Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «**Дослідження та покращення точності класифікації нео-фаззі систем для задач розпізнавання образів**»

Спеціальність 123 – Комп'ютерна інженерія.
Галузь знань: 12 – Інформаційні технології.
Освітня програма «Комп'ютерна інженерія».

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КІ– 41
ТКАЧЕНКО Кирило Сергійович _____

Керівник: к.т.н., доцент, доцент закладу вищої освіти
БИКОВА Тетяна Володимирівна _____

Рецензент: к.т.н., доцент кафедри математичного моделювання та аналізу даних
КОРОБЧИНСЬКИЙ Кирило _____

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 89 сторінок, з яких 47 сторінок основної частини, що містять 17 рисунків, 4 таблиці, 8 формул та 6 лістингів. До роботи додано 19 джерел літератури.

Метою кваліфікаційної роботи є розробка, математичне обґрунтування та програмна реалізація нео-фаззі системи для класифікації зображень. Такий підхід поєднує нечітку логіку з методами машинного навчання задля підвищення точності розпізнавання об’єктів у середовищах з високим рівнем невизначеності.

Об’єктом дослідження є процес класифікації зображень у системах комп’ютерного зору.

Предмет дослідження – нео-фаззі системи, що базуються на поєднанні нечіткої логіки та алгоритмів машинного навчання.

У роботі проведено аналіз еволюції нечітких систем, побудовано математичну модель нео-фаззі системи, реалізовано її програмну реалізацію та виконано порівняльне тестування із класичними підходами. Запропоновано рекомендації щодо використання таких систем у медичних, геоінформаційних та промислових додатках.

Ключові слова: нечітка логіка, нео-фаззі система, класифікація зображень, машинне навчання, фаззіфікація, дефаззіфікація.

ABSTRACT

The explanatory note of the bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices. The total volume of the work is 89 pages, including 47 pages of the main part with 17 figures, 4 tables, 8 formulas, and 6 code listings. The list of references includes 19 sources.

The purpose of the qualification work is the development, mathematical justification, and implementation of a neo-fuzzy system for image classification. This approach combines fuzzy logic with machine learning methods to increase recognition accuracy in environments with high uncertainty.

The object of the research is the process of image classification in computer vision systems.

The subject of the research is neo-fuzzy systems that integrate fuzzy logic with machine learning algorithms.

The work analyzes the evolution of fuzzy systems, constructs a mathematical model of a neo-fuzzy system, implements its software version, and performs comparative testing with classical methods. Recommendations for the application of such systems in medical, geoinformation, and industrial domains are proposed.

Keywords: fuzzy logic, neo-fuzzy system, image classification, machine learning, fuzzification, defuzzification..

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ANFIS	– Adaptive Neuro-Fuzzy Inference System
CNN	– Convolutional Neural Network
GPU	– Graphics Processing Unit
F1	– F1-міра
HSV	– Hue, Saturation, Value
MSE	– Mean Squared Error
ReLU	– Rectified Linear Unit
RGB	– Red, Green, Blue (колірна модель)
SVM	– Support Vector Machine
SSD	– Solid-State Drive
VS Code	– Visual Studio Code
БМ	– байєсівська мережа
ЕА	– еволюційний алгоритм
МН	– машинне навчання
НЛ	– нечітка логіка
НМ	– нейронна мережа
ШІ	– штучний інтелект

ВСТУП

У сучасному світі зростає потреба у системах, здатних ефективно працювати з невизначеними, неповними або суперечливими даними. Однією з таких технологій є нечітка логіка — підхід, який дозволяє моделювати процеси з нечіткими або розмитими межами між станами, що неможливо реалізувати за допомогою класичної булевої логіки.

Особливе місце серед цих систем займають так звані нео-фаззі моделі, які поєднують нечітку логіку з методами машинного навчання, забезпечуючи автоматичну адаптацію до нових умов. Такі системи мають вищу точність і гнучкість, що особливо важливо у задачах класифікації зображень, де природні сцени не мають чітко окреслених ознак.

Актуальність роботи. У зв'язку зі стрімким розвитком технологій комп'ютерного зору та зростаючим обсягом візуальної інформації особливої актуальності набувають методи, здатні ефективно аналізувати зображення в умовах невизначеності. Класичні моделі машинного навчання часто не справляються зі складними або розмитими класами, у той час як нео-фаззі системи демонструють високу точність та інтерпретованість результатів. Їх здатність до адаптації відкриває нові можливості у таких сферах, як медицина, автономний транспорт, охорона довкілля та безпека.

Метою дослідження є розробка, математичне обґрунтування та програмна реалізація нео-фаззі системи класифікації зображень, що поєднує адаптивні можливості машинного навчання з гнучкістю нечіткої логіки, з подальшим аналізом її ефективності на практичних прикладах.

Об'єкт дослідження – процес автоматизованого аналізу та класифікації зображень у системах комп'ютерного зору.

Методи дослідження. Під час виконання дослідження використовувались методи теорії нечітких множин, машинного навчання (нейронні мережі, SVM), статистичні методи оцінки ефективності, методи математичного моделювання та програмна реалізація із застосуванням мови Python та спеціалізованих бібліотек для обробки зображень.

Предмет дослідження. Нео-фаззі системи, що базуються на поєднанні методів нечіткої логіки й машинного навчання, а також їх застосування для класифікації зображень.

Завдання дослідження

1. Проаналізувати еволюцію систем нечіткої логіки та становлення нео-фаззі підходів.
2. Розробити математичну модель нео-фаззі системи, орієнтовану на задачі класифікації зображень.
3. Реалізувати програмну модель системи у відповідному середовищі розробки.
4. Провести порівняльний аналіз ефективності реалізованої системи з класичними методами.
5. Сформулювати практичні рекомендації щодо впровадження нео-фаззі систем у прикладних задачах.

РОЗІДЛ 1

ЕВОЛЮЦІЯ СИСТЕМ НЕЧІТКОЇ ЛОГІКИ: ВІД ОСНОВ ДО СУЧАСНОСТІ

Термін "нечіткість" стосується явищ, які є невизначеними або розпливчастими. У реальному світі часто виникають ситуації, коли неможливо однозначно визначити істинність чи хибність певного стану. У таких випадках нечітка логіка (НЛ) надає необхідну гнучкість для прийняття рішень, дозволяючи враховувати неточності та невизначеності.

НЛ, як форма багатозначної логіки, дозволяє змінним приймати значення у діапазоні від 0 до 1, на відміну від традиційних бінарних значень "істина/хибність". Подібна властивість забезпечує математичний підхід до представлення невизначеності у процесі прийняття рішень, що робить її особливо корисною для задач з неоднозначною інформацією. В основі концепції лежить ідея, що реальний світ не обмежується чіткими категоріями "істинно" чи "хибно", а включає численні проміжні стани (рис.1.1). НЛ допускає часткові істини, коли твердження може бути одночасно частково істинним і частково хибним.

Однією з ключових концепцій НЛ є функція належності, яка визначає ступінь належності вхідного значення до певного набору або категорії. Функція відображає вхідне значення на ступінь належності в інтервалі [0, 1], де 0 відповідає повній відсутності належності, а 1 — повній належності. Реалізація НЛ відбувається через нечіткі правила у вигляді операторів "якщо-то", що забезпечують взаємозв’язок між вхідними та вихідними змінними [1].

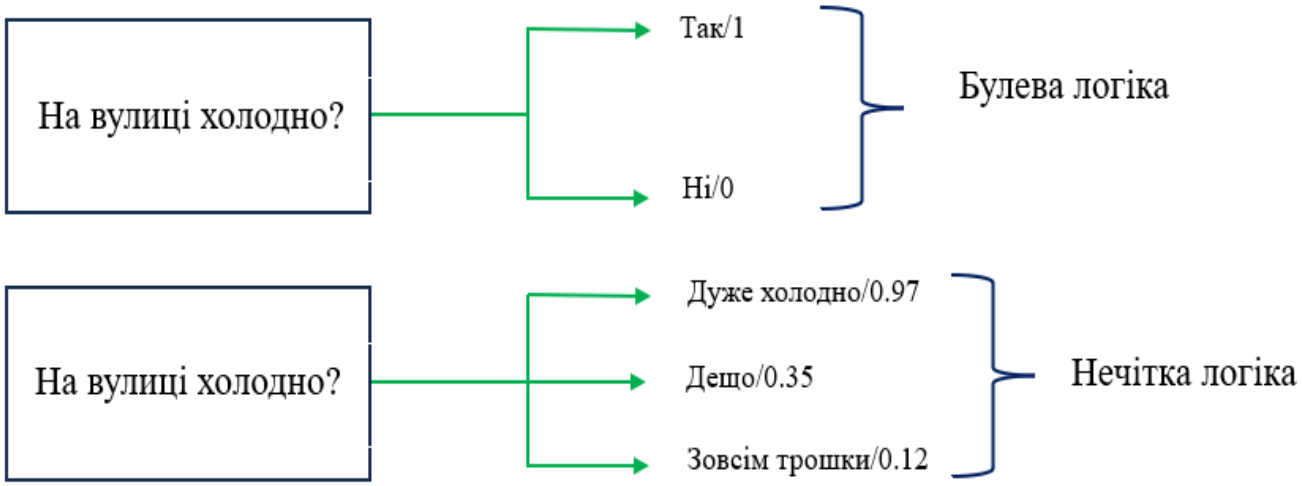

Рисунок 1.1 – Представлення традиційної логіки у порівнянні з нечіткою.

З моменту свого виникнення нечітка логіка зазнала значного розвитку. Класичні нечіткі системи, запропоновані Лотфі Заде у 1960-х роках, стали революційним підходом до роботи з невизначеністю. Вони базуються на теорії нечітких множин, яка дозволяє використовувати ступені належності для опису невизначених станів, що дало змогу забезпечити більш гнучкий підхід до прийняття рішень, особливо у сферах автоматизації, управління та наближених міркувань [2].

У подальшому розвиток теорії нечітких множин створив фундамент для появи нео-нечітких систем, що буде розглянуто нижче.

1.1 Основні принципи роботи фаззі систем

Загальний опис нечітких систем розглянуто раніше, що дає можливість перейти до глибшого аналізу їхньої структури [3]. Фаззі системи є складними інтегрованими моделями, які використовують НЛ для вирішення задач із невизначеністю. Їхня структура базується на чотирьох ключових елементах: фаззіфікації, базі знань, машині висновків та дефаззіфікації. Кожен із цих компонентів відіграє важливу роль у забезпеченні ефективності роботи системи та взаємодії між вхідними і вихідними даними.

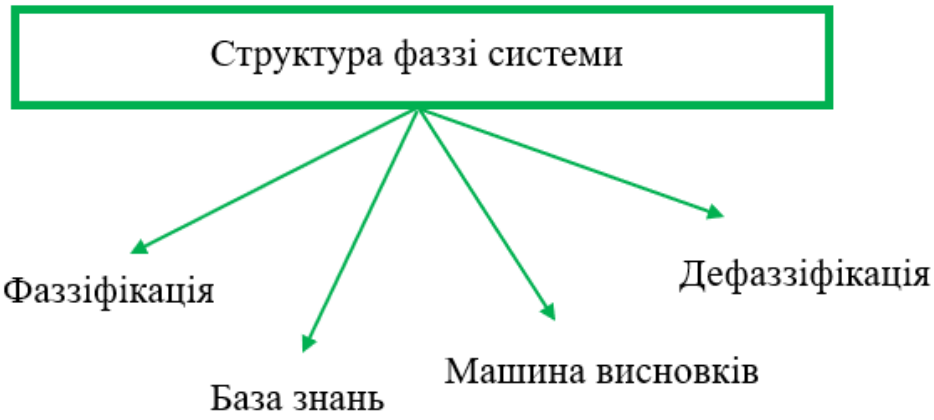

Рисунок 1.2 – Структура фаззі системи.

Фаззіфікація - це процес перетворення чітких даних у нечіткі множини за допомогою функцій належності. Функції визначають ступінь належності кожного вхідного значення до певної лінгвістичної категорії, наприклад, "низька", "середня" або "висока". Значення належності, що лежать у діапазоні від 0 до 1, дозволяють ураховувати неоднозначність та можливі похибки у вихідних даних, що є особливо важливим при роботі з реальними системами.

База знань складається з правил у форматі "якщо-то", які встановлюють залежності між вхідними змінними та очікуваними результатами. Формування таких правил базується на знаннях експертів або аналітичних дослідженнях. Наприклад, у системах управління кліматом правило може виглядати так: "ЯКЩО температура висока і вологість низька, ТО вентилятор має працювати на високій швидкості". Правила задають основу для логічних операцій, які виконуються в наступних етапах роботи системи.

Машина висновків відповідає за обробку нечітких правил і генерування нечітких вихідних даних. Найбільш поширеними підходами до реалізації цього етапу є методи Мамдані та Сугено.

Метод Мамдані (рис. 1.3), запропонований у 1975 році, застосовується для отримання нечітких виходів через агрегування та обрізання функцій належності відповідно до правил. Основним етапом є побудова нечіткої множини виходу для кожного правила, після чого ці множини комбінуються для формування єдиного

нечіткого вихідного значення. Метод є інтуїтивно зрозумілим і широко застосовується у системах управління завдяки своїй здатності ефективно працювати з багатьма змінними [4].

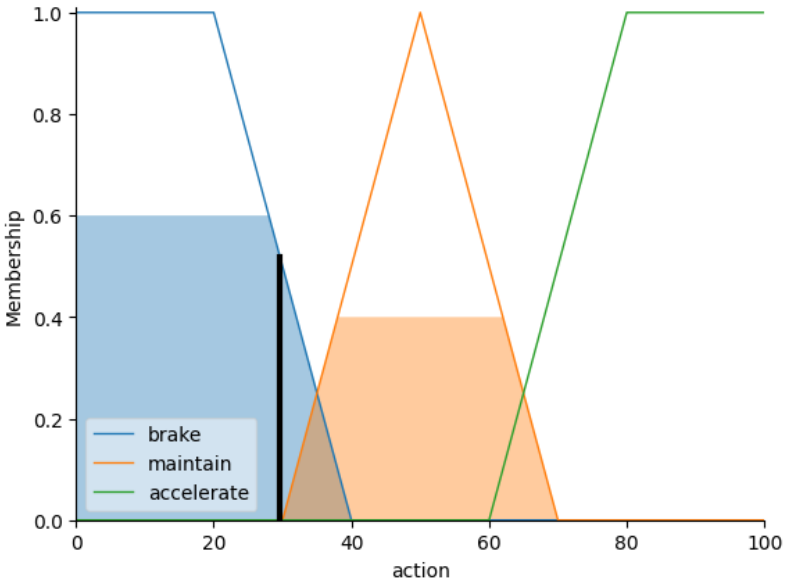

Рисунок 1.3 – Представлення методу Мамдані.

Метод Сугено, розроблений пізніше, дозволяє отримувати чітке вихідне значення за допомогою лінійних чи константних функцій. На відміну від Мамдані, вихід у цьому методі завжди є числовим, що полегшує інтеграцію таких систем у задачі прогнозування чи автоматизації. Висновок базується на ваговому середньому значенні вихідних функцій належності, що забезпечує високу швидкість обчислень. Метод Сугено часто використовується у задачах, де необхідні точні числові результати, таких як моделювання складних процесів [5].

Дефаззіфікація завершує роботу фаззі системи, переводячи нечітке вихідне значення у чітке, що може бути використане у прийнятті рішень або для управління процесами. Найбільш популярним підходом є метод центру тяжіння, який визначає чітке значення як зважене середнє всіх можливих вихідних значень, що забезпечує баланс між точністю і стабільністю вихідних даних.

Таким чином, структура фаззі систем забезпечує обробку невизначеної інформації на кожному етапі, дозволяючи моделювати складні системи та процеси у різноманітних галузях науки і техніки. Їхня здатність працювати з нечіткими даними та генерувати чіткі результати робить їх важливим інструментом для вирішення задач із високим ступенем складності.

1.2 Особливості, переваги та виклики традиційних нечітких систем

Однією з ключових переваг традиційних нечітких систем є здатність інтерпретувати експертні знання за допомогою нечітких правил, що дозволяє кодувати інформацію, отриману з досвіду або інтуїції, як зазначалось раніше. Апроксимаційна здатність нечітких систем дозволяє моделювати процеси у таких сферах, як автоматизація, управління транспортом, медична діагностика, де чітке математичне моделювання є складним або неможливим. Наприклад, вони успішно використовуються для розробки інтелектуальних регуляторів та забезпечують гнучке управління в умовах мінливих даних.

Однак традиційні нечіткі системи мають суттєве обмеження — відсутність здатності до самонавчання. Властивість самонавчання означає, що система не може адаптувати свої правила або функції належності на основі нових даних. Правила формулюються вручну, часто на основі знань експертів, що робить систему статичною. У динамічних середовищах, де умови постійно змінюються, традиційні нечіткі системи втрачають свою ефективність, оскільки не здатні автоматично оновлювати свої параметри. Наприклад, у задачах прогнозування або класифікації нових зразків необхідність ручного коригування правил може значно ускладнити процес адаптації. Інтеграція методів машинного навчання, таких як нейронні мережі або еволюційні алгоритми, дозволяє долати ці обмеження, створюючи так звані нео-нечіткі системи, які можуть вивчати залежності між вхідними і вихідними даними самостійно.

