $\Delta AD_{lc, lc-1} > \tau$)
 (3) $DNN_{conf_{lc}} = D_{lc-1}$ // select the DNN configuration for the module
 (4) $SOFTMAX_MATRIX = \text{softmax output of } DNN_{conf_{lc}} \forall c \in S$ // softmax output matrix
 (5) $CHILD_GROUPS = \emptyset$ // keeps track of newly formed children super-groups
 (6) **for each** $c \in S$ // use the ASL
 find set V , s.t. $\{V \subset S : \text{prob}(c = v) = \text{argf}(SOFTMAX_MATRIX[c, v]), \forall v, v \in V\}$
 add untrained DNN corresponding to V into $TREE$
 $CHILD_GROUPS = CHILD_GROUPS \cup V$ // similar categories added to set of children
 (7) train $DNN_{conf_{lc}}$ to classify between elements of $CHILD_GROUPS$
 (8) $S = \text{children of next untrained DNN in } TREE$
return tree

$$\Delta AD_{ij} = \frac{VA_i - VA_j}{MS_i - MS_j}$$

Рисунок Д.6 – Слайд №6

Порівняння мов програмування

Критерій	Python	Java	Scala	R
Парсинг вибірок даних	Високий рівень	Середній рівень	Високий рівень	Високий рівень
Препроцесинг та обробка даних	Достатньо методів	Мала кількість методів	Достатньо методів	Достатньо методів
Масштабування застосунків	Високий рівень	Високий рівень	Середній рівень	Середній рівень
Програмні надбудови для машинного навчання	Велика кількість	Недостатньо	Середня кількість	Середня кількість
Підтримка спільноти	Високий рівень	Високий рівень	Середній рівень	Середній рівень

Рисунок Д.7 – Слайд №7

Порівняння фреймворків МН

Критерій	TensorFlow	PyTorch	CNTK	Theano
Топології нейронних мереж	Усі сучасні топології	Обмежена кількість	Обмежена кількість	Обмежена кількість
Організація паралельних обчислень	Макиміве використання CPU та GPU	Обмеження	Макиміве використання CPU	Обмеження
Швидкість розробки	Висока	Висока	Середня	Середня
Простота розгортання застосунків	Просто	Середня складність	Обмежені можливості	Обмежені можливості
Оновлення та підтримка	Комплексна, середня частота	Комплексна, середня частота	Недостатня, середня частота	Низька, низька частота

Рисунок Д.8 – Слайд №8

Порівняння середовищ розробки

Критерії	Anaconda	PyCharm	Atom	Php Storm
Доступність	Безкоштовна	Платний	Безкоштовне	Платний
Модулі надбудови машинного навчання	Присутні (деякі необхідно встановлював окремо)	Доступні, відрізняються за вартістю	Присутні, ускладнене встановлення	Доступні, відрізняються за вартістю
Спільнота підтримки оновлень	Активно оновлюється, активна спільнота	Активно оновлюється, активна спільнота	Оновлюється поступово, спільнота малоактивна	Активно оновлюється, активна спільнота
Робота засобами TensorFlow	Налагоджена	Налагоджена	Ускладнена	Налагоджена

