НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Використання генетичних алгоритмів в задачі машинного налаштування згорткових нейронних мереж**»**

Виконав:
студент IV курсу, групи КІ-03
Панасюк Р.І. _____

Керівник:
асистент Кот А.Т. _____

Консультант з економічного розділу:
доцент Мажара Г.А. _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець П.В. _____

Рецензент:
к.т.н., доцент кафедри АКІК ФАЕТ НАУ,
Василенко М.П. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Панасюку Руслану Ігоровичу

1. Тема роботи «Використання генетичних алгоритмів в задачі машинного налаштування згорткових нейронних мереж», керівник роботи асистент Кот А.Т., затверджені наказом по університету від «31» травня 2024 р. № 2240-С

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані до роботи: зображенння магнітно-резонансної томографії головного мозку пацієнтів з можливою наявністю пухлин

4. Зміст роботи: дослідження застосування генетичних алгоритмів для автоматичного налаштування гіперпараметрів згорткових нейронних мереж з метою підвищення ефективності їхнього навчання та продуктивності

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): схеми алгоритмів, графіки та блок-схеми, приклади вхідних даних

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Мажара Гліб Анатолійович, доцент		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	15.04.2024	Виконано
2	Підготовка першого розділу	24.04.2024	Виконано
3	Підготовка другого розділу	05.05.2024	Виконано
4	Розробка програмного продукту	9.05.2024	Виконано
5	Підготовка третього розділу	20.05.2024	Виконано
6	Підготовка економічної частини	23.05.2024	Виконано
7	Оформлення розділів відповідно до нормоконтролю	25.05.2024	Виконано
8	Оформлення дипломної роботи	28.05.2024	Виконано
9	Підготовка презентації доповіді	30.05.2024	Виконано

Студент Руслан Панасюк

Керівник Анатолій Кот

РЕФЕРАТ

Дипломна робота: 87 с., 26 рис., 6 табл., 23 посилань, 1 додаток.

ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ В ЗАДАЧІ МАШИННОГО НАЛАШТУВАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

У цьому документі розглянута проблема машинного налаштування згорткових нейронних мереж (ЗНМ) за допомогою генетичних алгоритмів (ГА). Робота присвячена розробці ефективних методів оптимізації параметрів ЗНМ, що можуть бути застосовані в різних сферах: класифікація зображень, розпізнавання облич, обробка медичних зображень, автоматичне керування транспортом, виявлення аномалій та багатьох інших.

Об’єкт дослідження – згорткові нейронні мережі.

Предмет дослідження – методи машинного налаштування параметрів ЗНМ з використанням генетичних алгоритмів.

Мета роботи – розробка автоматизованої системи для оптимізації параметрів згорткових нейронних мереж з використанням генетичних алгоритмів з метою поліпшення їх продуктивності у задачі класифікації.

Результат – в роботі розглянуто основні класи нейронних мереж та їх алгоритмів оптимізації, детально розглянуто ЗНМ та ГА, розроблено програмний продукт для автоматизованої системи вибору та оптимізації параметрів ЗНМ та визначено його економічну собівартість.

ABSTRACT

Master's thesis: 87 p., 26 figures, 6 tables, 23 references, 1 appendix.

USING GENETIC ALGORITHMS IN THE PROBLEM OF MACHINE TUNING OF CONVOLUTIONAL NEURAL NETWORKS

This document addresses the problem of tuning convolutional neural networks (CNNs) using genetic algorithms (GAs). The work is dedicated to the development of effective methods for optimizing the parameters of CNNs, which can be applied in various fields: image classification, face recognition, medical image processing, autonomous vehicle control, anomaly detection, and many others.

The object of the study is convolutional neural networks

The subject of research is methods of tuning CNN parameters using genetic algorithms.

The purpose of the work is to develop an automated system for optimizing the parameters of convolutional neural networks using genetic algorithms to improve their performance in classification and recognition tasks.

The result of the work includes an examination of the main classes of neural networks and their optimization algorithms, a detailed review of CNNs and GAs, the development of a software product for an automated system for selecting and optimizing CNN parameters, and the determination of its economic cost.

ЗМІСТ

ВСТУП

Розвиток та оптимізація нейронних мереж – це процес, що вимагає постійного пошуку ефективних методів налаштування параметрів моделей. Одним із найбільш перспективних підходів є використання генетичних алгоритмів, які відтворюють природний процес еволюції для пошуку оптимальних конфігурацій нейронних мереж. З миттєвим зростанням обсягів даних та складності завдань у сфері штучного інтелекту, машинне навчання стає важливим інструментом для вирішення різноманітних проблем. В даному дослідженні досліджується використання генетичних алгоритмів у задачі машинного налаштування згорткових нейронних мереж, що є актуальним напрямом у сфері комп'ютерного зору та обробки зображень. Глибокі згорткові нейронні мережі здатні до складних розпізнавань та класифікацій, але їх ефективність значно залежить від правильного налаштування ваг і гіперпараметрів. Використання генетичних алгоритмів дає змогу автоматизувати цей процес та знаходити оптимальні конфігурації мережі, підвищуючи її точність та швидкість роботи. Такий підхід має потенціал зробити нейронні мережі більш адаптивними та ефективними у вирішенні реальних завдань обробки інформації.

