няльний аналіз показав, що використання локального модуля на базі ResNet50 є економічно вигіднішим, порівняно з хмарними API (Google Cloud Vision, Microsoft Custom Vision) для задач, що не вимагають обробки надвеликих масивів даних у хмарі, водночас забезпечуючи зіставну точність розпізнавання поширених видів тварин.

Результати дослідження підтверджують доцільність використання архітектури ResNet50 із застосуванням трансферного навчання для задач ідентифікації тварин. Розроблений програмний засіб забезпечує високу точність (92 %) та швидкодію, достатню для використання у прикладних задачах. Подальший розвиток дослідження вбачається в адаптації моделі для обробки відеопотоків та інтеграції алгоритмів детекції об'єктів (YOLO) для задач підрахунку поголів'я.

Список використаних джерел

1. Research on animal image recognition based on improved CNN model / J. Wang, Y. Sun, Y. Bao, J. Zhao. *Journal of Intelligent & Fuzzy Systems.* 2020. Vol. 38, № 3. P. 2881–2892.

2. Deep residual learning for image recognition / K. He, X. Zhang, S. Ren, J. Sun. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016. P. 770–778. URL: https://arxiv.org/abs/1512.03385 (дата звернення: 30.10.2025).

3. Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring / H. Nguyen, S. Maclagan, T. Nguyen. *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA).* 2017.

УДК 004.422.5

Заболотна У. К., здобувачка 2 курсу спеціальності 122 Комп'ютерні науки,
Потапова Н. А., канд. екон. наук, доцент, доцент кафедри інформаційних технологій

ГЕНЕТИЧНІ АЛГОРИТМИ ТА ЇХ ЗАСТОСУВАННЯ

Донецький національний університет імені Василя Стуса, м. Вінниця

Генетичні алгоритми (GA) – це адаптивні евристичні алгоритми пошуку, що належать до ширшого класу еволюційних алгоритмів. Вони базуються на принципах природного добору та генетики, імітуючи процеси еволюції живих організмів. Це розумна експлуатація випадкових пошуків, наданих історичними даними для спрямування пошуку в область кращої продуктивності в просторі рішень.

Зазвичай генетичні алгоритми застосовують для розв'язання задач оптимізації та пошуку. Ці алгоритми кодують потенційне рішення певної задачі у вигляді простої структури, що нагадує хромосому, та застосовують оператори рекомбі-

нації, щоб зберегти критичну інформацію. Генетичні алгоритми часто розглядають як оптимізатори функцій, хоча спектр задач, де вони застосовуються, значно ширший.

Реалізація генетичного алгоритму починається зі створення популяції – зазвичай випадкової – хромосом. Далі кожен індивід оцінюється за певною метрикою якості, яку ще називають функцією пристосованості (fitness). Рішення, які демонструють кращу якість, отримують більше шансів «передати свої гени» наступним поколінням. Отже, добір відбувається відносно всієї поточної популяції.

Генетичні алгоритми імітують процес природного відбору, що означає, що ті види, які можуть адаптуватися до змін у своєму середовищі, можуть виживати, розмножуватися і переходити до наступного покоління. Інакше кажучи, вони імітують «виживання найбільш пристосованих» серед людей послідовних поколінь для вирішення проблеми. Кожне покоління складається з населення індивіда, і кожна людина являє собою точку в пошуковому просторі та можливе рішення. Кожна людина представлена у вигляді рядка символа / ціле число / плавання / біт.

Населення людей зберігається в пошуковому просторі. Кожна людина являє собою рішення в пошуковому просторі для даної проблеми. Кожна людина кодується як кінцевий вектор довжини (аналогічний до хромосоми) компонентів. Ці змінні компоненти аналогічні генам. Отже, хромосома (індивідуальна) складається з декількох генів (змінних компонентів).

Після створення початкового покоління алгоритм розвиває генерацію, використовуючи такі оператори:

– ***Оператор відбору:*** Функція відбору визначає, які особини з популяції будуть використовуватися для створення нового покоління. Її головна ідея – віддати перевагу особам із вищою пристосованістю, щоб саме вони передавали свої гени далі. Це імітує природний добір: найкраще пристосовані особини мають більші шанси вижити й розмножитися.

– Кожен метод має свої переваги і використовується залежно від характеру задачі та бажаного рівня селективного тиску.

– ***Оператор кросовера (схрещування):*** Кросовер створює нових особин (потомство), поєднуючи генетичну інформацію двох батьків. Мета – успадкувати корисні риси обох та покращувати різноманітність популяції.

– ***Оператор мутації:*** Мутація додає до хромосом потомства випадкові зміни, підтримуючи генетичну різноманітність популяції. Це запобігає передчасній конвергенції – ситуації, коли всі особини стають надто схожими і алгоритм перестає знаходити кращі рішення.

Мутація зазвичай застосовується з малою ймовірністю, але її роль є критичною для дослідження нових ділянок простору рішень.

Розглянемо приклад використання генетичного алгоритму (рис. 2).

У цій задачі генетичний алгоритм шукає число, найближче до 100. Популяція – це масив випадкових чисел, функція пристосованості оцінює близькість до 100, кросовер бере два найкращі числа і створює середнє потомство, а мутація додає випадкове число для різноманітності. Алгоритм повторюється 30 поколінь, поступово наближаючись до цільового числа. Користь GA в тому, що він демонструє пошук оптимальних рішень через відбір, кросовер і мутацію, ефективний для задач, де простий перебір не підходить, імітуючи еволюційні процеси.

```
using System;
using System.Linq;

0 references
class Program
{
    static Random r = new Random();
    1 reference
    static int Fitness(int x) => Math.Abs(100 - x);
    1 reference
    static int Mutate(int x) => x + r.Next(-5, 6);
    1 reference
    static int Cross(int a, int b) => (a + b) / 2;

    0 references
    static void Main()
    {
        int[] pop = Enumerable.Range(0, 10).Select(_ => r.Next(0, 200)).ToArray();

        for (int gen = 0; gen < 30; gen++)
        {
            pop = pop.OrderBy(x => Fitness(x)).ToArray();
            Console.WriteLine($"Gen {gen}: best = {pop[0]}");

            int p1 = pop[0], p2 = pop[1];
            pop = pop.Select(_ => Mutate(Cross(p1, p2))).ToArray();
        }
    }
}
```

Рисунок 1 – Приклад використання генетичного алгоритму

Отже, генетичні алгоритми – це еволюційний метод пошуку, який імітує природний добір, щоб поступово покращувати розв'язки. Вони ефективні для складних задач, де важко знайти оптимальний варіант звичайними методами. Завдяки поєднанню випадковості та відбору генетичні алгоритми здатні швидко знаходити якісні наближення оптимуму й широко застосовуються в інженерії, науці та штучному інтелекті.

Список використаних джерел

1. Генетичні алгоритми. GeeksforGeeks. URL: https://www.geeksforgeeks.org/dsa/genetic-algorithms/ (дата звернення: 25.11.2025).

2. Алгоритм: повний посібник з реалізації Python. DataCamp. URL: https://www.datacamp.com/tutorial/genetic-algorithm-python (дата звернення: 25.11.2025).

3. Огляд застосування генетичних алгоритмів до прогнозування цін на товари. URL: https://www.mdpi.com/2227-7099/9/1/6 (дата звернення: 25.11.2025).

4. A Genetic Algorithm Tutorial. URL: https://www.cs.jhu.edu/~ayuille/courses/Stat202CSpring10/ga_tutorial.pdf (дата звернення: 25.11.2025).