Рисунок Д.9 – Слайд №9

Рівні розроблюваного застосунку

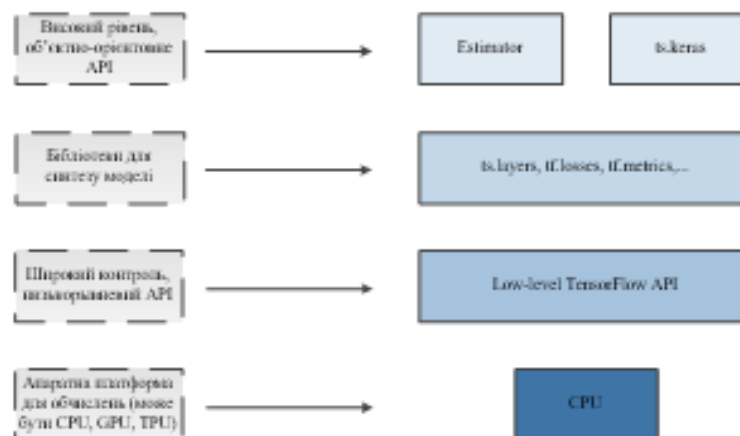


Рисунок Д.10 – Слайд №10

Модулі розроблюваного застосунку за етапами розробки

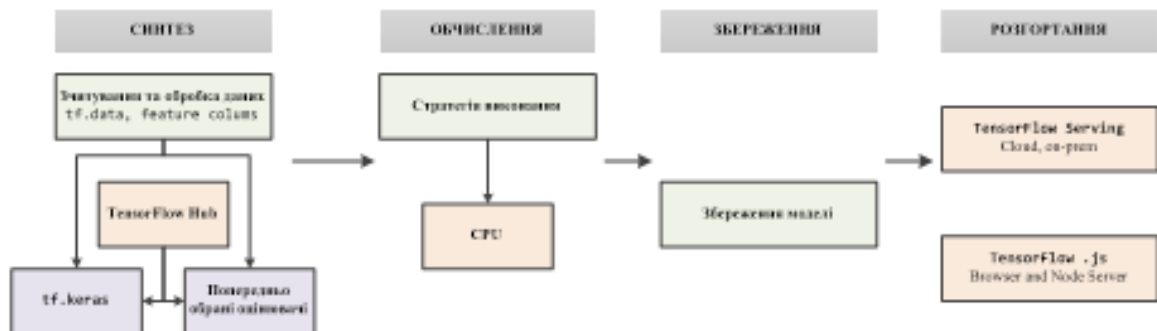


Рисунок Д.11 – Слайд №11

Імплементация структуры в разработку

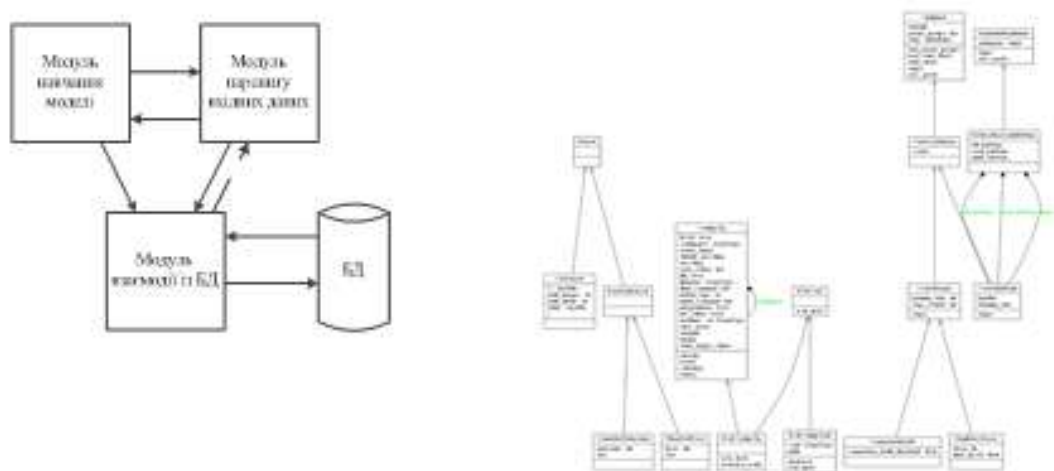


Рисунок Д.12 – Слайд №12

Вибірки даних для тестування

Вибірка	Розмір зображень	Кількість зображень для навчання	Кількість зображень для тестування	Кількість класів зображень
CIFAR-10	32x32x3	50000	10000	10
CIFAR-100	32x32x3	50000	10000	100
SVHN	32x32x3	604388	26032	10
EMNIST	28x28x1	112800	18800	47
ImageNet-2012	Варіюється	1200000	75000	1000
ImageNet-2012 (підвибірка)	Варіюється	26000	2000	20
Caltech (підвибірка)	Варіюється	2000	400	11

Рисунок Д.13 – Слайд №13

Аналіз результатів

Dataset	Мережа	Розмір моделі в пам'яті (KB)	Кількість ітерацій	Помилка моделі на тестових даних
CIFAR-10	VGG-16	78,410	313 M	0,067
	VGG-Pruned	28,200	206 M	0,066
	MHM	806	28 M	0,079
CIFAR-100	VGG-16	78,590	207 M	0,295
	VGG-Pruned	28,910	210 M	0,252
	ResNet	141,100	25,800 M	0,192
	MHM	832	34 M	0,209
SVHN	ResNet	11,000	1,935 M	0,016
	MHM	522	30 M	0,018
	EDEN	—	—	0,117
EMNIST	MHM	363	4M	0,078
	VGG-16	528,000	15,300 M	0,076
ImageNet-2012 (subset)	ResNet	84,000	3,640 M	0,081
	MHM	1,872	605 M	0,124
	VGG-16	528,120	15,300 M	0,295
	ResNet-34	84,100	3,640 M	0,076

Рисунок Д.14 – Слайд №14

Висновки

- Під час виконання дипломної кваліфікаційної роботи магістра, було запропоновано новий метод класифікації зображень на основі моделі на базі модулярних нейронних мереж, що навчається використовуючи підхід машинного навчання з підкріпленням.
- Запропонований метод використовує найсучасніші засади навчання з підкріпленням та модулярних нейронних мереж у якості базису для моделей.
- Під час розробки програми та аналізу розробленого програмного комплексу з'явилися ідеї з удосконалення, такі як: використання паралельних потужностей обчислювальних систем на базі GPU для організації на їх базі безпосередньо модулярних нейромереж.

Рисунок Д.15 – Слайд №15