Традиційні математичні підходи, такі як методи класичної алгебри або чисельного аналізу, часто виявляються недостатніми для роботи з невизначеними чи нечіткими даними. Методи базуються на точних значеннях і

чітко визначених залежностях між змінними. Наприклад, класичні статистичні методи припускають наявність повних і достовірних даних, що не завжди відповідає реальним умовам. НЛ заповнює цей розрив, дозволяючи працювати з нечіткими множинами, які описують стан системи у вигляді ступенів належності. Подібний підхід робить нечіткі системи особливо корисними там, де традиційні підходи не можуть забезпечити адекватного рішення, наприклад, у задачах з неповними або суперечливими даними.

Ще однією важливою проблемою традиційних нечітких систем є обмеженість їх використання для роботи з неевклідовими структурами даних. Евклідові дані, такі як координати точок у просторі або метрики на площині, є стандартними для багатьох моделей. Традиційні нечіткі системи добре працюють із такими даними, дозволяючи створювати прості й ефективні моделі. Однак у сучасних задачах часто виникає необхідність працювати з більш складними структурами, такими як графи, мережі чи неевклідові простори. Наприклад, у задачах аналізу соціальних мереж або біоінформатики дані часто мають графову природу, де відстані не відповідають правилам евклідової геометрії. Традиційні нечіткі системи виявляються менш ефективними в таких умовах, оскільки їх алгоритми не враховують складність та багатовимірність таких даних. Розвиток нових методів, здатних адаптуватися до неевклідових структур, є важливим напрямком у вдосконаленні нечітких систем [6].

1.3 Виникнення та розвиток нео-фаззі систем

Розвиток нео-фаззі систем тісно пов’язаний із еволюцією традиційної нечіткої логіки, яка була вперше запропонована Лотфі Заде у 1965 році. Одним із ключових історичних моментів стало поєднання нечіткої логіки з методами машинного навчання (МН) у 1980–1990-х роках, коли почали розроблятися гібридні підходи, здатні адаптувати правила та функції належності на основі даних. Розвиток обчислювальної техніки, включаючи збільшення обчислювальної потужності комп’ютерів та поява розподілених систем, також став каталізатором для створення адаптивних систем.

Важливою подією було впровадження нейро-фаззі систем, які поєднали нечітку логіку з нейронними мережами (НМ), що дозволило автоматизувати процес навчання функцій належності та правил, забезпечивши динамічну адаптацію до змін у середовищі. Наприклад, модель адаптивної нейро-нечіткої системи виводу (ANFIS), запропонована у 1993 році Джангом, стала основою для подальшого розвитку інтегрованих систем [7].

Нео-фаззі системи можуть автоматично навчатися на основі данних і адаптувати свої правила та функції належності до змін у середовищі. Досягається це за допомогою методів МН, таких як градієнтний спуск чи генетичні алгоритми, які дозволяють оптимізувати параметри без потреби у втручанні людини. Іншою ключовою характеристикою є здатність до ефективної роботи з великими обсягами даних. Завдяки вдосконаленим алгоритмам обробки інформації нео-фаззі системи успішно справляються з аналізом складних структур данних.

Суттєвим покращенням виступає зниження суб’єктивності. Використання автоматизованих методів налаштування параметрів, таких як вибір функцій належності або формування правил, забезпечує більшу об’єктивність і стабільність роботи системи. Окрім цього, нео-фаззі системи демонструють вищу швидкість і продуктивність завдяки оптимізованим алгоритмам обчислень і можливості використання розподілених систем.

Доцільно доповнити цей блок аналізом змін у структурі систем після впровадження ідей нео-фаззі, оскільки саме ці трансформації визначили їхній успіх у вирішенні сучасних складних задач.

Впровадження ідей нео-фаззі значно змінило структуру традиційних нечітких систем. Основні зміни стосувалися інтеграції нових компонентів і методів обробки, що дозволило системам працювати в умовах динамічних середовищ та з великими обсягами даних.

Функції належності стали динамічними. База знань стала гнучкішою. Замість фіксованих правил у форматі "якщо-то", які створювалися вручну, нео-фаззі системи використовують алгоритми, що автоматично формують і

оптимізують правила на основі вхідних даних. Машина висновків отримала здатність до адаптації. У нео-фаззі системах додається механізм навчання для оптимізації процесу висновків. Замість статичних алгоритмів використовуються гібридні підходи, що поєднують нейронні мережі з фаззі логікою. Наприклад, у моделях типу ANFIS нейронна мережа налаштовує параметри нечіткої системи, забезпечуючи більшу точність та узгодженість рішень [7].

Обробка даних була розширена для багатовимірних і неевклідових структур. У традиційних системах акцент робився на роботу з евклідовими даними, що обмежувало їх застосування. Нео-фаззі системи використовують спеціалізовані методи обробки, наприклад, для графових структур чи складних багатовимірних даних, що робить їх придатними для аналізу складних взаємозв'язків.

Головним нововведенням стала можливість навчання системи на основі даних. Методи навчання включають:

- Градієнтний спуск для оптимізації функцій належності (рис. 1.4).
- Генетичні алгоритми для автоматичного формування правил.
- НМ для створення і адаптації нечітких множин.

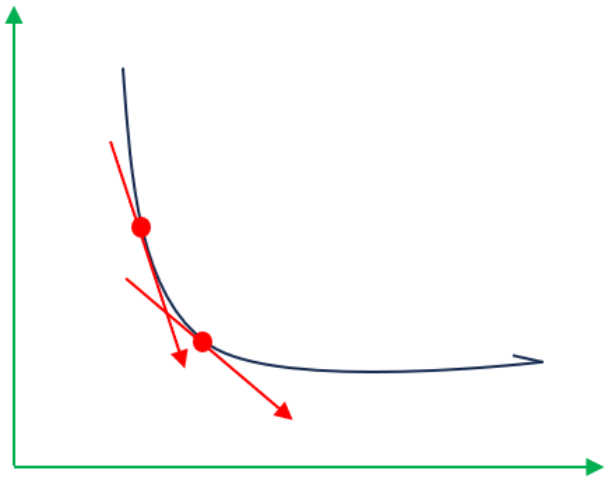

Рисунок 1.4 – Представлення градієнтного методу.

Для підвищення ефективності обчислень додаються сучасні алгоритми оптимізації, такі як ройовий інтелект, еволюційні алгоритми чи алгоритми частинок, що дозволяє системам швидко адаптуватися до змін у середовищі.

Зміни у структурі нео-фаззі систем зробили їх більш універсальними, адаптивними та продуктивними у складних середовищах. Інтеграція компонентів машинного навчання, методів оптимізації та модулів для обробки складних даних дозволила їм виходити за межі обмежень традиційних систем. Такий підхід відкрив нові можливості для їх застосування у різних сферах, таких як медицина, автоматизація, фінанси та аналіз великих даних [8].

1.4 Теоретичний порівняльний аналіз нео-фаззі систем із ададаптивними підходами

Нео-фаззі системи представляють собою значний крок уперед у сфері адаптивних підходів, поєднуючи гнучкість традиційної нечіткої логіки з можливостями автоматичного навчання та адаптації. Вони вважаються революційними завдяки здатності вирішувати задачі з високим ступенем невизначеності, складністю даних та динамічністю середовища. Однак, щоб зрозуміти, яким саме чином нео-фаззі системи перевершують інші широко розповсюджені адаптивні методи, необхідно провести їхній теоретичний аналіз і порівняти із такими підходами, як нейронні мережі, еволюційні алгоритми та байєсівські мережі.

У цьому підрозділі здійснено порівняльний аналіз нео-фаззі систем з іншими адаптивними підходами, що включає розгляд ключових принципів їхньої роботи, особливостей застосування та критеріїв оцінки, з метою виявлення унікальних переваг і обмежень кожного з методів. Результат порівняння визначить, чому і як нео-фаззі системи стали проривним рішенням у сучасній адаптивній аналітиці.

Адаптивні системи — це клас моделей та алгоритмів, які здатні змінювати свою поведінку або параметри на основі нових даних чи змін у зовнішньому середовищі. Вони характеризуються високим ступенем гнучкості та

самонавчання, що дозволяє їм ефективно вирішувати задачі навіть в умовах високої невизначеності чи динамічності.

Нейронні мережі — це адаптивні обчислювальні моделі, натхнені біологічною нервовою системою, які навчаються і адаптуються на основі даних. Вони складаються з множини пов’язаних між собою "нейронів" (вузлів), які імітують роботу людського мозку. НМ використовують оптимізаційні алгоритми, такі як градієнтний спуск, для налаштування ваг між нейронами, що дозволяє виявляти складні залежності у даних [9].

НМ мають здатність до автоматичного навчання на основі великих обсягів даних, що робить їх надзвичайно ефективними у задачах із високою складністю. Вони здатні витягувати приховані патерни, працювати з багатовимірними даними та мають високу масштабованість, але вимагають значних обчислювальних ресурсів.

Архітектура НМ (рис. 1.5) складається з вхідного шару, одного або декількох прихованих шарів та вихідного шару. Зв’язки між шарами характеризуються ваговими коефіцієнтами, які оптимізуються під час навчання. Глибокі НМ, такі як згорткові або рекурентні мережі, додають складніші структури для аналізу зображень, тексту чи часових рядів.

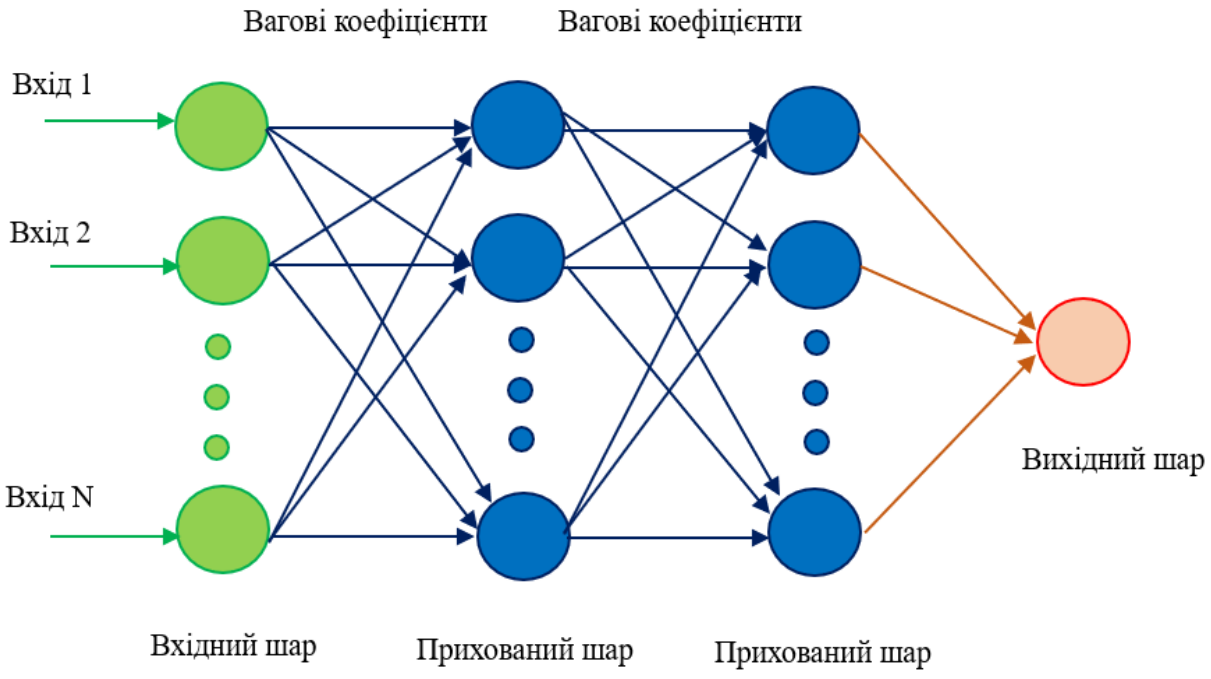

Рисунок 1.5 – Архітектура нейронної мережі.

Еволюційні алгоритми (ЕА) — це адаптивні оптимізаційні методи, які базуються на принципах природної еволюції. Вони використовують механізми мутації, кросоверу та селекції для вдосконалення рішень у популяції. ЕА відомі своєю здатністю знаходити глобальні оптимуми в задачах із багатьма локальними мінімумами.

ЕА не потребують градієнтних обчислень, що робить їх придатними для задач із нерегулярними або дискретними функціями. Вони працюють у задачах із високою кількістю параметрів, але часто потребують великих обчислювальних ресурсів і часу [10].

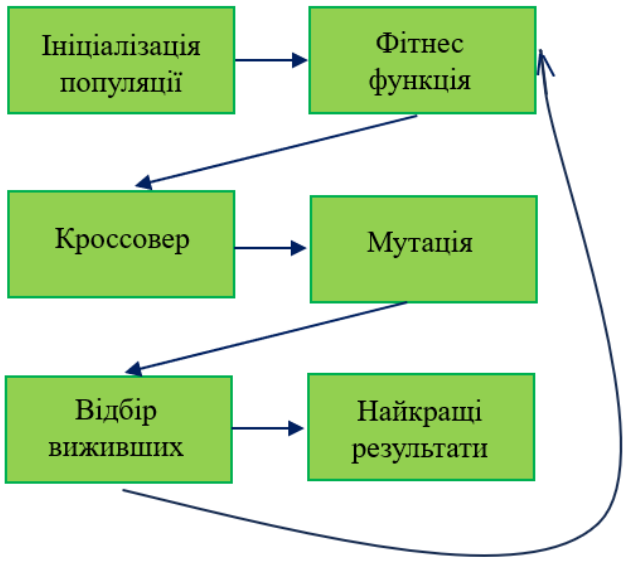

Рисунок 1.6 – Архітектура еволюційного алгоритму.

Архітектура складається з популяції рішень (індивідуумів), яка еволюціонує через ітерації. Кожне рішення характеризується хромосомою, яка відображає параметри моделі. Оператори мутації та кросоверу генерують нові хромосоми, а селекція зберігає найбільш успішні.

Байєсівські мережі (БМ) — це ймовірнісні моделі, які описують залежності між змінними у вигляді орієнтованого ациклічного графа. Вузли представляють змінні, а дуги — ймовірнісні залежності між ними. Вони використовують теорему Байєса для оновлення ймовірностей при надходженні нових даних.

БМ дозволяють працювати з неповними або невизначеними даними, моделювати причинно-наслідкові залежності та інтегрувати експертні знання. Однак навчання великих мереж може бути обчислювально складним [11].

Архітектура (рис.1.7) складається з вузлів, які представляють змінні, та орієнтованих дуг, що моделюють залежності між ними. Кожен вузол містить таблицю ймовірностей, яка задає ймовірнісний розподіл залежно від стану його батьківських вузлів.

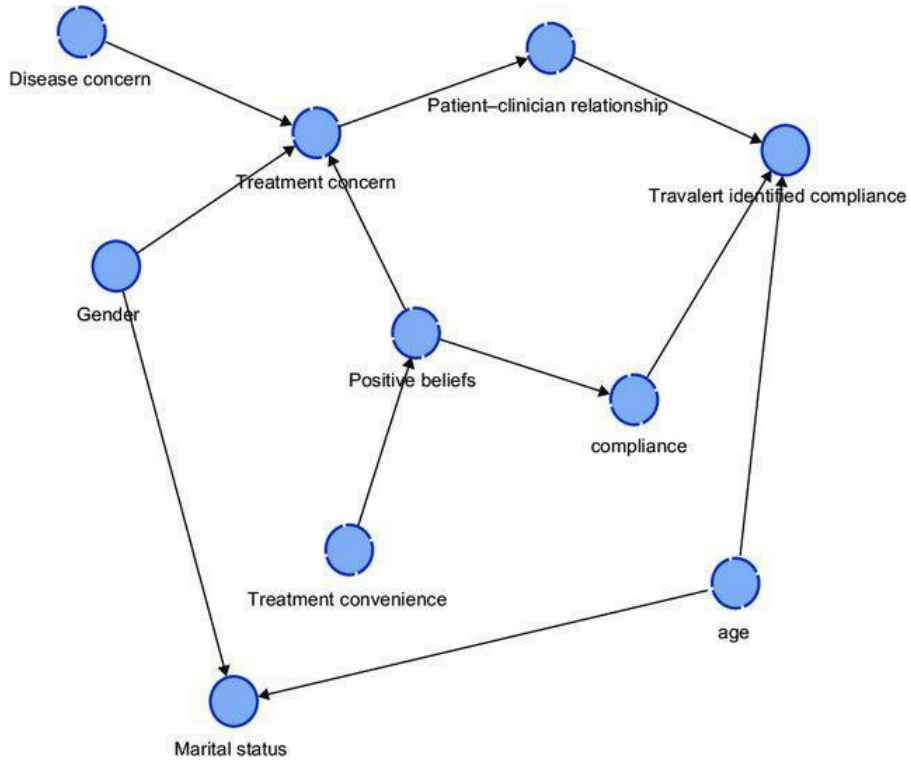

Рисунок 1.7 – Архітектура Байєсівської мережі [11].