РОЗДІЛ 1 ШТУЧНИЙ ІНТЕЛЕКТ ТА ЙОГО ОСОБЛИВОСТІ

1.1 Перспектива розвитку штучного інтелекту

Штучний інтелект все більше впливає на щоденне життя кожної людини. Останні роки принесли значний розвиток у галузі алгоритмів машинного та глибокого навчання. Цей прогрес дає змогу штучному інтелекту та його компонентам, таким як нейронні мережі, вирішувати складні завдання, такі як розпізнавання образів та мовлення, генерація зображень, тексту, звуків і багато іншого. Штучний інтелект оптимізує процеси, підвищує ефективність та мінімізує помилки, полегшуючи дослідження та вирішення складних завдань [1].

Проте, неправильне використання штучного інтелекту може мати негативний вплив. Помилки у програмах можуть викликати непередбачені наслідки, спричиняючи проблеми у функціонуванні систем. Використання штучного інтелекту може призвести до проблем безробіття через автоматизацію робочих процесів, втрату креативності при написанні програм, залежності людей від технологій. Окрім того, розвиток ШІ сповільнюється тим фактом, що для розвитку технологій необхідна витрата великої кількості грошей та часу.

Незважаючи на це, ШІ розвивається семимильними кроками. Оцінюється, що конкретний внесок штучного інтелекту у світовий ВВП буде на рівні 15,700 млрд. доларів США між 2018 і 2030 роками, що становить зростання на 14%. Створення вартості повинно бути вищим у регіоні Азіатсько-Тихоокеанського регіону (26%) та Північної Америки (14.5%), ніж у Європі (від 9.9% до 11.5%) та країн що розвиваються. Це головним чином призведе до збільшення продуктивності (55%) та відновлення споживання (45%) до 2030 року. Кількість роботів у китайській виробничій галузі досягла співвідношення 322 одиниці на 10,000 працівників у 2021 році, за якими йде американська промисловість з 274 одиницями на 10,000 працівників [2].

1.2 Класифікація штучних нейронних мереж

Нейронна мережа - це модель обчислювання, яка намагається моделювати роботу людського мозку, застосовуючи штучні нейрони (або просто "нейрони") для обробки і передачі інформації. Вона складається з набору взаємопов'язаних один з одним штучних нейронів, які організовані у шари. Кожен нейрон отримує вхідні сигнали, обробляє їх, і видає вихідний сигнал. Шляхом налаштування ваг, які визначають силу зв'язків між нейронами, нейронні мережі можуть "навчатися" розв'язувати різноманітні завдання, такі як розпізнавання зображень або мови, класифікація тексту, прогнозування тенденцій та багато інших [3].

Розрізняють такі нейронні мережі.

Нейронна мережа прямого поширення (FNN[1]) – це тип штучної нейронної мережі, де інформація поширюється лише у напрямку від вхідного шару до вихідного шару. Вхідні дані *x* проходять через проміжні обчислення, які використовуються для визначення функції *f*, і, нарешті, результат обчислень виходить на вихід *y*.

Рекурентна нейронна мережа (RNN) — це тип штучної нейронної мережі, призначений для обробки послідовних даних за допомогою збереження пам'яті про попередні входи.

Згорткові нейронні мережі (CNN) — це спеціалізований тип нейронних мереж для обробки даних, які мають відому сіткову топологію.

Глибокі нейронні мережі (DNN) - це тип штучної нейронної мережі з кількома прихованими шарами між вхідним та вихідним шарами. Ці приховані шари дають змогу мережі вивчати складні представлення вхідних даних шляхом ієрархічного вилучення функцій на різних рівнях абстракції. Глибина мережі вказує на кількість прихованих шарів, які вона містить.

[1] тут і далі в дужках терміни англійською

Варто зазначити, що згадані вище типи нейронних мереж є лише кількома найпопулярнішими представниками широкого спектру можливостей нейронних мереж.

1.3 Оптимізація нейронних мереж

Оптимізація нейронних мереж є ключовим етапом у навчанні та покращенні їх продуктивності. Існує багато алгоритмів оптимізації, серед яких можна виділити основні групи: ройові алгоритми оптимізації, еволюційні алгоритми, градієнтні методи та баєсові методи.