Для проведення порівняльного аналізу адаптивних систем (табл.1.1), таких як нео-фаззі системи, нейронні мережі, еволюційні алгоритми та байєсівські мережі, було обрано шість ключових характеристик: адаптивність, точність, швидкість, обчислювальна складність, інтерпретованість та масштабованість. Параметри були визначені як основні критерії для оцінки ефективності та придатності різних підходів у вирішенні складних задач у динамічних умовах.

1. Адаптивність є важливим показником здатності системи змінювати свою поведінку або параметри на основі нових даних чи змін у середовищі. Висока адаптивність забезпечує ефективну роботу в умовах невизначеності.
2. Точність визначає, наскільки добре система здатна моделювати залежності або прогнозувати результати. Він є критично важливим у задачах, де помилки неприпустимі.
3. Швидкість оцінює, скільки часу потрібно системі для навчання чи отримання результатів, що особливо важливо для задач реального часу.
4. Обчислювальна складність характеризує ресурси, необхідні для роботи системи, включаючи час і обсяг пам'яті, що впливає на її застосовність у масштабних задачах.
5. Інтерпретованість визначає, наскільки зрозумілими є результати роботи системи для користувача, що важливо у випадках, де необхідна прозорість алгоритмів.
6. Масштабованість показує здатність системи адаптуватися до збільшення обсягів даних чи складності задачі без значного зниження продуктивності.

Таблиця 1.1

Порівняльна характеристика нео-фаззі систем з адаптивними методами.

Характеристика	**Нео-фаззі системи**	**Нейронні мережі**	**Еволюційні алгоритми**	**Байєсівські мережі**
Адаптивність	Висока	Висока	Висока	Помірна
Точність	Висока	Висока	Середня	Висока
Швидкість	Помірна	Низька	Низька	Помірна
Обчислювальна складність	Помірна	Висока	Висока	Висока
Інтерпретованість	Помірна	Низька	Низька	Висока
Масштабованість	Висока	Висока	Середня	Помірна

На основі порівняльної таблиці можна зробити наступні висновки. Усі розглянуті системи демонструють високу або помірну адаптивність, проте нео-

фаззі системи виділяються здатністю поєднувати нечітку логіку з автоматичним навчанням, що робить їх гнучкими та ефективними в задачах з високим ступенем невизначеності. НМ також демонструють високу адаптивність, але вони вимагають великих обсягів даних для ефективної роботи. ЕА є адаптивними завдяки своїм ітеративним механізмам, але їх ефективність залежить від початкових умов і параметрів. БМ мають обмежену адаптивність через залежність від фіксованої графової структури.

Нео-фаззі системи забезпечують високу точність, особливо у задачах із невизначеними або неоднозначними даними, що робить їх універсальними. НМ досягають високої точності на великих наборах даних, але потребують значних ресурсів. Точність ЕА залежить від правильної настройки параметрів, що може ускладнити їх використання. БМ демонструють високу точність у випадках, коли можна чітко моделювати причинно-наслідкові залежності.

Нео-фаззі системи демонструють помірну швидкість роботи, що робить їх придатними для практичного використання у багатьох задачах. НМ мають низьку швидкість навчання, особливо у випадку глибоких моделей, що може бути критичним недоліком у задачах реального часу. ЕА також відзначаються низькою швидкістю через велику кількість ітерацій, необхідних для досягнення оптимального рішення. БМ демонструють помірну швидкість, залежно від складності графової структури.

Нео-фаззі системи мають помірну обчислювальну складність, що зростає із збільшенням кількості правил, але залишається керованою. НМ, особливо глибокі, мають високу складність, що може обмежувати їх використання у задачах із обмеженими ресурсами. ЕА потребують значних обчислювальних ресурсів для виконання численних ітерацій. БМ демонструють високу складність у задачах із великою кількістю змінних або залежностей.

Попри те, що кожен із розглянутих адаптивних методів має свої переваги та недоліки, їх інтеграція відкриває нові горизонти у вирішенні складних задач.

Поєднання нео-фаззі систем із НМ дозволяє об’єднати гнучкість нечіткої логіки та потужність нейронних мереж у виявленні складних залежностей. У

таких гібридних моделях НМ можуть автоматично налаштовувати функції належності та оптимізувати правила, використовуючи свої можливості до навчання з великих обсягів даних.

ЕА доповнюють нео-фаззі системи своїми механізмами глобальної оптимізації. Вони можуть використовуватися для автоматичної оптимізації параметрів нечітких систем, таких як вибір форм функцій належності, створення оптимального набору правил або пошук глобального оптимуму у складних задачах. Вони застосовуються для управління автономними автомобілями, оптимізуючи правила поведінки транспортного засобу на основі складних дорожніх умов. Результат одного із досліджень, пов'язаного з автономним керуванням на основі нео-фаззі системи та ЕА (рис.1.8) за умов мокрої дороги, швидкості автомобіля та відстані до перешкоди [12].

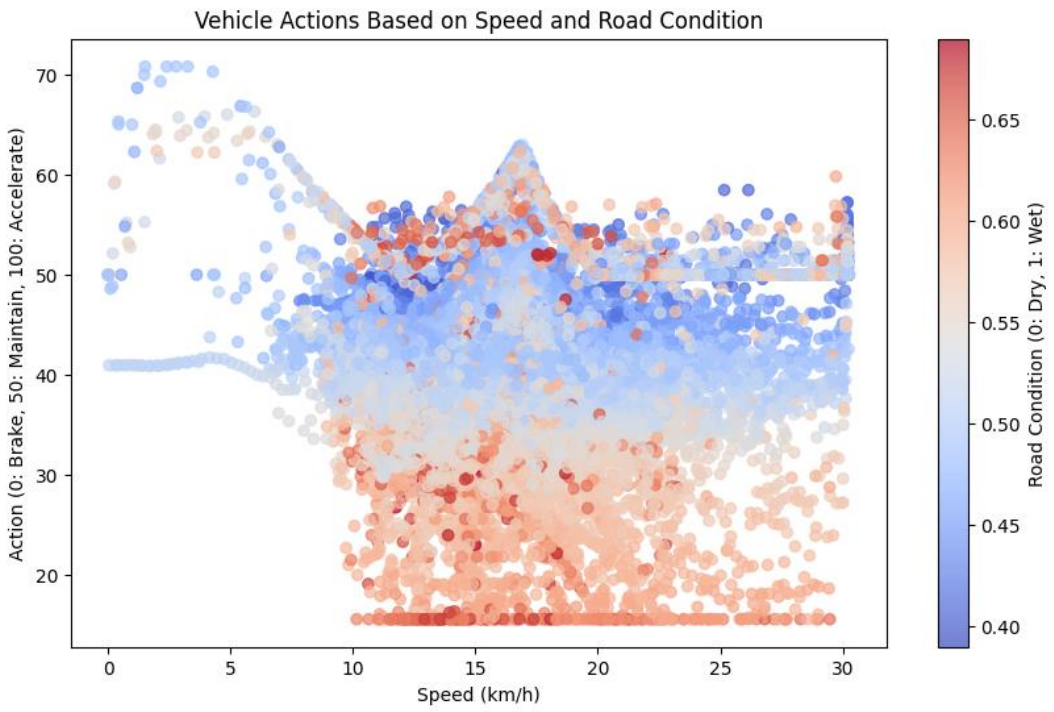

Рисунок 1.8 – Результат дослідження автономного керування.

Висновки за розділом 1

У першому розділі було досліджено розвиток систем нечіткої логіки, починаючи з основних концепцій і закінчуючи сучасними адаптивними

підходами. Розгляд включав аналіз історичних етапів розвитку, структурних компонентів фаззі систем, їхніх переваг, викликів і обмежень, а також порівняльний аналіз із сучасними адаптивними методами.

Було виявлено, що впровадження ідей нео-фаззі дозволило вирішити більшість обмежень традиційних систем. Завдяки інтеграції методів машинного навчання, таких як нейронні мережі, генетичні алгоритми та еволюційні підходи, нео-фаззі системи здатні автоматично навчатися, оптимізувати свої параметри та адаптуватися до змін середовища.

Порівняльний аналіз підтвердив, що нео-фаззі системи перевершують інші методи за ключовими критеріями, зокрема в задачах, де необхідна гнучкість та інтерпретованість.

РОЗДІЛ 2

НЕО-ФАЗЗІ СИСТЕМИ: МАТЕМАТИЧНІ ОСНОВИ ТА ШЛЯХИ ВДОСКОНАЛЕННЯ

Процес дослідження цього розділу присвячений глибшому вивченню теоретичних і практичних аспектів нео-фаззі систем, які є ключовим елементом сучасних підходів до обробки та класифікації складних даних. У попередньому розділі було розглянуто еволюцію нечіткої логіки та адаптивних систем, що створює основу для подальшого аналізу.

Акцент роботи зроблено на математичному апараті, який забезпечує функціонування нео-фаззі систем у задачах класифікації зображень, а також на огляді сучасних методів покращення точності таких систем. Особливу увагу приділено інтеграції гібридних підходів, що поєднують переваги нечіткої логіки та методів машинного навчання.

Основною метою є обґрунтування вибору методів та оптимізаційних підходів для практичної реалізації задач класифікації, які демонструють ефективність у розпізнаванні образів і вирішенні задач з високим ступенем невизначеності.

2.1 Основи роботи нео-фаззі систем у задачах розпізнавання образів

Розпізнавання образів є однією з суттєвих задач сучасного штучного інтелекту та комп’ютерного зору. Залежно від типу зображень, які потрібно обробити, та рівня складності даних, застосовуються різні підходи: від класичних ймовірнісних методів до адаптивних систем, таких як нео-фаззі.

Класичні підходи до класифікації, такі як ймовірнісні методи (SVM, Naive Bayes) або дерева рішень, базуються на припущенні про чіткі межі між класами.

Метод опорних векторів — це один із найпоширеніших підходів для класифікації та регресії. SVM базується на пошуку оптимальної гіперплощини, яка максимально розділяє класи у багатовимірному просторі. Основна мета —

знайти таку гіперплощину, яка забезпечує найбільшу відстань (маржу) між найближчими точками двох класів – опорними векторами [13].

Метод наївного Баєса базується на застосуванні теореми Баєса для класифікації. Основна ідея — оцінка ймовірності приналежності об’єкта до певного класу на основі його ознак. Метод передбачає, що всі ознаки незалежні одна від одної, що є основною "наївною" припущенням [13].

Їх основна перевага полягає у високій швидкості обчислень і точності в умовах, коли класи мають добре розділені характеристики. Проте вони виявляються недостатньо ефективними у випадках, коли межі між класами є розмитими або дані мають високий рівень невизначеності. Такі ситуації часто виникають при роботі із зображеннями природного походження, де характеристики, такі як колір, текстура або контраст, можуть значно змінюватися в залежності від освітлення, сезону чи інших факторів.

Класифікація натуральних зображень вимагає детального аналізу спектрів кольорів, яскравості та контрастності, а також характеристики кольорів.

Спектральний аналіз кольорів дозволяє визначити основні кольорові компоненти зображення. У випадку натуральних сцен спектри часто відображають домінуючі кольори, а саме зелені та коричневі кольори з високою насиченістю для лісів ($R: 34-100, G: 80-255, B: 34-120$), сині та бірюзові відтінки для моря ($R: 0-60, G: 40-180, B: 100-255$), сірі, коричневі та білі (снігові піки) для гір ($R: 80-180, G: 80-180, B: 80-180$) та відтінки сірого, коричневого, іноді червоного для будівель ($R: 100-200, G: 80-150, B: 80-150$). Частотний аналіз кольорів часто здійснюється за допомогою гістограм у колірному просторі RGB (рис.2.1) або HSV(рис.2.2).

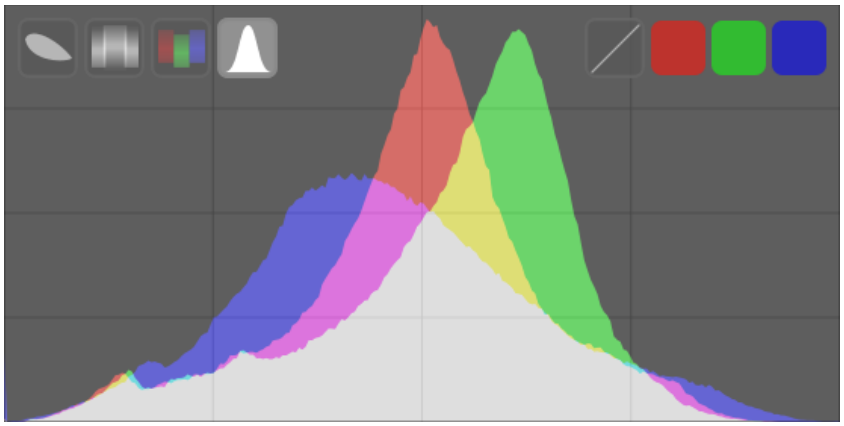

Рисунок 2.1 – Гістограма RGB [14].

Яскравість визначається як середнє значення інтенсивності пікселів у зображенні. Для натуральних сцен характерні наступні діапазони яскравості:

- Середня яскравість для лісів 50–120 (залежить від освітлення та часу доби).
- Морські сцени мають 60–180, причому яскравість води може змінюватися в залежності від хвиль і освітлення.
- Гори орієнтовно 80–200, з більш високою яскравістю для снігових зон.
- Для будівель 70–160, залежить від матеріалів і умов освітлення.

Яскравість розраховується за формулою:

$$B = \frac{R+G+B}{3} \tag{2.1}$$

де R, G, B — інтенсивності відповідних кольорових каналів.

Контрастність визначає різницю між найтемнішими і найсвітлішими частинами зображення. Високий рівень контрастності забезпечує краще виділення об'єктів та обчислюється за формулою:

$$C = \frac{\sqrt{\sum_{i=1}^{N} (I_i-\mu)^2}}{N} \tag{2.2}$$

де I_i — інтенсивність пікселів;

μ — середнє значення інтенсивності;

N — кількість пікселів;

Для лісів характерна середня контрастність, яка зазвичай варіюється в діапазоні від 0.2 до 0.5, що забезпечує помірну видимість деталей у тінях та

світлі. Морські сцени демонструють високу контрастність, яка може досягати значень до 0.7, завдяки чіткій різниці між водною поверхнею та горизонтом. У випадку гір, контрастність є середньою або високою, особливо для засніжених вершин, де яскраві снігові ділянки контрастують із темними гірськими схилами. Будівлі зазвичай характеризуються низькою або середньою контрастністю, яка варіюється в межах 0.3 – 0.6, залежно від матеріалів та умов освітлення, що визначають рівень деталізації структури фасадів.

І остання за аналізом, проте не по важливості, характеристика кольорів у зображеннях часто аналізується у колірному просторі HSV (Hue, Saturation, Value) (рис.2.2) [15].

Компонента Hue визначає основний колір у сцені. Для лісів відтінки знаходяться в діапазоні 60–120 градусів, що відповідає зеленому кольору. Морські сцени мають відтінки в межах 180–240 градусів, які відповідають синьому кольору. Для гір відтінки розташовуються у двох діапазонах: 0–60 градусів для сірого кольору та 240–360 градусів для білого, характерного для снігових вершин.

Компонента Saturation характеризує інтенсивність кольорів. Ліси відрізняються високою насиченістю, яка варіюється в межах 0.7–1.0, підкреслюючи яскравість зелених відтінків. Гори, натомість, демонструють низьку насиченість у межах 0.3–0.5, що зумовлено приглушеними тонами каменю та снігу.

Value, яке визначає загальну яскравість сцени, є особливо високим для морських сцен, де значення варіюються в межах 0.8–1.0. Компонента відображає світлі відтінки хвиль, поверхні води та горизонту під сонячним освітленням.

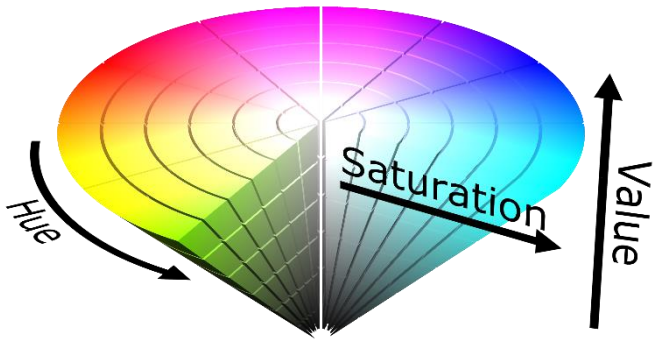

Рисунок 2.2 – Гістограма HSV [15].

Отже, виходячи із представленого опису і зробленого аналізу з попереднього розділу у задачах класифікації природних зображень, таких як дерева, гори чи морські пейзажі, нео-фаззі системи демонструють найкращі результати. Пов'язано це з їхньою здатністю працювати із розмитими класами, де інші підходи виявляються менш ефективними, тому що нечіткі логічні системи базуються на концепції нечітких множин, де елементи можуть належати до кількох класів одночасно з різним ступенем належності. Наприклад, у статті «Modified generalized neo-fuzzy system with combined online fast learning in medical diagnostic task for situations of information deficit» авторства [16] показано, що використання гібридної нео-фаззі системи дозволило досягти точності понад 90% у задачах розпізнавання складних сцен, тоді як класичні підходи демонстрували точність близько 75%.

2.2 Математичний апарат нео-фаззі систем

Математичний апарат є фундаментальною основою для реалізації нео-фаззі систем, забезпечуючи їхню здатність обробляти нечіткі дані, здійснювати висновки на основі складних залежностей і адаптуватися до змін середовища. Завдяки чіткій формалізації кожного компонента — від фаззіфікації до дефаззіфікації — нео-фаззі системи отримують можливість об'єктивно

моделювати процеси, де точні математичні залежності або недоступні, або занадто складні для традиційних підходів.

Першим буде розглянуто фаззіфікації, або перетворення чітких значень у нечіткі множини. Він є одним із найважливіших етапів у роботі. Його мета — відобразити вхідні дані, які можуть бути представлені числовими значеннями, у вигляді нечітких множин, що описують їхню належність до певних категорій. Формально, x — це чітке значення вхідної змінної, а A — нечітка множина. Ступінь належності x до множини A визначається за допомогою функції належності $\mu_A(x)$, яка має значення в інтервалі $[0,1]$. Величина $\mu_A(x)$ відображає ступінь істинності твердження, що x належить до A.

Найпоширенишими видами функцій належності є трикутна, трапецієподібна та гаусівська.

Трикутна функція належності (Triangular Membership Function) (2.3) визначається трьома параметрами: початком (a), вершиною (b) і кінцем (c). Якщо значення знаходиться лівіше початку або правіше кінця, його належність дорівнює 0. Якщо значення знаходиться між початком і вершиною, належність зростає лінійно, а між вершиною і кінцем — зменшується.

$$\mu_A(x) = \{0,\ x < a \text{ або } x > c\ \frac{x-a}{b-a},\ a \le x \le b\ \frac{c-x}{c-b},\ b < x \le c \tag{2.3}$$

Трапецієподібна функція належності (Trapezoidal Membership Function) (2.4) подібна до трикутної, але має горизонтальну вершину, що визначається чотирма параметрами: початком (a), лівою вершиною (b), правою вершиною (c) і кінцем (d). Якщо значення знаходиться в межах вершини, належність дорівнює 1. Якщо значення знаходиться між початком і лівою вершиною або між правою вершиною і кінцем, належність зростає або зменшується лінійно.

$$\mu_A(x) = \{0,\ x < a \text{ або } x > d\ \frac{x-a}{b-a},\ a \le x < b\ 1,\ b \le x \le c\ \frac{d-x}{d-c},\ c < x \le d \tag{2.4}$$

Гаусівська функція належності (Gaussian Membership Function) (2.5) має форму дзвону і визначається двома параметрами: центром (c) і шириною (σ).

Чим ближче значення до центру, тим вищий ступінь належності. Вона плавно зменшується у міру віддалення значення від центру. Завдяки своїй гладкості, гаусівська функція широко використовується в задачах, де необхідно уникнути різких переходів між ступенями належності.

$$\mu_A(x) = e^{-\frac{(x-C)^2}{2\sigma^2}} \tag{2.5}$$

Для прикладу було взято зображення (рис. 2.3) будівлі із помірною рівномірністю яскравості $x = 150$. Для такого типу зображень трикутна функція може розділяти яскравість і контрастність на три основні категорії (наприклад, "темна будівля", "світла будівля", "середня яскравість"). Значення обрано наступні $a = 100, b = 150, c = 200$.

Тоді ступінь належності $\mu_A(x)$ розрахується наступним чином.

$$\mu_A(x) = \{0, \quad x < 100 \text{ або } x > 200 \; \frac{x - 100}{150 - 100},$$
$$100 \le x \le 150 \; \frac{200 - x}{200 - 150}, \quad 150 < x \le 200$$

Оскільки $x = 150$, то $\mu_A(x) = \frac{150-100}{150-100} = 1$ що свідчить про його повну відповідність до множини А (світла будівля).

Рисунок 2.3 – Зображення для обробки.

Форма функцій належності безпосередньо впливає на якість класифікації в задачах розпізнавання образів, оскільки відображає спосіб інтерпретації вхідних даних системою. Для зображень натуральних сцен, наприклад, лісів чи

морських пейзажів, використання трикутних функцій належності може забезпечити базову точність класифікації, особливо у випадках, коли яскравість і контрастність змінюються поступово. Однак, у випадку сцен моря, де контрастність між водою, горизонтом і небом часто є високою, трикутна функція може виявитися недостатньо гнучкою для точного визначення меж між категоріями. У таких випадках гаусівська функція належності краще враховує плавні переходи кольорів і відтінків, забезпечуючи більш точні результати. Наприклад, при класифікації областей води та горизонту гаусівська функція надає більш реалістичне уявлення про варіації кольорів і яскравості.

У медичних зображеннях, таких як МРТ чи КТ, часто виникають складні структури з різкими межами між тканинами або органами. Для таких задач трапецієподібні функції належності можуть бути ефективнішими, оскільки дозволяють створювати стабільні зони для класифікації, особливо для ділянок із чіткими границями. Проте для медичних зображень із низьким контрастом, таких як судини чи м'які тканини, гаусівські функції належності допомагають краще врахувати плавні переходи в яскравості.

Штучно створені зображення часто мають особливості, що відрізняють їх від натуральних, такі як висока чіткість, неприродні кольорові переходи чи ідеальні симетрії. Для їх класифікації трикутні функції можуть бути менш ефективними, оскільки вони не враховують специфічної природи даних. У таких випадках трапецієподібні функції належності можуть забезпечити більш стабільну класифікацію завдяки можливості охоплення широких діапазонів значень.

Зазначений вище аналіз функцій належності показав, як їхня форма та параметри впливають на результати класифікації. Однак для побудови високоточних нео-фаззі систем важливо також врахувати алгоритми оптимізації параметрів цих функцій, регуляризацію для запобігання перенавчанню та методи оцінки точності роботи системи.

Оптимізація спрямована на налаштування параметрів функцій належності, правил бази знань або інших параметрів моделі, щоб мінімізувати помилки

класифікації. Одним із найпоширеніших підходів є використання градієнтного спуску.

Градієнтний спуск — це ітеративний алгоритм мінімізації функції помилки $J(\theta)$ де θ — набір параметрів системи. Основна ідея полягає в тому, щоб змінювати параметри θ у напрямку, протилежному до градієнта [17].

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta) \quad (2.6)$$

де η – швидкість навчання;

$\nabla_\theta J(\theta)$ - градієнт функції помилки за параметрами;

Регуляризація використовується для уникнення перенавчання моделі та забезпечення її стійкості до шумів у даних. У нео-фаззі системах регуляризація може стосуватися обмеження складності функцій належності (наприклад, зменшення кількості правил або розгладження функцій) або введення штрафних членів до функції помилок.

Найпопулярнішими регуляризаторами вважаються L_1 (ридж-регуляризація) та L_2 (ларсо-регуляризація). L_1-регуляризація використовує штраф, пропорційний модулю параметрів, а L_2 додає штраф за великі значення параметрів.

Оптимізація параметрів функцій належності та використання регуляризації дозволяють налаштувати нечіткі системи таким чином, щоб вони могли ефективно працювати із даними різного характеру. Однак, навіть після налаштування функцій належності, ключову роль у роботі нео-фаззі систем відіграє машина висновків, яка забезпечує прийняття рішень на основі набору правил. Саме цей компонент об'єднує вхідні дані, знання, закладені у вигляді правил, і отримує вихідні результати у формі передбачених класів.

Процес роботи машини висновків складається з наступних етапів:

1. Активізація правил визначає ступеня істинності кожного правила.
2. Агрегація результатів об'єднює виходи усіх правил для формування загального нечіткого висновку.
3. Дефаззіфікація перетворює нечіткі висновки у чітке значення.

Кожне правило має ступінь активізації, який визначається за допомогою функцій належності для відповідних нечітких множин. Наприклад, для правила $R_i : \mu_{R_i} = \left(\mu A_1^i(x_1), \mu A_2^i(x_2)\right)$,

де $\mu A_1^i(x_1), \mu A_2^i(x_2)$ - ступені належності змінних x_1, x_2 відповідним нечітким множинам A_1^i , A_2^i;

min - операція для об'єднання;

Наступним етапом нечітка машина висновків агрегує результати всіх активованих правил для формування нечіткої множини виходу. Формула описує процес агрегації:

$$\mu_B(y) = \max_i(\mu_{Ri}) \tag{2.7}$$

де $\mu_B(y)$ – об'єднана нечітка множина для вихідної змінної;

$\max_i$ - операція об'єднання;

Після формування нечіткої множини для виходу $B(y)$, необхідно перетворити її у чітке значення. Один із найпоширеніших методів — метод центру тяжіння, а саме:

$$y_{\text{чітке}} = \frac{\int y \cdot \mu_B(y)\, dy}{\int \mu_B(y)\, dy} \tag{2.8}$$

де $\int y \cdot \mu_B(y)\, dy$ - зважений середній центр нечіткої множини;

$\int \mu_B(y)\, dy$ - загальна площа під функцією належності;

Автоматизація налаштування правил логічно випливає з розгляду роботи машини висновків. Оскільки її ефективність залежить від правильності бази правил, ручне створення правил стає надто обтяжливим і не завжди оптимальним, особливо у задачах із великою кількістю вхідних параметрів чи складною структурою даних. Тому наступним природним кроком є дослідження методів автоматичного налаштування правил, які дозволяють значно покращити точність та ефективність системи.

Проблема пошуку оптимального набору правил у нео-фаззі системах полягає в тому, щоб визначити такі правила, які мінімізують похибку класифікації та максимізують загальну точність системи. Формально цю задачу

можна представити як задачу оптимізації, де можуть бути використані метрики оцінки, а саме точність, F1-міра, зменшення похибок (MSE)[18].

Точність є однією з найпростіших та найпоширеніших метрик для оцінки класифікаційних моделей. Вона визначає відсоток правильно передбачених класів відносно загальної кількості прикладів і є надійною метрикою у випадку збалансованих класів, проте у випадку дисбалансу (коли деякі класи зустрічаються частіше за інші) вона може бути неінформативною).

F1-міра є гармонійним середнім між двома іншими метриками: Precision (точністю передбачень для одного класу) та Recall (повнотою виявлення об’єктів класу). Precision показує, наскільки модель точна у передбаченні об'єктів конкретного класу, тобто який відсоток передбачених об'єктів справді належать до цього класу, а recall визначає, наскільки модель добре виявляє всі зразки конкретного класу, тобто який відсоток об'єктів певного класу було правильно передбачено.

F1-міра особливо важлива у випадках, коли класифікація є незбалансованою, оскільки дозволяє оцінювати якість передбачення незалежно від загальної кількості прикладів кожного класу.

Середньоквадратична похибка є основною метрикою для оцінки точності неперервних передбачень, але також використовується для аналізу відхилень у класифікаційних задачах, наприклад, коли модель передбачає ймовірності приналежності об'єкта до класу. Вона розраховується як середнє значення квадратів різниці між передбаченим значенням та реальним класом.

Комбінований підхід до оптимізації моделі, що базується на використанні декількох метрик оцінки, дозволяє досягти максимально точної та збалансованої класифікації. Такий підхід сприяє адаптивному налаштуванню параметрів нео-фаззі системи, зокрема функцій належності, нечітких правил та вагових коефіцієнтів. Завдяки мінімізації помилок класифікації та балансуванню між recall і precision, модель отримує здатність узагальнювати закономірності у складних даних, а не просто запам’ятовувати тренувальні приклади.

Висновки за розділом 2

У другому розділі було проведено глибокий аналіз математичних основ нео-фаззі систем та їх застосування у класифікації зображень. Розглянуто ключові процеси фаззіфікації, дефаззіфікації та формування нечітких правил, які дозволяють моделі адаптуватися до розмитих класів і невизначених даних. Було визначено, що класичні методи класифікації, такі як SVM і наївний Байєс, демонструють ефективність у випадку чітких меж між класами, але є менш придатними для задач, де перехід між категоріями є поступовим. Детально розглянуто основні функції належності (трикутна, трапецієподібна, гаусівська), а також встановлено, що гаусівські функції забезпечують кращу адаптацію до плавних змін характеристик зображень, тоді як трапецієподібні функції ефективно працюють у випадках із чітко вираженими границями.

Додатково розглянуто оптимізаційні методи, такі як градієнтний спуск для автоматичного налаштування функцій належності та регуляризацію (L1, L2) для запобігання перенавчанню. Окрему увагу приділено метрикам оцінки класифікації (Accuracy, F1-score, MSE) та їх ролі у підборі оптимального набору нечітких правил. Було показано, що використання комбінованого підходу дозволяє значно підвищити точність класифікації та зменшити похибки, особливо у випадку складних об’єктів із високим рівнем варіативності.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ НЕО-ФАЗЗІ СИСТЕМ

У цьому розділі виконується практична реалізація та оцінка ефективності нео-фаззі систем у задачах класифікації зображень. На основі попереднього теоретичного аналізу та математичного обґрунтування, він демонструє результати застосування розроблених підходів до реальних даних. У рамках роботи виконано реалізацію чотирьох підходів: класичної нечіткої системи, стандартної згорткової нейромережі, інтегрованої системи нечіткої логіки зі CNN моделлю та вдосконаленої нео-фаззі моделі.

3.1 Технічні характеристики і середовище виконання

Першим етапом практичної реалізації доцільно розглянути апаратне забезпечення та програмне середовище, які забезпечують достатню продуктивність для обробки зображень і навчання моделей машинного навчання.

Розрахунки виконувалися на ноутбуці, оснащеному процесором Intel Core i5-13500H із тактовою частотою від 2.6 ГГц до 4.7 ГГц, що забезпечувало значну швидкість обчислень. Обсяг оперативної пам’яті становив 16 ГБ, що дозволило ефективно обробляти дані та працювати з бібліотеками машинного навчання. Накопичувач типу SSD із об’ємом 512 ГБ забезпечував швидкий доступ до даних та короткий час завантаження моделей. Відеокарта Intel Iris Xe Graphics виконувала базові графічні обчислення, необхідні для роботи із задачами комп’ютерного зору. Усі обчислення проводилися під управлінням операційної системи Windows 10 x64.

Для реалізації моделей використовувалася мова програмування Python, яка є одним із найпоширеніших інструментів у сфері машинного навчання та аналізу даних. Для побудови нейронних мереж застосовувалися фреймворки TensorFlow і PyTorch, а для реалізації компонентів нечіткої логіки використовувалася бібліотека Scikit-fuzzy. Код розроблявся у середовищі VS Code з інтеграцією

Jupyter Notebook, що забезпечило зручність створення, налагодження та тестування моделей.

Датасет Intel Image Classification [19] є відкритим набором даних, створеним для задач класифікації зображень. Він широко використовується для навчання та тестування моделей машинного навчання, зокрема у задачах розпізнавання образів. Датасет представлений у вигляді зображень, розподілених за категоріями, що полегшує його використання для побудови моделей класифікації.

Датасет включає шість основних категорій зображень (рис. 3.1), кожна з яких відображає унікальні типи сцен. Категорія Buildings містить зображення різноманітних архітектурних споруд. Forest представляє природні сцени, що включають дерева та густу рослинність. Glacier охоплює зображення холодних гірських районів із засніженими вершинами, тоді як Mountain демонструє скелясті або зелені гірські пейзажі. Категорія Sea зосереджена на сценах із водними поверхнями та горизонтом, а Street відображає урбаністичні сцени, що включають дороги, тротуари та транспорт.

Рисунок 3.1 – Зображення з датасету.

Загальна кількість зображень у датасеті перевищує 25 000 файлів, які рівномірно розподілені між категоріями.

3.2 Реалізація моделей для аналізу зображень

Після характеристики технічного середовища логічним кроком є перехід до практичної реалізації підходів класифікації зображень, а саме заглиблення в процес створення та аналізу чотирьох вищезгаданих моделей. Кожен підхід супроводжується описом алгоритмів, архітектурних рішень і технічних особливостей реалізації.

Першою системою, реалізованою для аналізу зображень, є традиційна нечітка система. Для побудови системи визначено три ключові змінні: brightness (яскравість), contrast (контрастність) та output_class (вихідна класифікація). Кожна з цих змінних має відповідні функції належності. Наприклад, для змінної "яскравість" визначено дві категорії: "low" (низька) та "high" (висока), які представлені трикутними функціями належності. Аналогічно, для змінної "контрастність" задано подібні категорії. Вихідна змінна "output_class" описує класи зображень, такі як "class1" та "class2".