Ройові алгоритми оптимізації базуються на поведінці групи агентів або частинок, що співпрацюють для знаходження оптимального розв'язку. Найбільш відомий приклад — це алгоритм рою частинок (Particle Swarm Optimization, PSO). В основі цього методу лежить імітація соціальної поведінки птахів чи риб, де кожен агент оновлює свою позицію на основі власного досвіду та досвіду сусідів. Такі алгоритми ефективні при розв'язанні складних багатовимірних задач, де градієнтні методи можуть бути менш ефективними [4].

Еволюційні алгоритми оптимізації надихаються принципами природного відбору та генетики. Серед них відомі генетичні алгоритми, еволюційні стратегії та генетичне програмування. Ці методи використовують процеси мутації, схрещування та відбору для створення нових поколінь рішень, поступово покращуючи їхню якість. Вони особливо корисні для задач з великим числом локальних мінімумів або для оптимізації функцій з невідомою структурою [5].

Градієнтні методи є найбільш популярними в контексті навчання нейронних мереж. До них належать методи стохастичного градієнтного спуску (SGD), адаптивний градієнтний алгоритм (Adagrad), RMSprop та Adam. Ці

методи базуються на обчисленні градієнта функції втрат та оновленні параметрів моделі в напрямку зменшення цієї функції. Завдяки своїй ефективності та простоті реалізації, градієнтні методи широко використовуються в глибокому навчанні [6].

Баєсові методи оптимізації використовують ймовірнісні моделі для пошуку оптимальних гіперпараметрів. Одним з основних прикладів є баєсова оптимізація, яка будує ймовірнісну модель функції втрат та використовує цю модель для вибору наступних точок оцінки. Цей підхід дає змогу зменшити кількість обчислень, необхідних для знаходження оптимальних параметрів, що робить його корисним для задач з високою обчислювальною вартістю [7].

Крім основних груп, можна виділити інші методи оптимізації, які також використовуються в навчанні нейронних мереж. Серед них такі методи, як диференційна еволюція, імітація відпалу та градієнтні методи з імпульсом. Кожен з цих методів має свої переваги та недоліки, що робить їх корисними в різних сценаріях оптимізації.

1.4 Постановка задачі дослідження

Дослідження спрямоване на розробку автоматизованої системи, яка б оптимізувати гіперпараметри ЗНМ з використанням ГА, сприяючи підвищенню їхньої ефективності в різноманітних задачах, зокрема, у завданні класифікації.

Висновки по розділу 1

Штучний інтелект суттєво впливає на щоденне життя людей, сприяючи вирішенню різних завдань. Останні роки приносять значний прогрес у сфері алгоритмів машинного та глибокого навчання, що дає змогу штучному інтелекту та його компонентам, таким як нейронні мережі, розв'язувати складні завдання, включаючи розпізнавання образів, мовлення, генерацію зображень та тексту. Цей прогрес оптимізує процеси, підвищує ефективність та мінімізує помилки, сприяючи вирішенню складних завдань.

Проте, неправильне використання штучного інтелекту може мати негативні наслідки. Помилки у програмах можуть призвести до непередбачених проблем у функціонуванні систем, а автоматизація робочих процесів може призвести до безробіття та втрати креативності у написанні програм.

Щодо класифікації штучних нейронних мереж, важливо відзначити, що ці моделі використовуються для моделювання роботи людського мозку. Існують різні типи нейронних мереж, такі як мережі прямого поширення, рекурентні нейронні мережі, згорткові нейронні мережі та глибокі нейронні мережі. Кожен тип має свої унікальні особливості та застосування у різних галузях.

Розглядаючи основні групи алгоритмів оптимізації, можна визначити, що кожен з них має свої переваги та обмеження, які варто враховувати при виборі методу для конкретного завдання. Ройові алгоритми оптимізації дають змогу знаходити оптимальні рішення у складних багатовимірних просторах, еволюційні методи надають можливість знаходити рішення в задачах з великою кількістю локальних мінімумів, градієнтні методи відомі своєю ефективністю та широким застосуванням в глибокому навчанні, а баєсові методи дають змогу ефективно використовувати обмежені обчислювальні ресурси для знаходження оптимальних параметрів.

РОЗДІЛ 2 ОПТИМІЗАЦІЯ ЗНМ ЗА ДОПОМГОЮ ГА

2.1 Основи генетичних алгоритмів

Генетичні алгоритми – це пошукові алгоритми засновані на механізмах природного відбору та генетики. Генетичні алгоритми використовують терміни, запозичені з генетики, для пояснення своєї роботи. Також застосовуються відповідні терміни з технічного лексикону, такі як ланцюг, двійкова послідовність, структура.

Популяція - це скінченна група особин. У генетичних алгоритмах особини представлені хромосомами, які містять параметри задачі, також відомі як точки в просторі пошуку. У деяких випадках особини називаються організмами.