Кодова реалізація охоплює наступні етапи:

- Визначення функцій належності для вхідних змінних.
- Формування бази правил для прийняття рішень. Наприклад, якщо яскравість низька та контрастність низька, зображення належить до першого класу.
- Створення нечіткої системи керування та її симуляція для визначення вихідного класу на основі вхідних даних.

Лістинг 3.1. Традиційна нечітка система. Симуляція.

```
brightness = ctrl.Antecedent(np.arange(0, 256, 1), 'brightness')
contrast = ctrl.Antecedent(np.arange(0, 256, 1), 'contrast')
output_class = ctrl.Consequent(np.arange(0, 10, 1), 'output_class')


brightness['low'] = fuzz.trimf(brightness.universe, [0, 0, 128])
brightness['high'] = fuzz.trimf(brightness.universe, [128, 255, 255])
```

```
contrast['low'] = fuzz.trimf(contrast.universe, [0, 0, 128])
contrast['high'] = fuzz.trimf(contrast.universe, [128, 255, 255])
output_class['class1'] = fuzz.trimf(output_class.universe, [0, 0, 5])
```

Продовження лістингу 3.1

```
output_class['class2'] = fuzz.trimf(output_class.universe, [5, 10, 10])


rule1 = ctrl.Rule(brightness['low'] & contrast['low'], output_class['class1'])
rule2 = ctrl.Rule(brightness['high'] & contrast['high'], output_class['class2'])
control_system = ctrl.ControlSystem([rule1, rule2])
fuzzy_system = ctrl.ControlSystemSimulation(control_system)


fuzzy_system.input['brightness'] = 90
fuzzy_system.input['contrast'] = 20
fuzzy_system.compute()


if 'output_class' in fuzzy_system.output:
    print(f"Predicted class: {fuzzy_system.output['output_class']}")
else:
    print("Output variable 'output_class' not generated. Check the rules or inputs.")
```

Приклад роботи системи представлено у вигляді симуляції, де вхідні значення яскравості становлять 90, а контрастності — 20. На основі цих параметрів система виконує обчислення та виводить передбачений клас. Якщо результат не був згенерований, пропонується перевірити коректність правил або вхідних значень.

Покращеною версією традиційної нечіткої системи є модель, яка реалізує гнучкі функції належності та правила для класифікації зображень. Система працює шляхом фаззіфікації основних характеристик, таких як яскравість та контрастність, і обчислення класу на основі визначених правил.

На цьому етапі дослідження основна увага буде зосереджена на формуванні та оптимізації правил, які визначатимуть зв'язок між тренувальними даними, що не мають попередньо визначених міток.

Лістинг 3.2. Правила нечіткої системи

```
rules = [

ctrl.Rule(brightness['low'] & contrast['low'], output_class['buildings']),
ctrl.Rule(brightness['medium'] & contrast['medium'], output_class['forest']),
ctrl.Rule(brightness['high'] & contrast['low'], output_class['glacier']),
ctrl.Rule(brightness['low'] & contrast['high'], output_class['mountain']),
ctrl.Rule(brightness['medium'] & contrast['high'], output_class['sea']),
ctrl.Rule(brightness['high'] & contrast['medium'], output_class['street']), ]
```

Особливий акцент буде зроблено на розробці універсальних і гнучких правил, які враховують різноманіття вхідних даних і сприяють підвищенню точності класифікації.

Правила базуються на інтуїтивному розумінні того, як людське око сприймає різні типи сцен. Наприклад, людина легко розрізнить будівлі на фоні лісу завдяки яскравості та контрастності. Логіка правил випливає з аналізу типових візуальних ознак кожної категорії зображень. Наприклад, сцени з морем мають високу контрастність через різницю між водою та горизонтом, а льодовики вирізняються високою яскравістю через відбиття світла.

Гірські пейзажі можуть характеризуватися тінями (низька яскравість) через круті схили та одночасно високою контрастністю між тінями і освітленими ділянками (наприклад, сонячне освітлення на вершині).

Урбаністичні сцени, такі як вулиці, часто знімаються при денному світлі, що забезпечує високу яскравість. Водночас їхня контрастність є середньою, оскільки на зображеннях можуть бути як освітлені ділянки, так і затінені.

Після виконаного навчання та тестування системи доцільно перейти до аналізу отриманих результатів. Загальна точність моделі виявилася вкрай низькою — лише 17,7%, що свідчить про правильну класифікацію приблизно одного з п’яти зображень (табл. 3.1).

Найкращих результатів досягнуто для класу "Buildings" із 41% recall та 19% F1-score. Другий за якістю результат отримано для класу "Glacier", який демонструє 33% precision, 56% recall і 41% F1-score. Водночас класи "Mountain", "Sea" та "Street" майже не розпізнаються, що відображається у близьких до нуля метриках, вказуючи на значні труднощі моделі у розрізненні цих категорій або їх плутанину з іншими класами.

Таблиця 3.1

Результати тестування традиційної нечіткої системи

	precision	**recall**	**f1-score**	**support**
buildings	0.13	0.41	0.19	437
forest	0.07	0.09	0.08	474
glacier	0.33	0.56	0.41	553
mountain	0.00	0.00	0.00	535
sea	0.00	0.00	0.00	510
street	0.00	0.00	0.00	501
accuracy			0.18	3000
macro avg	0.09	0.18	0.11	3000
weighted avg	0.09	0.18	0.12	3000

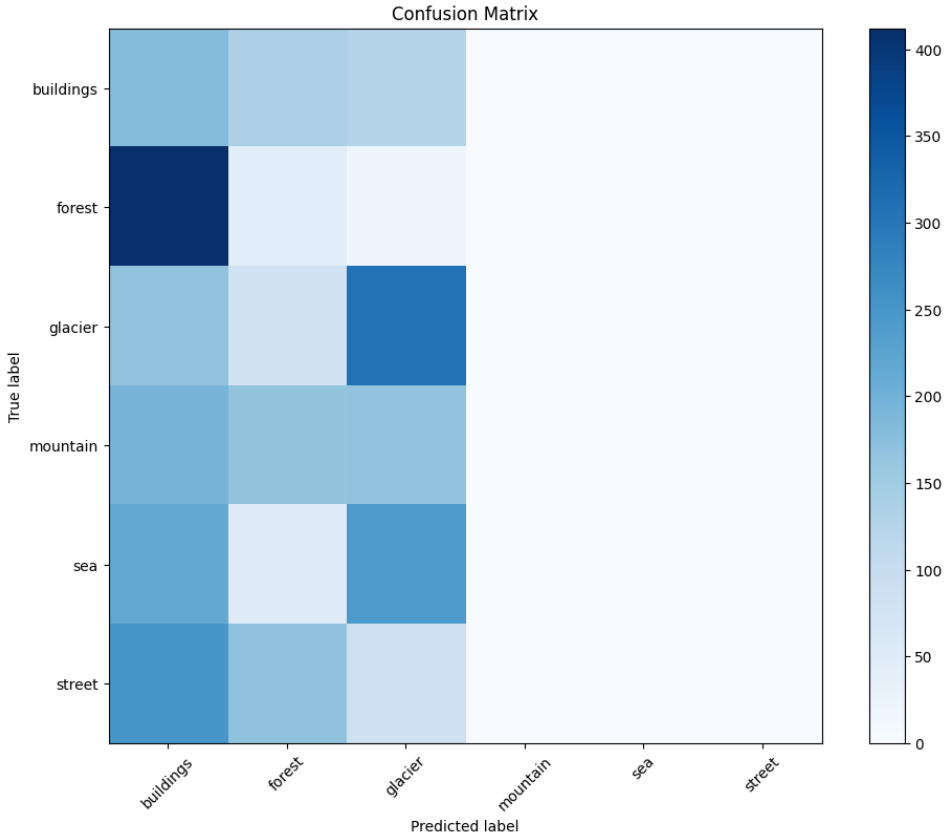

Рисунок 3.2 – Матриця плутанини для нечіткої системи.

Згідно з отриманими результатами, матриця плутанини (рис. 3.2) підтверджує тенденції, відображені в таблиці F1-score. Зокрема, клас "Buildings" демонструє найкращі результати, хоча і з певною плутаниною з іншими категоріями, що узгоджується з його recall у 41%. Аналогічно, клас "Glacier" також показує високу точність, з найбільшою кількістю правильних передбачень серед усіх класів, що відображається у 56% recall. Водночас класи "Mountain", "Sea" та "Street" майже не розпізнаються, що видно як у низьких показниках F1-score, так і в нульових діагональних елементах матриці плутанини, що вказує на суттєву плутанину з іншими категоріями.

Зі стрімким розвитком сучасних технологій та зростаючим використанням методів штучного інтелекту, доцільно зосередитися на дослідженні результатів класифікації, отриманих за допомогою згорткової нейронної мережі. Такий підхід є одним із найрозповсюджених інструментів у галузі комп'ютерного зору, що демонструє здатність ефективно аналізувати зображення, автоматично

виділяючи ключові особливості, які важко визначити вручну. Дослідження CNN у рамках цієї роботи дозволить оцінити її переваги в порівнянні з традиційними методами класифікації, такими як нечіткі системи, а також визначити можливості її адаптації до процесів еволвінга.

Згорткова мережа має наступну архітектуру: три згорткові шари, три шари пулінгу, повнозв'язний шар із регуляризацією Dropout та вихідний шар із функцією softmax для багатокласової класифікації.

Перший згортковий шар використовує 32 фільтри розміром 3×3, активацію ReLU, регуляризацію $L2 = 0.001$ та підвибірку 2×2. Другий та третій шари збільшують кількість фільтрів до 64 та 128 відповідно, зберігаючи активацію та регуляризацію. Після згорткових шарів результати перетворюються у вектор за допомогою шару Flatten. Повнозв'язний шар із 128 нейронами використовує Dropout із ймовірністю 50% для уникнення перенавчання, а вихідний шар має 6 нейронів для класифікації 6 категорій: buildings, forest, glacier, mountain, sea, street.

Лістинг 3.3. Архітектура згорткової мережі

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3), kernel_regularizer=l2(0.001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))  # 6 класів
```

Для роботи з датасетом зображення попередньо оброблялися. Було використано аугментацію даних, яка включала нормалізацію пікселів у діапазоні [0,1], горизонтальне відображення, зсув і масштабування. Завдяки цьому

збільшився обсяг тренувальних даних, що зменшило ймовірність перенавчання моделі.

Лістинг 3.4. Попередня обробка данних

```
data_gen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
train_generator = data_gen.flow_from_directory(
    directory='./seg_train',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    shuffle=True)
```

Навчання моделі відбувалося протягом п’яти епох із використанням функції втрат categorical_crossentropy та оптимізатора Adam, що дозволило поступово покращувати її здатність до класифікації. Початкова точність моделі становила 60%, що є достатньо високим результатом для базового навчання, а після завершення п’ятої епохи точність досягла 77% (рис. 3.3), що свідчить про значне покращення розпізнавання зображень порівняно з нечіткою системою, проте певні категорії залишаються складними для коректного визначення.

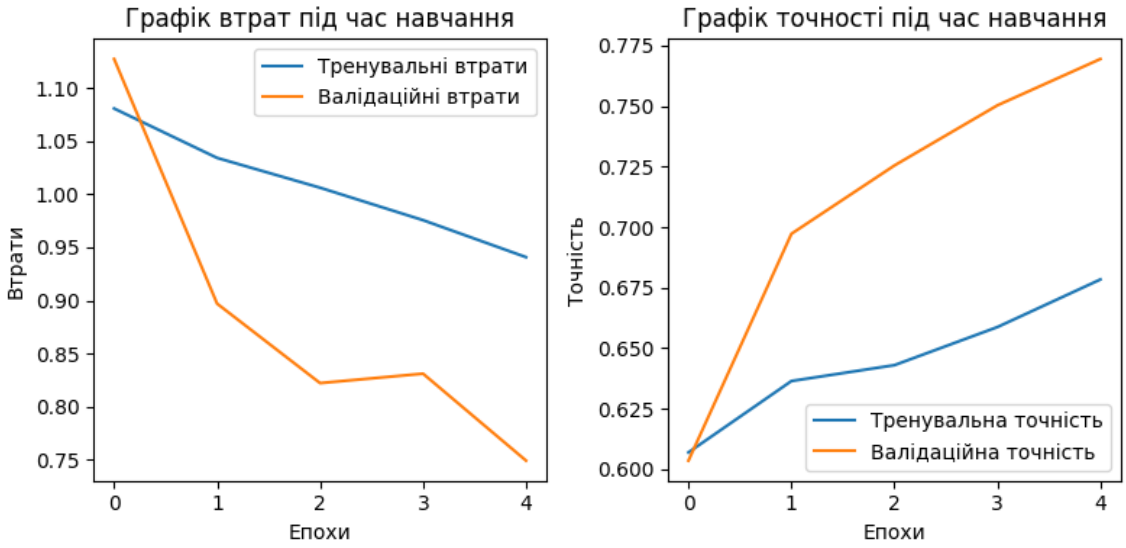

Рисунок 3.3 – Графіки втрат та точності під час навчання та валідації.

Отримані результати демонструють, що класифікація зображень значно покращується завдяки використанню згорткової нейронної мережі, однак певні категорії все ще викликають труднощі (рис. 3.4). Особливо це стосується класів із менш вираженими текстурними та контрастними характеристиками, де розмежування між категоріями є нечітким.

Рисунок 3.4 – Матриця плутанини для валідаційних данних.

Наступним етапом аналізу є логічний перехід до реалізації та тестування інтегрованої системи нечіткої логіки та згорткової нейронної мережі. Базова архітектура залишається незмінною відносно попередньої реалізації, однак ключовою особливістю цієї моделі є те, що вона використовує нечітку логіку для попередньої обробки даних, що дозволяє гнучкіше визначати приналежність зображень до певних класів, беручи до уваги невизначеність і варіації у зображеннях.

Застосування нечіткої логіки відбувається наступним чином: спочатку система аналізує ключові характеристики вхідних зображень, визначаючи їх контрастність та яскравість. На етапі попередньої обробки розділяє зображення на групи відповідно до їхніх загальних ознак, що, в свою чергу, приводить до більш точного навчання та узагальнення отриманих закономірностей.

Для цього проводиться аналіз яскравості та контрастності кожного зображення. Яскравість визначається як середнє значення пікселів зображення, тоді як контрастність розраховується на основі стандартного відхилення піксельної інтенсивності. Завдяки цьому система отримує числові показники, що відображають загальні візуальні характеристики об’єкта, дозволяючи на ранньому етапі визначити його належність до певного класу.

Лістинг 3.5 Функція інтегрування нечітких критеріїв для класифікації зображень за певними характеристиками

```
def preprocess_image(image_path):
    image = Image.open(image_path).convert('L')
    image = image.resize((128, 128))
    pixel_values = np.array(image)
    brightness = np.mean(pixel_values)
    contrast = np.std(pixel_values)
    return brightness, contrast
```

Модель навчалася протягом 10 епох і отримала загальну точність тестових на рівні 82% в процесі навчання, проте на валідаційних данних отримані результати покращилися на 10%. На відміну від класичної CNN, отримані результати свідчать про кращу узагальнюючу здатність моделі, оскільки нечітка логіка допомагає зменшити випадки неправильних класифікацій.

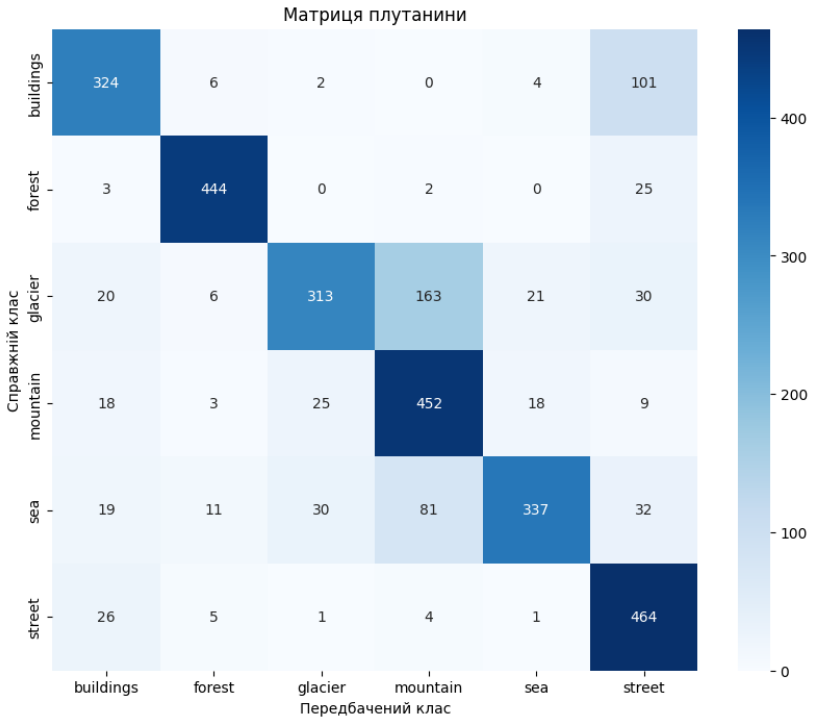

Рисунок 3.5 – Матриця плутанини для валідаційних данних.

Порівнюючи отриману матрицю плутанини з попереднім результатом для звичайної згорткової нейромережі, можна помітити суттєве покращення класифікації для тих класів, що раніше визначилися більш складно. Проте, навіть із використанням нечіткої логіки отримані результати мають відхилення та погіршення.

Останнім етапом реалізації систем штучного інтелекту полягає у створенні нео-фаззі системі для класифікації зображень. У створеній моделі нео-фаззі логіка використовується як окремий шар у згортковій нейромережі, що дозволяє

об’єднати переваги CNN (витягування ознак) та нечіткої логіки (адаптація до розмитих даних).

Лістинг 3.6 Нео-фаззі шар