Хромосоми (або ланцюжки) - це впорядковані послідовності генів. Ген (також відомий як властивість чи ознака) - це основний елемент генотипу, який складає хромосому.

Генотип або структура - це набір хромосом даної особини. Отже, особинами популяції можуть бути як генотипи, так і окремі хромосоми.

Фенотип - це набір значень, що відповідає даному генотипу, або розв'язок задачі.

Алель - це конкретне значення гена або властивості.

Локус або позиція показує місце розташування гена на хромосомі. Сукупність позицій генів називається локусами [8].

Дуже важливим поняттям у генетичних алгоритмах є функція пристосованості (або функція оцінки), яка визначає ступінь пристосованості особини в популяції. Ця функція визначає, наскільки добре особина пристосована до середовища, і дозволяє відібрати найбільш пристосовані особини для наступного покоління.

Генетичний алгоритм є методом пошуку, що ґрунтується на ідеї виживання найбільш пристосованих [9]. Він використовує популяцію і

використовує генетичні операції для створення нових поколінь. Процес ГА може бути узагальнений наступним чином: спочатку створюється початкова популяція Y хромосом, яка ініціалізується випадковим чином. Далі обчислюється пристосованість кожної хромосоми у популяції Y. Потім вибираються дві хромосоми, наприклад, $C1$ і $C2$, на основі їх пристосованості. Застосовується одноточковий оператор кросинговеру з ймовірністю C_p до $C1$ і $C2$, щоб створити нащадка, O. Після цього застосовується оператор рівномірної мутації з ймовірністю мутації M_p до нащадка O, створюючи O'. Нове покоління O' додається до нової популяції. Ці операції повторюються для кожної хромосоми поточної популяції, поки не буде сформована нова популяція [10].

В ГА застосовуються різноманітні оператори під час процесу пошуку, це схеми кодування, кросинговер, мутація та відбір. Оператори, які використовуються в ГА, зображено на рис. 2.1

2.1.1 Схеми кодування

Для більшості обчислювальних задач важливу роль відіграє схема кодування (тобто перетворення в певну форму). Надана інформація має бути закодована в певному бітовому рядку [11].

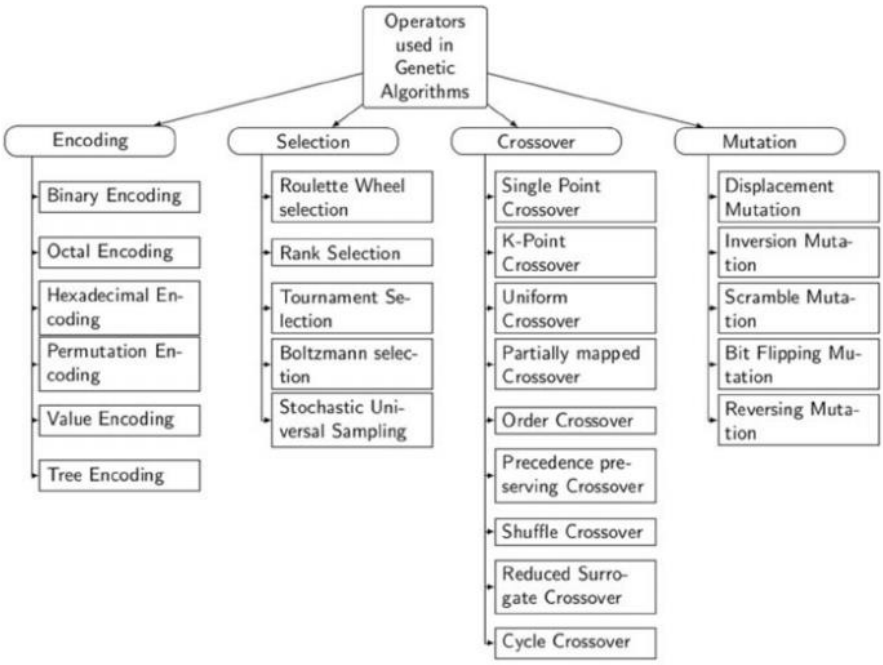

Рисунок 2.1 – Оператори які застосовуються в ГА [10]

Двійкове кодування є загальновживаною схемою кодування. Кожен ген або хромосома представлена у вигляді рядка з 1 або 0 [12]. У двійковому кодуванні кожен біт представляє характеристики рішення. Він забезпечує швидшу реалізацію операторів кросинговеру та мутації. Однак для перетворення у двійкову форму потрібні додаткові зусилля, а точність алгоритму залежить від двійкового перетворення.

У вісімковій схемі кодування ген або хромосома представлена у вигляді вісімкових чисел (0–7). У шістнадцятковій схемі кодування ген або хромосома представлені у вигляді шістнадцяткових чисел (0-9, A-F)

Схема перестановочного кодування зазвичай використовується в задачах упорядкування. У цій схемі кодування ген або хромосома представлена рядком чисел, що представляє позицію в послідовності.