```
class NeoFuzzyLayer(Layer):
    def __init__(self, num_rules=5, **kwargs):
        self.num_rules = num_rules
        super(NeoFuzzyLayer, self).__init__(**kwargs)
    def build(self, input_shape):
        self.centers = self.add_weight(name='centers',
                                       shape=(input_shape[-1], self.num_rules),
                                       initializer='uniform',
                                       trainable=True)
        self.widths = self.add_weight(name='widths',
                                      shape=(input_shape[-1], self.num_rules),
                                      initializer='ones',
                                      trainable=True)

        self.rules_weights = self.add_weight(name='rules_weights',
                                             shape=(input_shape[-1], self.num_rules),
                                             initializer='uniform',
                                             trainable=True)

        super(NeoFuzzyLayer, self).build(input_shape)
```

Ключовими параметрами моделі є координати центрів функцій належності, їхні ширини, а також вагові коефіцієнти, які визначають вплив кожного правила на обчислення вихідного значення. Центри функцій належності визначають опорні точки розподілу вхідних даних у просторі ознак, ширини контролюють ступінь узагальнення та гнучкість моделі, а вагові коефіцієнти впливають на силу кожного правила під час комбінування нечітких висновків. Оптимізація цих параметрів дозволяє покращити адаптивність системи та її здатність до коректної інтерпретації складних вхідних даних.

Для покращення продуктивності мережі було додано BatchNormalization, який стабілізує градієнти під час навчання, що сприяє швидшій та ефективнішій конвергенції та використано GlobalAveragePooling2D для зменшення ризику перенавчання, оскільки модель навчається узагальнювати просторові особливості без необхідності великої кількості параметрів.

За результатами (рис. 3.6) навчання покращила валідаційна точність до 89%, навчання моделі стало більш стабільним після 5-6 епохи та зменшилися втрати відповідно (табл. 3.2).

Таблиця 3.2

Результати навчання нео-фаззі системи

Epoch	Train Accuracy	Val Accuracy	Train Loss	Val Loss
1	36.79%	39.87%	1.50	1.62
5	81.85%	83.07%	0.52	0.46
10	85.98%	84.70%	0.40	0.44
20	92.67%	89.07%	0.20	0.30

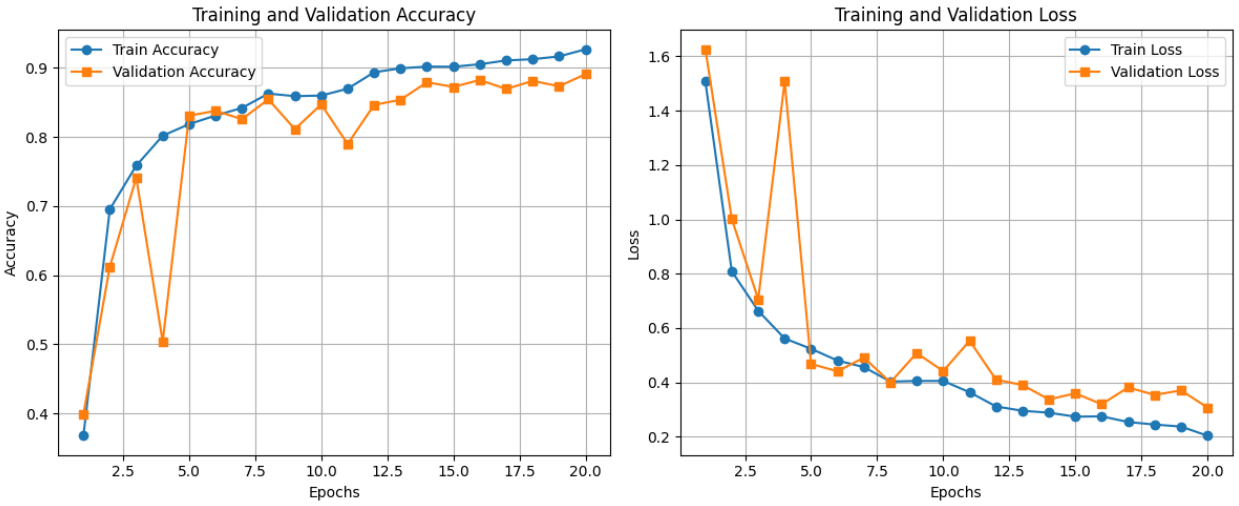

Рисунок 3.6 – Результати навчання та валідації нео-фаззі.

3.3 Порівняльна характеристика реалізації систем із нечіткою логікою

Заключним етапом дослідження є порівняльна характеристика реалізованих моделей, що поєднують методи нечіткої логіки та глибокого навчання. Основна мета цього аналізу — визначити, наскільки ефективним є застосування нечітких підходів для класифікації зображень та які саме переваги забезпечує нео-фаззі підхід у порівнянні зі стандартною згортковою нейронною мережею та її модифікованими версіями (табл 3.3).

Таблиця 3.3

Порівняльна таблиця реалізованих систем

Модель	Точність на тестових даних	Основні особливості
Звичайна CNN	77%	Висока точність для простих класів, але значна плутанина у складних випадках.
CNN + нечітка логіка	85%	Використання нечітких правил допомагає зменшити помилки класифікації.
Гібридна нео-фаззі система	89%	Найкраще працює із розмитими межами класів, адаптивно підлаштовуючи параметри.

Найвищий рівень точності (89%) був досягнутий завдяки поєднанню можливостей класичної нейронної мережі з гнучкими принципами нечіткої логіки. Нео-фаззі система здатна класифікувати зображення, що мають нечіткі кордони між класами, оскільки вона використовує адаптивні функції приналежності, що змінюються в процесі навчання. На відміну від класичної нейромережі, нео-фаззі модель здатна ефективніше узагальнювати знання, що дозволяє їй краще працювати на реальних даних, де варіативність зображень значно вища, ніж у тренувальному наборі.

Аналіз матриць плутанини підтверджує висновки щодо підвищеної ефективності нео-фаззі системи. У порівнянні зі стандартною CNN, нео-фаззі модель значно зменшує плутанину між класами та покращує розпізнавання категорій із низьким контрастом або схожими текстурами.

- Зменшення плутанини між класами forest та street, які раніше були найбільш проблемними для класичної CNN.

- Покращення класифікації класу mountain, який у стандартній моделі мав значні помилки.
- Значне зростання точності для класу sea, що вказує на ефективність нечітких функцій приналежності у визначенні специфічних характеристик зображень.

Висновки за розділом 3

У третьому розділі дипломного проєкту було здійснено практичну реалізацію та аналіз ефективності нео-фаззі систем у задачах класифікації зображень. Досліджено чотири підходи: класичну нечітку систему, стандартну згорткову нейронну мережу, інтегровану систему нечіткої логіки з CNN та вдосконалену нео-фаззі модель. Реалізація кожного з підходів супроводжувалася детальним аналізом архітектури, алгоритмів та отриманих результатів. На основі проведених експериментів встановлено, що нео-фаззі система демонструє найкращу точність класифікації, досягнувши 89% точності на тестових даних, що є значним покращенням у порівнянні зі стандартною CNN (77%) та інтегрованою нечіткою моделлю (85%). Це пояснюється тим, що нео-фаззі підхід адаптивно налаштовує функції приналежності, забезпечуючи більшу гнучкість у класифікації складних випадків. Аналіз матриць плутанини показав, що дана модель краще розпізнає класи з нечіткими межами, наприклад, гори, вулиці та море, які у стандартній CNN викликали найбільше плутанини.

ВИСНОВКИ

У першому розділі роботи було розглянуто теоретичні основи нечіткої логіки, її історичний розвиток та основні компоненти фаззі-систем. Проведено огляд традиційних нечітких моделей, зокрема методів Мамдані та Сугено, визначено їхні переваги та обмеження. Особливу увагу приділено аналізу нео-фаззі систем, які об'єднують адаптивні можливості машинного навчання з інтерпретованістю нечіткої логіки. Також здійснено порівняльний аналіз з іншими адаптивними методами: нейронними мережами, еволюційними алгоритмами та байєсівськими мережами

Опис дослідження показує, що актуальність теми обумовлена зростаючою потребою в ефективному аналізі зображень із високим рівнем невизначеності, де класичні методи демонструють недостатню точність.

У другому розділі було розглянуто математичний апарат, що лежить в основі нео-фаззі систем. Детально описано етапи фаззіфікації, побудову функцій належності (трикутних, трапецієподібних, гаусівських), дефаззіфікацію, а також механізм формування правил та обчислення вихідного результату через машину висновків.

Досліджувалось питання точності та гнучкості різних типів функцій належності залежно від типу зображень. Було встановлено, що гаусівські функції найкраще підходять для класифікації сцен із поступовими змінами кольору (наприклад, море чи ліс), тоді як трапецієподібні — для зображень із чіткими межами, таких як будівлі або медичні знімки.

Також досліджено алгоритми оптимізації, зокрема градієнтний спуск та регуляризаційні методи (L1, L2), які дозволяють уникнути перенавчання моделі. Особливу увагу приділено вибору метрик для оцінки якості класифікації, таких як точність (accuracy), F1-міра та середньоквадратична похибка.

У третьому розділі реалізовано практичну частину роботи — створення та тестування системи на основі датасету Intel Image Classification. Було побудовано чотири моделі: класична нечітка система, CNN, інтегрована модель CNN+фаззі логіка та нео-фаззі система.

Проведено тестування моделей на однаковому наборі даних. Найвищу точність показала нео-фаззі система — вона перевищила як традиційні нечіткі моделі, так і базову CNN у випадках класифікації зображень із нечіткими ознаками. Також доведено її кращу інтерпретованість та здатність працювати з обмеженими даними. У моделі враховано специфіку різних типів сцен — будівлі, ліси, гори, море, вулиці, льодовики — і побудовано відповідну базу правил.

У результаті дослідження було створено нео-фаззі систему, здатну до високоточного аналізу зображень у складних умовах. Поєднання адаптивного навчання з нечіткою логікою забезпечило високу гнучкість, інтерпретованість рішень та адаптацію до нових даних. Результати підтверджують ефективність підходу, особливо для задач із нечіткими або частково визначеними класами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. GeeksforGeeks. Fuzzy Logic | Introduction [Електронний ресурс]. – режим доступу: https://www.geeksforgeeks.org/fuzzy-logic-introduction/ (дата звернення – 22.02.2025).
2. Kruse R., Gebhardt J., Klawonn F. Foundations of fuzzy systems. – 1994. – XII, 265 p.
3. Klir G., Yuan B. Fuzzy sets and fuzzy logic. Vol. 4. – New Jersey : Prentice Hall, 1995. – P. 1–12.
4. Fuzzy sets and pattern recognition [Електронний ресурс]. – режим доступу: https://www.cs.princeton.edu/courses/archive/fall07/cos436/HIDDEN/Knapp/fuzzy004.htm (дата звернення – 25.02.2025).
5. Arif M. F., Anoraga B., Handoyo S., Nasir H. Algorithm apriori association rule in determination of fuzzy rule based on comparison of fuzzy inference System (FIS) Mamdani Method and Sugeno Method // Business Management and Strategy. – 2016. – Vol. 7, № 1. – P. 103. – режим доступу: https://doi.org/10.5296/bms.v7i1.9410 (дата звернення – 28.02.2025).
6. Cherkassky V. Fuzzy Inference Systems: A Critical Review // Studies in Fuzziness and Soft Computing. – 1998. – P. 177–197. – режим доступу: https://doi.org/10.1007/978-3-642-58930-0_10 (дата звернення – 02.03.2025).
7. Chopra S., Dhiman G., Sharma A., Shabaz M., Shukla P., Arora M. Taxonomy of Adaptive Neuro-Fuzzy Inference System in Modern Engineering Sciences // Computational Intelligence and Neuroscience. – 2021. – режим доступу: https://doi.org/10.1155/2021/6455592 (дата звернення – 05.03.2025).
8. Da Silva A., Caminhas W., Lemos A., Gomide F. A fast learning algorithm for evolving neo-fuzzy neuron // Applied Soft Computing. – 2014. – Vol. 14. – P. 194–209. – режим доступу: https://doi.org/10.1016/j.asoc.2013.03.022 (дата звернення – 08.03.2025).
9. Ладуба М. Нейромережі пишуть книги та рятують життя: що таке нейронна мережа і як вона працює [Електронний ресурс]. – режим доступу:

https://mc.today/uk/shho-take-nejronna-merezha/ (дата звернення – 11.03.2025).

10. Ganguli R. Genetic Algorithms and Machine Learning a Qualitative Approach [Електронний ресурс]. – режим доступу: https://www.researchgate.net/publication/354683575_Genetic_Algorithms_and_Machine_Learning_a_Qualitative_Approach (дата звернення – 13.03.2025).
11. Berdeaux G. Identification of noncompliant glaucoma patients using Bayesian networks and the Eye-Drop Satisfaction Questionnaire // Clinical Ophthalmology. – 2010. – P. 1489. – режим доступу: https://doi.org/10.2147/opth.s11818 (дата звернення – 15.03.2025).
12. Self-driving car data [Електронний ресурс]. – режим доступу: https://www.kaggle.com/datasets/mertsonmezer/self-driving-car-data/data (дата звернення – 17.03.2025).
13. Obinna W. Comparing naïve Bayes and SVM for text classification [Електронний ресурс]. – режим доступу: https://medium.com (дата звернення – 19.03.2025).
14. darktable user manual – Гістограми [Електронний ресурс]. – режим доступу: https://docs.darktable.org/usermanual/development/uk/module-reference/utility-modules/shared/scopes/ (дата звернення – 21.03.2025).
15. Файл:HSV color solid cone.png — Вікіпедія [Електронний ресурс]. – режим доступу: https://uk.m.wikipedia.org/wiki/Файл:HSV_color_solid_cone.png (дата звернення – 24.03.2025).
16. Bodyanskiy Y., Chala O., Kasatkina N., Pliss I. Modified generalized neo-fuzzy system with combined online fast learning in medical diagnostic task for situations of information deficit // Mathematical Biosciences & Engineering. – 2022. – Vol. 19, № 8. – P. 8003–8018. – режим доступу: https://doi.org/10.3934/mbe.2022374 (дата звернення – 27.03.2025).
17. Donges N. Gradient Descent in Machine Learning: A Basic Introduction [Електронний ресурс]. – режим доступу: https://builtin.com/data-science/gradient-descent (дата звернення – 30.03.2025).

18. Classification: accuracy, completeness, precision and related indicators [Електронний ресурс]. – режим доступу: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=ua (дата звернення – 07.04.2025).
19. Intel Image Classification [Електронний ресурс]. – режим доступу: https://www.kaggle.com/datasets/puneet6060/intel-image-classification (дата звернення – 17.04.2025).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

ТКАЧЕНКА КИРИЛА СЕРГІЙОВИЧА
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«ДОСЛІДЖЕННЯ ТА ПОКРАЩЕННЯ ТОЧНОСТІ КЛАСИФІКАЦІЇ НЕО-ФАЗЗІ СИСТЕМ ДЛЯ ЗАДАЧ РОЗПІЗНАВАННЯ ОБРАЗІВ»**

керівник роботи **Бикова Тетяна Володимирівна, доцент кафедри КСР, кандидат технічних наук**
затверджені наказом по університету від **16.04.*2025 року № 4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити:
   1. Дослідити еволюцію та принципи роботи систем, переваги та недоліки.
   2. Вивчити адаптивні підходи (нейронні мережі, еволюційні алгоритми, байєсівські мережі) для обробки складних даних.
   3. Виконати теоретичне порівняння нео-фаззі систем з іншими підходами.
   4. Розглянути математичний апарат нео-фаззі систем: фаззіфікація, функції належності, дефаззіфікація.
   5. Проаналізувати особливості обробки зображень для задач класифікації (яскравість, контрастність, HSV).
   6. Реалізувати традиційну фаззі систему та гібридну модель нечіткої логіки.
   7. Побудувати нео-фаззі модель для класифікації зображень.
   8. Провести оптимізацію параметрів функцій належності (градієнтний спуск, регуляризація).
   9. Провести порівняльний аналіз моделей за метриками (точність, recall, F1-score).
   10. Підготувати підсумкові висновки та оформити результати у кваліфікаційній роботі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи та погодження з керівником	05.09.2024 - 09.09.2024
2	Збір літератури, аналіз джерел по нечітким системам та нео-фаззі підходам	10.09.2024 - 09.10.2024
3	Дослідження адаптивних систем: НМ, ЕА, БМ, порівняння з нео-фаззі	10.10.2024 - 09.11.2024
4	Формалізація математичного апарату: фаззіфікація, функції належності, дефаззіфікація	10.11.2024 - 24.11.2024
5	Огляд методів обробки зображень та їх характеристик (HSV, контраст, яскравість)	25.10.2024 - 08.11.2024
6	Реалізація традиційної нечіткої системи класифікації	09.11.2024- 01.02.2025
7	Реалізація згорткової нейронної мережі (CNN) для класифікації	01.03.2025- 30.04.2025-
8	Побудова інтегрованої системи нечіткої логіки та CNN	01.03.2025- 30.04.2025-
9	Реалізація та налаштування нео-фаззі моделі (з NeoFuzzy Layer)	01.03.2025- 30.04.2025-
10	Тестування моделей, побудова матриць плутанини, обчислення метрик	01.04.2025- 30.04.2025-
11	Оптимізація функцій належності, застосування регуляризації	01.04.2025- 30.04.2025-
12	Аналіз результатів, формулювання висновків, підготовка до захисту	01.05.2025- 30.05.2025-
13	Оформлення пояснювальної записки, додатків та літератури.	01.05.2025- 30.05.2025-
14	Представлення кваліфікаційної роботи керівнику та рецензенту	01.05.2025- 30.05.2025-

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент Ткаченко К.С.
ініціали, прізвище
підпис

Керівник роботи Бикова Т.В.
ініціали, прізвище
підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання

на розробку програмного виробу «Дослідження та покращення точності класифікації нео-фаззі систем для задач розпізнавання образів»

1.	Введення	1.1. Назва: Інтелектуальна нео-фаззі система класифікації природних зображень. 1.2. Галузь застосування: комп'ютерний зір, інтелектуальний аналіз зображень, геоінформатика, навчальні дослідження у сфері штучного інтелекту.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра №4101-5/962 від «16» квітня 2025 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: дослідження системи класифікації зображень із використанням нео-фаззі моделей. 3.2. Розробка призначена для автоматичної класифікації зображень природних сцен за допомогою інтелектуальної системи на основі нео-фаззі логіки. Система дозволяє розпізнавати категорії зображень (ліс, море, гори, будівлі, льодовик, вулиця) навіть при наявності обмеженої кількості навчальних даних. 3.3. Вихідні дані розробки: модельні дані класифікації, статистичні дані якості кваліфікації.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик:немає 4.2. Вимоги до надійності: система повинна забезпечувати точність та достовірність визначення інформативності параметрів стану зображення не

		гіршу за існуючі методи та програмне забезпечення, що використовуються в аналогічних задачах класифікації природних сцен. 4.3.Вимоги до умов експлуатації: немає 4.4. Вимоги до складу і параметрів технічних засобів: звичайне обчислювальне обладнання, ПК з інстальованим Python та середовищем Jupyter Notebook. 4.5. Вимоги до інформаційної та програмної сумісності: немає 4.6. Вимоги до маркування та упаковки: немає 4.7. Вимоги до транспортування і зберігання: на звичайних носіях інформації 4.8. Спеціальні вимоги: немає.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Інтелектуальна нео-фаззі система класифікації природних зображень» вважати: 1) Справжнє Технічне завдання на розробку виробу представлено у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи. 2) Методику оцінювання інформативності параметрів стану, реалізовану у вигляді аналізу точності класифікації, теплових карт (heatmaps) та статистичних показників (точність, повнота, F1-міра), що представити у розділі 3 пояснювальної записки до кваліфікаційної роботи. 3) Опис виробу включаючи його призначення, архітектуру, принцип роботи, алгоритмічну реалізацію та результати тестування, що також представити у розділі 3 пояснювальної записки до кваліфікаційної роботи.
6.	Вимоги до техніко-економічних показників	Програмною документацією до виробу «Інтелектуальна нео-фаззі система класифікації природних зображень» вважати:

		1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Методику оцінювання інформативності параметрів стану, реалізовану у вигляді аналізу точності класифікації, теплових карт (heatmaps) та статистичних показників (точність, повнота, F1-міра), що представити у розділі 3 пояснювальної записки до кваліфікаційної роботи. 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)	
7.	Стадії і етапи розробки	Дата	Назва етапу
		до 15 жовтня 2024 до 25 листопада 2024	Аналіз практики предметної області та огляд сучасних підходів у галузі класифікації зображень із використанням нечіткої логіки та машинного навчання.
		від 2 грудня 2024 до 16 грудня 2024	Розробка теоретико-методологічної бази дослідження, включаючи формулювання наукової гіпотези, вибір підходів та методів дослідження.
		від 2 грудня 2024 до 2 січня 2025	Побудова математичної моделі нео-фаззі системи, розгляд фаззіфікації, дефаззіфікації, формування правил, машин висновків. Вибір функцій належності, опис алгоритмів оптимізації та регуляризації.
			Обґрунтування вибору апаратного забезпечення та

			програмного середовища (Python, TensorFlow, Scikit-Fuzzy, VS Code). Ознайомлення з набором даних Intel Image Classification та попередня обробка зображень.
		від 3 січня 2025 до 30 березня 2025	Створення традиційної нечіткої системи, CNN-моделі, інтегрованої моделі з нечіткою логікою та остаточної нео-фаззі системи. Розробка коду, навчання моделей, тестування на практичних прикладах.
		від 11 лютого 2025 до 27 травня 2025	Проведення експериментів, побудова метрик точності (Accuracy, F1-score, MSE), інтерпретація результатів. Порівняння точності та продуктивності різних моделей, виявлення переваг нео-фаззі підходу.
			Оформлення результатів. Написання пояснювальної записки.
		від 31 березня 2025 до 27 квітня 2025	Представлення кваліфікаційного проєкту керівнику кваліфікаційної роботи та рецензенту.

		від 1 травня 2025 до 28 травня 2025 30 травня 2025	
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Захист розробленої моделі провести на засіданні Атестаційної комісії. 3. Пояснювальну записку подати в електронному вигляді в 1 примірнику.	

Виконавець
студент групи КІ- 41
Ткаченко К.С.

_____

Замовник
доцент кафедри, кандидат
технічних наук Бикова Т.В.

_____

Додаток В

Програма і методика випробувань програмного виробу

«Дослідження та покращення точності класифікації нео-фаззі систем для задач розпізнавання образів»

1 Об'єкт випробувань

1. Назва програмного виробу: «Інтелектуальна нео-фаззі система класифікації природних зображень»
2. Галузь застосування: Комп’ютерний зір, інтелектуальний аналіз зображень, геоінформатика, навчальні дослідження у сфері ШІ

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації системи вимогам технічного завдання (Додаток Б до пояснювальної записки до кваліфікаційної роботи), зокрема точності, інформативності та класифікаційної здатності нео-фаззі системи.

3. Загальні положення

1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3. Обсяг випробувань

Приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1. підтримка ОС: Windows, Linux, MacOS;

2. забезпечення точності та достовірності результатів не гіршої за існуючі методи;
3. захист від некоректних дій користувача;
4. сумісність із Python середовищем та Jupyter Notebook;
5. робота на звичайному ПК;
6. зберігання на стандартних носіях інформації.

Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Дослідження та покращення точності класифікації нео-фаззі систем для задач розпізнавання образів» вважати:

1. Технічне завдання на розробку (Додаток Б).
2. Методику розрахунку інформативності змінних стану (описано в розділі 3 ПЗ).
3. Опис програмного виробу (архітектура, логіка, принципи роботи та результати) — розділ 3 ПЗ.
4. Програмний код і структура експериментів — Додаток Г.

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідний

1. Обчислювальна система з ОСWindows/Linux/MacOS
2. Python 3.x
3. Середовище Jupyter Notebook
4. Доступ до набору зображень (наприклад, Intel Image Classification Dataset)

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.
3. Перевірку комплектності складу технічних і програмних засобів.
4. Методику проведення перевірок на 1 етапі випробувань.
5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;
2. перевірку ступеня виконання функціональних вимог до програми;
3. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

1. Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux та MacOS.
2. Для роботи необхідний Jupyter Notebook з мовою програмування python, версії не нижчої ніж 3.0
3. Порядок проведення випробувань:
    3.1. Підготовка до випробувань полягає в запуску середовища Jupyter Notebook, попередньому встановленні бібліотек (tensorflow, sklearn, matplotlib, numpy, seaborn, cv2, PIL, shap, gradcam) та завантаженні підготовленого датасету з класифікації зображень (Intel Image Classification);
    3.2. Після запуску виконавчих комірок навчання моделі проводиться на навчальному наборі даних протягом визначеної кількості епох;
    3.3. Після завершення навчання здійснюється тестування моделі на відкладеній тестовій вибірці;
    3.4. Після проведеного тестування результати тестування виводяться у вигляді точності класифікації, графіків помилок, матриць змішування та візуалізацій heatmap.

    Для проведення випробувань пропонується тест 1, тест 2 та тест 3.

Тест 1

1. перевірка виконання програми запуском Запуск Jupyter Notebook diplomas.ipynb у середовищі з попередньо встановленими бібліотеками.

2. отримання відповіді: усі комірки виконуються без помилок, виводяться графіки та результати класифікації.

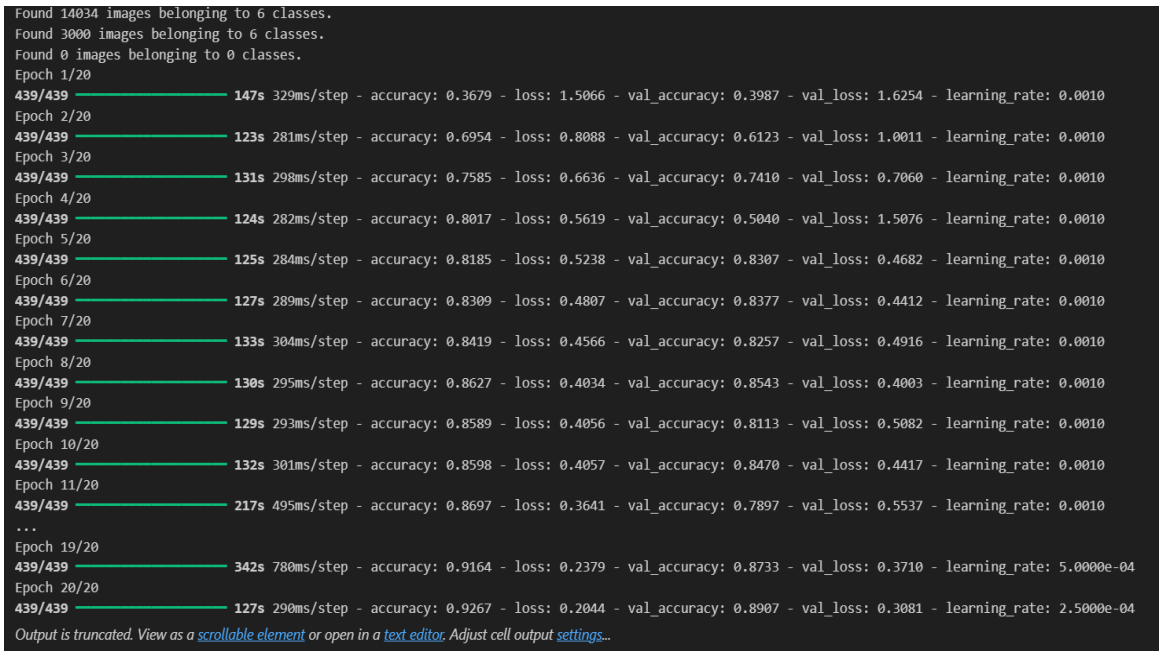

Рисунок. В.1 Тест 1.

Тест 2

1. запуск блоку побудови та навчання моделі з використанням тренувальної вибірки.
2. отримання відповіді: модель успішно проходить етап навчання протягом 20 епох, спостерігається зменшення функції втрат (loss) та підвищення точності (accuracy).

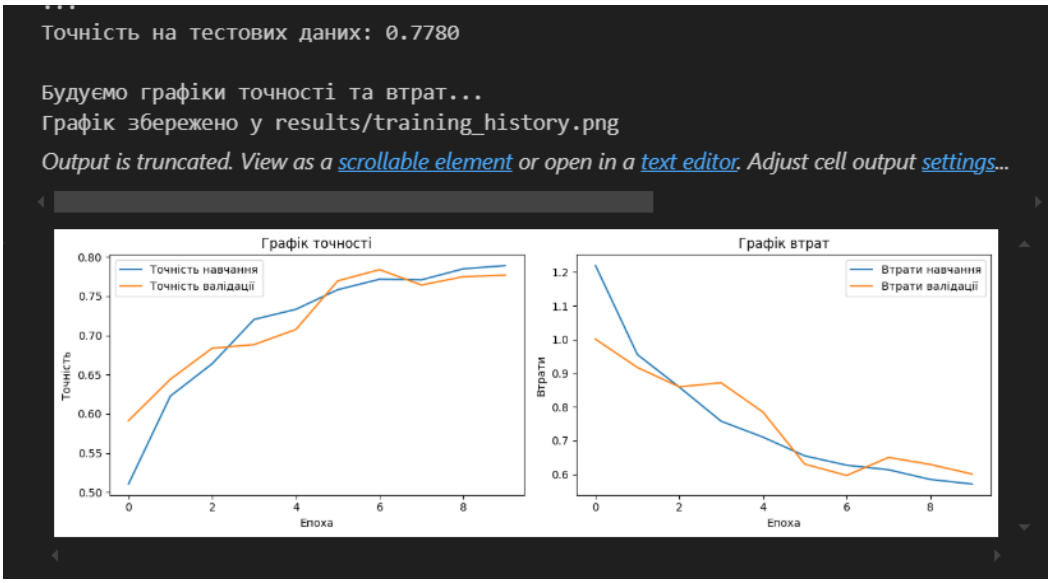

Рисунок В.2 Тест 2.

Тест 3

1. запуск блоку передбачення (інференсу) моделі для нових зображень.
2. отримання відповіді: всі зображення були успішно оброблені, результати класифікації збережені, передбачення завершено коректно.

Рисунок В.2 Тест 2.

Виконавець: студент групи КІ-41, Ткаченко К.С. 

Додаток Г

Лістинг Г.1 – Імпорт бібліотек

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
import skfuzzy as fuzz
import skfuzzy.control as ctrl
import tensorflow as tf
import time


# Setting seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)
```