У схемі кодування значень ген або хромосома представлена за допомогою рядка деяких значень. Ці значення можуть бути дійсними, цілими числами або символами [13].

У дереві кодування ген або хромосома представлена деревом функцій або команд.

2.1.2 Техніки відбору

Відбір — це важливий крок у генетичних алгоритмах, який визначає, чи братиме певний рядок участь у процесі відтворення чи ні.

Відбір колеса рулетки відображає всі можливі ланцюжки на колесі з частиною колеса, яка призначається їм відповідно до їх значення придатності. Потім це колесо обертається випадковим чином, щоб вибрати конкретні рішення, які братимуть участь у формуванні наступного покоління.

Вибір за рангом є модифікованою формою відбору колесом рулетки. Він використовує ранги замість значення пристосованості. Ранги присвоюються відповідно до рівня пристосованості, щоб кожна особа отримала шанс бути обраною відповідно до своїх рангів.

Техніка відбору турніром полягає в тому, щоб випадковим чином вибрати підмножину хромосом (особин) з поточної популяції і порівняти їх між собою, обираючи тих, хто має краще значення пристосованості.

Стохастична універсальна вибірка є розширенням існуючого методу відбору колеса рулетки. Він використовує випадкову початкову точку в списку індивідів із покоління та вибирає нову особину через рівні інтервали

Больцманівська селекція - це метод вибору особин для наступного покоління який використовує температурний параметр, щоб керувати рівнем випадковості та допомагає уникнути застрягання в локальних оптимумах. Основна ідея полягає в тому, щоб розглядати значення пристосованості кожної особини як енергію, а температурний параметр як показник випадковості [14].

2.1.3 Оператори кросинговеру

Оператори кросинговеру використовуються для створення нащадків шляхом поєднання генетичної інформації двох або більше батьків.

В одноточковому кросинговері вибирається випадкова точка кросинговера. Генетична інформація двох батьків, яка знаходиться за межами цієї точки, буде обміняна один з одним. На рис. 2.2 зображено процес зміни інформації під час одноточкового кросинговеру.

У двоточковому та k-точковому схрещуванні вибираються дві або більше випадкових точок схрещування, і генетична інформація батьків буде замінена відповідно до сегментів, які були створені [15].

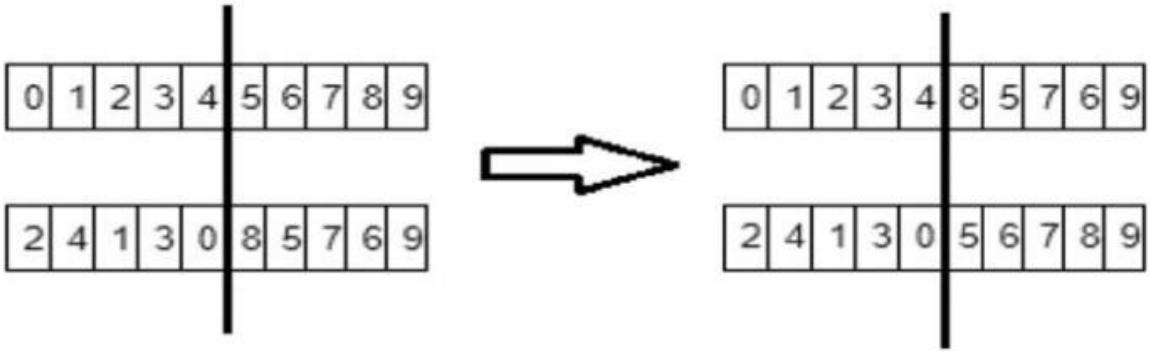

Рисунок 2.2 – Заміна генетичної інформації одноточкового кросинговеру [10]

Рівномірний кросинговер - це метод комбінації генетичного матеріалу двох батьківських, де кожен біт нащадка має рівну ймовірність бути успадкованим від одного з батьків.

При частково узгодженому кросинговері випадковим чином вибираються дві точки розриву на батьківських хромосомах, генетичний матеріал між цими двома точками копіюється без змін до нащадка. Далі, для кожного гена, який залишився між точками розриву на першому батьківському хромосомі,

знаходиться його відповідник на другому батьківському хромосомі. Потім замінюються ці гени в нащадка, але з урахуванням того, що їх нове розташування не суперечить уже існуючим генам у нащадка. Останній крок - це виправлення конфліктів, які можуть виникнути через дублювання генів. Для цього перевіряється, чи є дубльовані гени в нащадка, і, якщо так, замінюються на їх відповідники з другого батьківського хромосоми, що не включені до обраного фрагмента [16]. Процес узгодженого кросинговеру зображено на рис. 2.3.