Лістинг Г.2 – Завантаження данних та обробка

```
train_dir = './seg_train'
test_dir = './seg_test'


# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.1)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
```

```
    subset='training'
)
val_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

Лістинг Г.3. – Традиційна нечітка система (приклад)

```
# Example fuzzy system for image brightness (simplified)
brightness = ctrl.Antecedent(np.arange(0, 256, 1), 'brightness')
contrast = ctrl.Antecedent(np.arange(0, 256, 1), 'contrast')
output_class = ctrl.Consequent(np.arange(0, 10, 1), 'output_class')

brightness['low'] = fuzz.trimf(brightness.universe, [0, 0, 128])
brightness['high'] = fuzz.trimf(brightness.universe, [128, 255, 255])

contrast['low'] = fuzz.trimf(contrast.universe, [0, 0, 128])
contrast['high'] = fuzz.trimf(contrast.universe, [128, 255, 255])

output_class['class1'] = fuzz.trimf(output_class.universe, [0, 0, 5])
output_class['class2'] = fuzz.trimf(output_class.universe, [5, 10, 10])
```

```
rule1 = ctrl.Rule(brightness['low'] & contrast['low'], output_class['class1'])
rule2 = ctrl.Rule(brightness['high'] & contrast['high'], output_class['class2'])

control_system = ctrl.ControlSystem([rule1, rule2])
fuzzy_system = ctrl.ControlSystemSimulation(control_system)

# Simulate for sample input
fuzzy_system.input['brightness'] = 90
fuzzy_system.input['contrast'] = 20
fuzzy_system.compute()

if 'output_class' in fuzzy_system.output:
    print(f"Predicted class: {fuzzy_system.output['output_class']}")
else:
    print("Output variable 'output_class' not generated. Check the rules or inputs.")
```

Лістинг Г.4 – Тестування на основі традиційної нечіткої системи

```
import os
import numpy as np
import skfuzzy as fuzz
import skfuzzy.control as ctrl
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from PIL import Image
import pandas as pd

# Define classes
classes = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']

# Define fuzzy variables
brightness = ctrl.Antecedent(np.arange(0, 256, 1), 'brightness')
contrast = ctrl.Antecedent(np.arange(0, 256, 1), 'contrast')
output_class = ctrl.Consequent(np.arange(0, len(classes), 1), 'output_class')
```

```
# Membership functions for brightness
brightness['low'] = fuzz.trimf(brightness.universe, [0, 0, 128])
brightness['medium'] = fuzz.trimf(brightness.universe, [64, 128, 192])
brightness['high'] = fuzz.trimf(brightness.universe, [128, 255, 255])