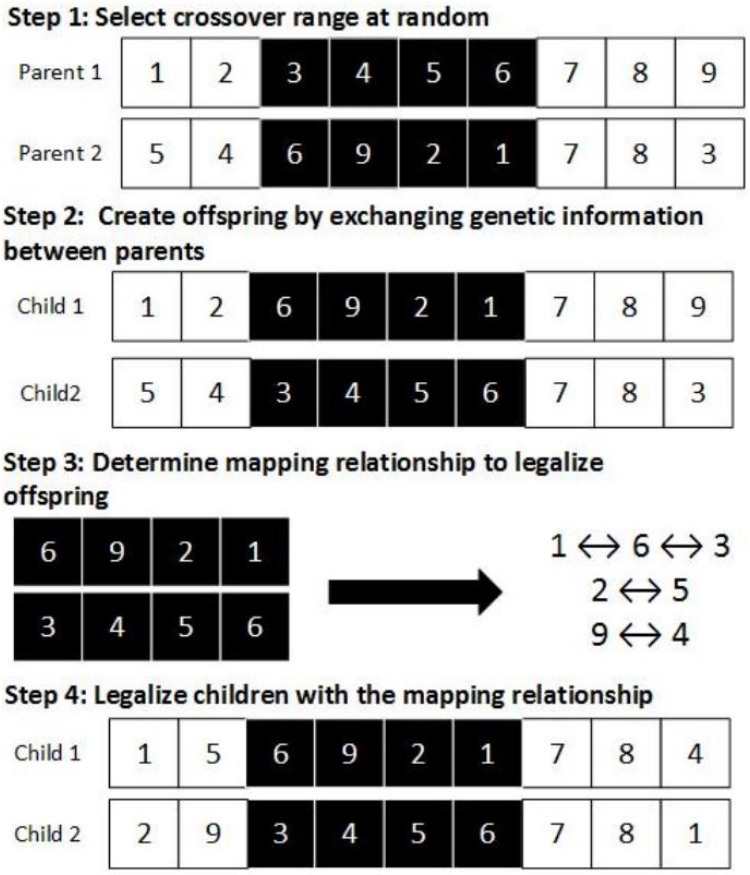

Рисунок 2.3 – Процес узгодженого кросинговеру [16]

Порядковий кросинговер копіює одну частину (або більше) частин батьківського елемента в потомство з вибраних точок розрізу та заповнює простір, що залишився, значеннями, відмінними від тих, що включені в скопійований розділ.

Кросинговер перемішуванням (shuffle crosover) спочатку випадковим чином перемішує гени обох батьків, але однаково. Потім він застосовує техніку одноточкового схрещування, випадковим чином вибираючи точку як точку схрещування, а потім поєднує обох батьків, щоб створити двох нащадків. Після виконання одноточкового кросинговеру гени в нащадках потім впорядковуються так само, як вони були перетасовані [17].

2.1.4 Оператори мутації

Мутація - це механізм, який забезпечує збереження генетичної різноманітності між поколіннями від однієї популяції до іншої. Добре відомими операторами мутації є зміщення, проста інверсія та перестановки.

Оператор мутації зміщення (displacement mutation) розпочинається з випадкового вибору підрядка генетичного коду. Цей підрядок видаляється зі свого початкового місця та вставляється у випадкову позицію всередині коду [18]. Процес мутації зміщення зображено на рис. 2.4.

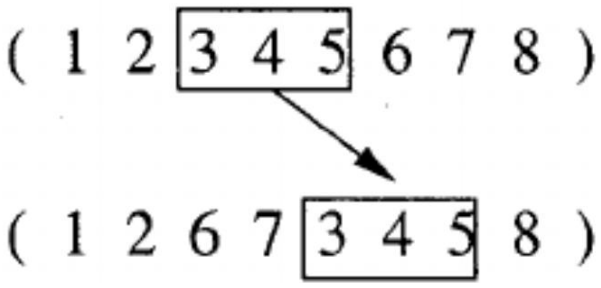

Рисунок 2.4 – Мутація зміщенням [19]

Простий оператор мутації інверсії (simple inversion mutation) - це генетичний алгоритм, де випадковим чином вибирається дві точки у генетичному коді (послідовності), і послідовність між цими точками

перевертається. Наприклад, якщо для послідовності [1, 2, 3, 4, 5, 6] випадково обираєтья точки між 2 та 5, то нова послідовність буде мати вигляд [1, 5, 4, 3, 2, 6].

Оператор мутації "перестановка" (SM) розміщує елементи в заданому діапазоні індивідуального розв'язку у випадковому порядку та перевіряє, чи покращилося значення функції пристосованості новоствореного розв'язку.

Рівномірна мутація передбачає додавання випадкового значення, взятого з рівномірного розподілу в певному діапазоні.

2.2 Згорткові нейронні мережі

ЗНМ є різновидом нейронної мережі прямого поширення, яка здатна отримувати характеристики з даних за допомогою структурної згортки [19]. Архітектура CNN надихнута візуальним сприйняттям. Біологічний нейрон відповідає штучному нейрону; фільтьтри (ядра) CNN представляють різні рецептори, які можуть реагувати на різні ознаки; функції активації моделюють функцію, за якою сигнали нейронів можуть передаватися до наступного нейрону лише у разі перевищення певного порога.

2.2.1 Принцип роботи ЗНМ

Для створення моделі ЗНМ зазвичай потрібні чотири компоненти. Згортка є ключовим кроком для виділення ознак. Виходи згортки можна назвати картою ознак. При встановленні згорткового фільтра з певним розміром втрачається інформацію на межі. Тому було введено доповнення, щоб збільшити вхід з нульовими значеннями, що може регулювати розмір

опосередковано. Крім того, для контролю щільності згортки використовується крок. Чим більший крок, тим менша щільність. Після згортки карти ознак складаються з великої кількості ознак, що схильні до проблеми перенавчання. У результаті було запропоновано пулінг (також відомий як зменшення), щоб уникнути зайвості, включаючи максимальне згладжування та середнє згладжування. Процедура ЗНМ зображено на рис. 2.5.

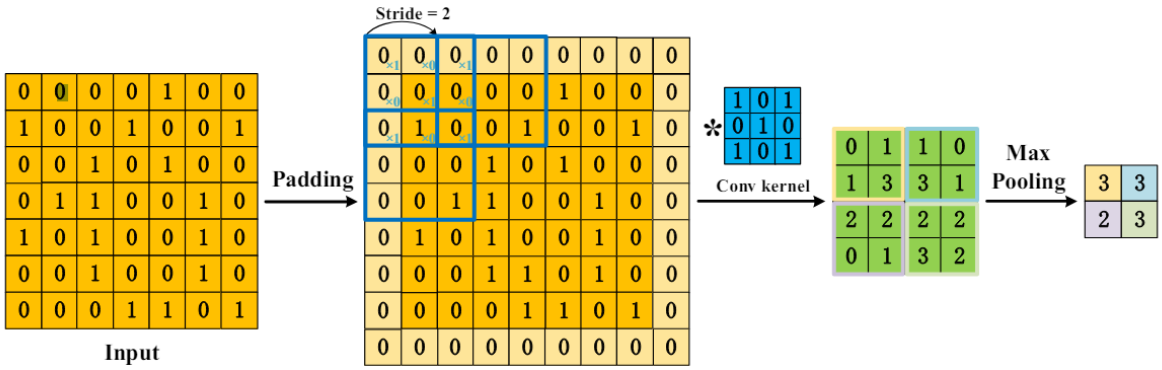

Рисунок 2.5 – Процедура двомірного ЗНМ [19]

2.2.3 Гіперпараметри ЗНМ

Гіперпараметри ЗНМ — це параметри, які визначають структуру мережі та процес її навчання, і вони не оновлюються під час навчання моделі. Натомість, ці параметри встановлюються перед початком навчання і значно впливають на якість та ефективність роботи мережі [20].

Кількість шарів визначає складність моделі та її здатність навчатися зображенням на різних рівнях абстракції. Збільшення кількості шарів дає змогу моделі краще розпізнавати складніші патерни, але може призвести до переобучення та збільшення обчислювальних витрат.

Кількість фільтрів впливає на обсяг інформації, яку може захопити кожен шар. Більша кількість фільтрів дає змогу виявляти більше деталей на кожному рівні, але збільшує обчислювальні витрати та потребу в пам’яті.

Розмір фільтра визначає масштаб локальних патернів, які може захоплювати модель. Менші фільтри краще виявляють дрібні деталі, тоді як більші фільтри захоплюють більші структури.

Активація (activation) визначає нелінійність моделі та її здатність до моделювання складних функціональних залежностей. Вибір активаційної функції може значно вплинути на здатність мережі до навчання та загальної продуктивності.

Розмір пулінгу (pooling) визначає ступінь зменшення розмірності після кожного шару згортки. Зменшення розмірності допомагає знизити обчислювальні витрати та уникнути перенавчяння, але надмірне зменшення може втратити важливу інформацію [21].

Ймовірність відсіву (dropout) використовується для регуляризації моделі, допомагаючи уникати перенавчяння шляхом випадкового вимкнення нейронів під час навчання. Це сприяє кращій узагальнюваності моделі на нових даних.

Крок згортки (stride) визначає, наскільки далеко зсувається фільтр під час згортки. Менші кроки дають змогу захоплювати більше деталей, але збільшують обчислювальні витрати.