# Membership functions for contrast
contrast['low'] = fuzz.trimf(contrast.universe, [0, 0, 128])
contrast['medium'] = fuzz.trimf(contrast.universe, [64, 128, 192])
contrast['high'] = fuzz.trimf(contrast.universe, [128, 255, 255])

# Membership functions for output class
for i, class_name in enumerate(classes):
    output_class[class_name] = fuzz.trimf(output_class.universe, [i, i, i + 1])

# Define fuzzy rules
rules = [
    ctrl.Rule(brightness['low'] & contrast['low'], output_class['buildings']),
    ctrl.Rule(brightness['medium'] & contrast['medium'], output_class['forest']),
    ctrl.Rule(brightness['high'] & contrast['low'], output_class['glacier']),
    ctrl.Rule(brightness['low'] & contrast['high'], output_class['mountain']),
    ctrl.Rule(brightness['medium'] & contrast['high'], output_class['sea']),
    ctrl.Rule(brightness['high'] & contrast['medium'], output_class['street']),
]

# Create control system
control_system = ctrl.ControlSystem(rules)
fuzzy_system = ctrl.ControlSystemSimulation(control_system)

# Function to preprocess images and extract brightness/contrast
def preprocess_image(image_path):
    image = Image.open(image_path).convert('L')  # Convert to grayscale
    image = image.resize((128, 128))
    pixel_values = np.array(image)
```

```
    brightness = np.mean(pixel_values)
    contrast = np.std(pixel_values)
    return brightness, contrast

# Predict function using fuzzy logic
def predict_class(image_path):
    brightness_val, contrast_val = preprocess_image(image_path)
    fuzzy_system.input['brightness'] = brightness_val
    fuzzy_system.input['contrast'] = contrast_val
    fuzzy_system.compute()
    return np.round(fuzzy_system.output['output_class'])

# Evaluate system on a dataset with subfolders
def evaluate_system(data_dir, true_classes):
    predictions = []
    y_true = []
    for class_idx, class_name in enumerate(true_classes):
        class_dir = os.path.join(data_dir, class_name)
        for image_file in os.listdir(class_dir):
            image_path = os.path.join(class_dir, image_file)
            if image_file.lower().endswith(('png', 'jpg', 'jpeg')):
                predicted_class = predict_class(image_path)
                predictions.append(predicted_class)
                y_true.append(class_idx)

    y_pred = np.array(predictions)
    y_true = np.array(y_true)

    # Metrics
    accuracy = np.mean(y_true == y_pred)
    print("Accuracy:", accuracy)

    f1 = classification_report(y_true, y_pred, target_names=true_classes)
    print("\nF1-Score:\n", f1)
```

```
    cm = confusion_matrix(y_true, y_pred)
    print("\nConfusion Matrix:\n", cm)

    # Plot confusion matrix
    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Confusion Matrix")
    plt.colorbar()
    tick_marks = np.arange(len(true_classes))
    plt.xticks(tick_marks, true_classes, rotation=45)
    plt.yticks(tick_marks, true_classes)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

    return accuracy, f1, cm

# Evaluate predictions for a flat directory
def evaluate_predictions(data_dir, fuzzy_system, classes):
    predictions = []
    file_names = []

    # Iterate over all images in the folder
    for image_file in os.listdir(data_dir):
        image_path = os.path.join(data_dir, image_file)
        if image_file.lower().endswith(('png', 'jpg', 'jpeg')):  # Ensure valid image files
            predicted_class = predict_class(image_path)
            predictions.append(predicted_class)
            file_names.append(image_file)

    # Convert predictions to class names
    class_predictions = [classes[int(pred)] for pred in predictions]
```

```
    # Create a DataFrame for results
    results_df = pd.DataFrame({'Image': file_names, 'Predicted Class': class_predictions})

    # Display the first few rows of the results
    print("\nPrediction Results:")
    print(results_df.head())

    # Save results to a CSV file
    results_csv_path = os.path.join(data_dir, 'predictions.csv')
    results_df.to_csv(results_csv_path, index=False)
    print(f"\nPrediction results saved to: {results_csv_path}")

    return results_df

# Training phase (implicit for fuzzy rules)
train_dir = './seg_train'

# Testing phase
test_dir = './seg_test'
print("Testing Results:")
evaluate_system(test_dir, classes)

# Prediction phase
pred_dir = './seg_pred'
print("Prediction Results:")
predictions_df = evaluate_predictions(pred_dir, fuzzy_system, classes)
```

Лістинг Г.5 – Згорткова нейронна мережа

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from sklearn.metrics import confusion_matrix, classification_report
```

```
import matplotlib.pyplot as plt
import seaborn as sns


class NeoFuzzyClassifier:
    def __init__(self, input_shape=(150, 150, 3), num_classes=6):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.history = None
        self.model = self.build_model()

    def build_model(self):
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=self.input_shape),
            tf.keras.layers.MaxPooling2D(),
            tf.keras.layers.Conv2D(64, 3, activation='relu'),
            tf.keras.layers.MaxPooling2D(),
            tf.keras.layers.Conv2D(64, 3, activation='relu'),
            tf.keras.layers.MaxPooling2D(),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(64, activation='relu'),
            tf.keras.layers.Dense(self.num_classes, activation='softmax')
        ])

        model.compile(
            optimizer='adam',
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=['accuracy']
        )
        return model

    def prepare_data(self, train_dir, test_dir, batch_size=32):
        train_datagen = ImageDataGenerator(
            rescale=1./255,
            shear_range=0.2,
```

```
        horizontal_flip=True,
        brightness_range=[0.8, 1.2],
        channel_shift_range=30.0,
    )

    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=self.input_shape[:2],
        batch_size=batch_size,
        class_mode='categorical',
        subset='training',
        shuffle=True
    )

    validation_generator = train_datagen.flow_from_directory(
        test_dir,
        target_size=self.input_shape[:2],
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation',
        shuffle=False
    )

    test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=self.input_shape[:2],
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    return train_generator, validation_generator, test_generator
```

```
def train(self, train_generator, validation_generator, epochs=10):
    print("\nПочинаємо навчання моделі...")
    self.history = self.model.fit(
        train_generator,
        validation_data=validation_generator,
        epochs=epochs,
        verbose=1
    )
    print("\nНавчання завершено!")

def evaluate(self, test_generator):
    print("\nОцінюємо модель на тестових даних...")
    return self.model.evaluate(test_generator, verbose=1)

def predict(self, image_dir):
    print(f"\nРобимо передбачення для зображень з {image_dir}...")
    pred_datagen = ImageDataGenerator(rescale=1./255)
    pred_generator = pred_datagen.flow_from_directory(
        image_dir,
        target_size=self.input_shape[:2],
        batch_size=1,
        class_mode=None,
        shuffle=False
    )
    predictions = self.model.predict(pred_generator, verbose=1)
    print("Передбачення завершено!")
    return predictions

def plot_training_history(self, save_path=None):
    print("\nБудуємо графіки точності та втрат...")
    acc = self.history.history['accuracy']
    val_acc = self.history.history['val_accuracy']
    loss = self.history.history['loss']
```

```
    val_loss = self.history.history['val_loss']
    epochs_range = range(len(acc))

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Точність навчання')
    plt.plot(epochs_range, val_acc, label='Точність валідації')
    plt.title('Графік точності')
    plt.xlabel('Епоха')
    plt.ylabel('Точність')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Втрати навчання')
    plt.plot(epochs_range, val_loss, label='Втрати валідації')
    plt.title('Графік втрат')
    plt.xlabel('Епоха')
    plt.ylabel('Втрати')
    plt.legend()

    plt.tight_layout()
    if save_path:
        plt.savefig(f"{save_path}/training_history.png")
        print(f"Графік збережено у {save_path}/training_history.png")
    plt.show()

def plot_confusion_matrix(self, y_true, y_pred, classes, save_path=None):
    print("\nБудуємо матрицю плутанини...")
    cm = confusion_matrix(y_true.argmax(axis=1), y_pred.argmax(axis=1))
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.title('Матриця плутанини')
```

```
    plt.ylabel('Справжній клас')
    plt.xlabel('Передбачений клас')
    if save_path:
        plt.savefig(f"{save_path}/confusion_matrix.png")
        print(f"Матрицю плутанини збережено у {save_path}/confusion_matrix.png")
    plt.show()

def predict_single_image(self, image_path):
    """Передбачення класу для одного зображення"""
    img = load_img(image_path, target_size=self.input_shape[:2])
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    return self.model.predict(img_array)

def predict_directory(self, image_dir):
    """Передбачення класів для всіх зображень у директорії"""
    print(f"\nРобимо передбачення для зображень з {image_dir}...")

    supported_formats = {'.jpg', '.jpeg', '.png', '.bmp'}
    predictions = []
    image_paths = []

    # Перевіряємо, чи існує директорія
    if not os.path.exists(image_dir):
        raise ValueError(f"Директорія {image_dir} не існує")

    # Отримуємо список всіх файлів
    for root, dirs, files in os.walk(image_dir):
        for file in files:
            if os.path.splitext(file)[1].lower() in supported_formats:
                image_path = os.path.join(root, file)
                image_paths.append(image_path)
```

```
        if not image_paths:
            raise ValueError(f"Не знайдено зображень у форматах {supported_formats} в {image_dir}")

        # Робимо передбачення для кожного зображення
        for image_path in image_paths:
            try:
                pred = self.predict_single_image(image_path)
                predictions.append(pred[0])
                print(f"Оброблено: {image_path}")
            except Exception as e:
                print(f"Помилка при обробці {image_path}: {str(e)}")
                continue

        predictions = np.array(predictions)
        print("Передбачення завершено!")
        return predictions, image_paths

def run_classification(train_dir, test_dir, pred_dir, results_dir='results'):
    # Створюємо папку для результатів
    os.makedirs(results_dir, exist_ok=True)

    print("Ініціалізація класифікатора...")
    classifier = NeoFuzzyClassifier()

    print("\nПідготовка даних...")
    train_generator, validation_generator, test_generator = classifier.prepare_data(train_dir, test_dir)

    # Навчання моделі
    classifier.train(train_generator, validation_generator)

    # Оцінка на тестовому наборі
    test_loss, test_accuracy = classifier.evaluate(test_generator)
    print(f'\nТочність на тестових даних: {test_accuracy:.4f}')
```

```
# Збереження результатів у файл
with open(f"{results_dir}/results.txt", 'w', encoding='utf-8') as f:
    f.write(f"Точність на тестових даних: {test_accuracy:.4f}\n")
    f.write(f"Втрати на тестових даних: {test_loss:.4f}\n")

# Побудова графіків навчання
classifier.plot_training_history(results_dir)

# Отримання передбачень для тестового набору
print("\nОтримуємо передбачення для тестового набору...")
test_predictions = classifier.model.predict(test_generator)

# Побудова матриці плутанини для тестового набору
class_names = list(test_generator.class_indices.keys())
classifier.plot_confusion_matrix(
    tf.keras.utils.to_categorical(test_generator.classes),
    test_predictions,
    class_names,
    results_dir
)

# Передбачення для папки pred
try:
    print("\nРобимо передбачення для нових зображень...")
    pred_predictions, image_paths = classifier.predict_directory(pred_dir)

    # Зберігаємо передбачення для нових зображень
    pred_classes = np.argmax(pred_predictions, axis=1)
    with open(f"{results_dir}/predictions.txt", 'w', encoding='utf-8') as f:
        f.write("Передбачення для нових зображень:\n")
        for i, (pred_class, image_path) in enumerate(zip(pred_classes, image_paths)):
            f.write(f"Зображення {os.path.basename(image_path)}: {class_names[pred_class]}\n")

except Exception as e:
```

```
        print(f"Помилка при обробці папки pred: {str(e)}")
        pred_predictions = None

    print(f"\nУсі результати збережено у папці {results_dir}")
    return classifier, test_accuracy, test_predictions, pred_predictions

# Приклад використання:
if __name__ == "__main__":
    train_dir = 'seg_train'
    test_dir = 'seg_test'
    pred_dir = 'seg_pred'  # тепер код буде шукати зображення безпосередньо в цій папці

    classifier, test_accuracy, test_predictions, pred_predictions = run_classification(
        train_dir, test_dir, pred_dir
    )
```

Лістинг Г.6 – Гібридна нео-фаззі мережа

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Layer
from tensorflow.keras import backend as K

class NeoFuzzyLayer(Layer):
    def __init__(self, num_rules=5, **kwargs):
        self.num_rules = num_rules
        super(NeoFuzzyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        # Параметри функцій приналежності
        self.centers = self.add_weight(name='centers',
                                       shape=(input_shape[-1], self.num_rules),
                                       initializer='uniform',
                                       trainable=True)
```

```
        self.widths = self.add_weight(name='widths',
                                      shape=(input_shape[-1], self.num_rules),
                                      initializer='ones',
                                      trainable=True)

        # Вагові коефіцієнти правил
        self.rules_weights = self.add_weight(name='rules_weights',
                                             shape=(input_shape[-1], self.num_rules),
                                             initializer='uniform',
                                             trainable=True)

        super(NeoFuzzyLayer, self).build(input_shape)

    def call(self, x):
        # Розрахунок функцій приналежності
        expanded_x = K.expand_dims(x, axis=2)
        diff = expanded_x - K.expand_dims(self.centers, axis=0)

        # Гаусівські функції приналежності
        membership = K.exp(-K.square(diff) / (2 * K.square(self.widths)))

        # Зважена сума по правилах
        weighted_sum = membership * self.rules_weights
        output = K.sum(weighted_sum, axis=2)

        return output

    def compute_output_shape(self, input_shape):
        return input_shape

class HybridNeoFuzzyCNN:
    def __init__(self, input_shape=(150, 150, 3), num_classes=6, num_rules=5):
        self.input_shape = input_shape
```

```
        self.num_classes = num_classes
        self.num_rules = num_rules
        self.model = self.build_model()

    def build_model(self):
        # CNN частина
        inputs = tf.keras.Input(shape=self.input_shape)

        # Згорткові шари
        x = tf.keras.layers.Conv2D(37, 3, activation='relu')(inputs)
        x = tf.keras.layers.MaxPooling2D()(x)
        x = tf.keras.layers.BatchNormalization()(x)

        x = tf.keras.layers.Conv2D(75, 3, activation='relu')(x)
        x = tf.keras.layers.MaxPooling2D()(x)
        x = tf.keras.layers.BatchNormalization()(x)

        x = tf.keras.layers.Conv2D(75, 3, activation='relu')(x)
        x = tf.keras.layers.MaxPooling2D()(x)
        x = tf.keras.layers.BatchNormalization()(x)

        x = tf.keras.layers.Conv2D(150, 3, activation='relu')(x)
        x = tf.keras.layers.MaxPooling2D()(x)
        x = tf.keras.layers.BatchNormalization()(x)

        # Перетворення в плоский вектор
        x = tf.keras.layers.GlobalAveragePooling2D()(x)

        # Нео-фаззі шар
        x = NeoFuzzyLayer(num_rules=self.num_rules)(x)

        # Додаткові щільні шари
        x = tf.keras.layers.Dense(150, activation='relu')(x)
        x = tf.keras.layers.Dropout(0.5)(x)
```

```
    # Вихідний шар
    outputs = tf.keras.layers.Dense(self.num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    # Компіляція моделі з покращеним оптимізатором
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

def train(self, train_generator, validation_generator, epochs=10, callbacks=None):
    if callbacks is None:
        callbacks = [
            tf.keras.callbacks.EarlyStopping(
                patience=5,
                restore_best_weights=True
            ),
            tf.keras.callbacks.ReduceLROnPlateau(
                factor=0.5,
                patience=3
            )
        ]

    return self.model.fit(
        train_generator,
        validation_data=validation_generator,
        epochs=epochs,
        callbacks=callbacks
```

```
    )


def create_data_generators(train_dir, test_dir, pred_dir, input_shape=(150, 150), batch_size=32):
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    channel_shift_range=30.0,
    )

  test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
  )

  train_generator = train_datagen.flow_from_directory(
    train_dir,
    batch_size=batch_size,
    class_mode='categorical',
    target_size=input_shape,
    shuffle=True,
  )

  validation_generator = train_datagen.flow_from_directory(
    test_dir,
    target_size=input_shape,
    batch_size=batch_size,
    class_mode='categorical',
    #subset='validation',
    shuffle=False
  )

  test_generator = test_datagen.flow_from_directory(
    pred_dir,
```

```
        target_size=input_shape,
        batch_size=batch_size,
        class_mode='categorical'
    )

    return train_generator, validation_generator, test_generator

# Приклад використання:
if __name__ == "__main__":
    # Параметри
    input_shape = (150, 150, 3)
    num_classes = 6
    num_rules = 5
    epochs = 20
    batch_size = 32

    # Шляхи до даних
    train_dir = 'seg_train'
    test_dir = 'seg_test'
    pred_dir = 'seg_pred'

    # Створення генераторів даних
    train_generator, validation_generator, test_generator = create_data_generators(
        train_dir, test_dir, pred_dir, input_shape[:2], batch_size
    )

    # Створення та навчання моделі
    model = HybridNeoFuzzyCNN(input_shape, num_classes, num_rules)
    history = model.train(train_generator, validation_generator, epochs=epochs)
```