Доповнення (padding) визначає, чи додавати нулі навколо входу зображення, щоб зберегти розмірність після згортки. Це впливає на розмір виходу кожного шару та здатність моделі до виявлення країв зображення.

Швидкість навчання (learning rate) впливає на швидкість та стабільність процесу навчання. Висока швидкість навчання може призвести до нестабільного навчання, тоді як низька швидкість може значно сповільнити процес.

Бета-1 (beta1) та бета-2 (beta2) є параметрами оптимізатора Adam, які визначають швидкість та стабільність збіжності. Вони впливають на те, як швидко та стабільно оптимізатор коригує ваги моделі під час навчання.

Правильний вибір цих параметрів може значно покращити ефективність навчання.

Зменшення швидкості навчання визначає, як швидкість навчання зменшується протягом тренування. Це може допомогти досягти більш точної збіжності на пізніх етапах навчання.

Моментум використовується в методах оптимізації, таких як SGD (стохастичний градієнтний спуск). Допомагає прискорити навчання в напрямках з послідовними градієнтами, пом'якшуючи осциляції.

Пакетна нормалізація (Batch Normalization) Включає нормалізацію вхідних даних для кожного шару до середнього значення 0 і стандартного відхилення 1, що може прискорити навчання і зробити його стабільнішим.

$L1$ і $L2$ регуляризація додає пеналізацію до функції втрат за велику величину ваг. Це допомагає уникати перенавчання шляхом накладання штрафу на великі ваги моделі.

Раннє зупинення припиняє навчання, якщо продуктивність на валідаційному наборі даних не покращується протягом певної кількості епох.

2.2.4 Застосування ЗНМ

ЗНМ в сучасних технологіях знаходить широке застосування в різних галузях, від медичних досліджень до обробки природної мови. Вони є потужним інструментом аналізу зображень та тексту, що дає змогу робити швидкі та точні прогнози, розвивати системи розпізнавання та здійснювати складні аналізи даних [22].

За допомогою ЗНМ можна аналізувати медичні зображення, такі як МРТ (магнітно-резонансна томографія), рентген та комп'ютерна томографія (КТ). Результат аналізу та класифікації може бути використаний медичними експертами або радіологами для передбачення або встановлення важкості

захворювань, таких як COVID-19, кашель, рак головного/молочної залози та пухлини.

ЗНМ можуть також бути використані у розробці систем розпізнавання облич, що є спеціальним класом розпізнавання зображень через складність. Це охоплює соціальні медіа та ідентифікацію. Такі зображення можуть включати обличчя людей або інших живих істот, таких як тварини, риби та комах.

Виявлення ліків за допомогою прогнозувальної аналітики є ще одним важливим напрямом у галузі охорони здоров'я, який широко використовує ЗНМ. Конвеєр виявлення та розробки ліків є довгим і складним і залежить від численних факторів. Підходи машинного навчання покращують виявлення та прийняття рішень для чітко визначених питань з великою кількістю даних. Розробка нових ліків складна через необхідність обробки великих наборів даних. ЗНМ використовуються для вилучення інформації з різних наборів даних, що призводить до точних інтерпретацій у біологічних дослідженнях, таких як фармакогеноміка [23]. Ще одна проблема полягає в тому, що виявлення та розробка ліків є часомістким і витратним процесом, де масштабність та ефективність важливі. Процес розробки нових ліків добре підходить для застосування нейронних мереж, що дозволяє скоротити час обробки великих обсягів даних. Алгоритм ЗНМ ефективний у оптимізації та оптимізації процесу виявлення ліків на критичних етапах та дає змогу зменшити час, необхідний для розробки лікування для нових захворювань.

Обробка природної мови, є важливим напрямком у галузі комп'ютерів та штучного інтелекту та є автоматичною обчислювальною технологією для аналізу та представлення людської мови. ЗНМ швидко розвиваються в області проектування та обчислення обробки природної мови. ЗНМ використовувалися у вирішенні проблем розпізнавання мов, виявлення спаму чи категоризація тем.

Висновки по розділу 2

ГАє ефективним методом пошуку, який базується на імітації природних процесів відбору та еволюції. Їх робота ґрунтується на поняттях з генетики, таких як популяція, хромосоми, гени та фенотипи. Головною метою генетичних алгоритмів є знаходження оптимального розв'язку задачі шляхом ітеративного пошуку та еволюції популяції.

У процесі роботи генетичного алгоритму ініціалізується початкова популяція, яка складається з хромосом. Кожна хромосома кодує потенційний розв'язок задачі. Шляхом застосування генетичних операцій, таких як кросинговер та мутація, генеруються нові покоління хромосом, які потім оцінюються за допомогою функції пристосованості. Найбільш пристосовані особини відбираються для наступного покоління, що сприяє конвергенції до оптимального розв'язку.

Застосування генетичних алгоритмів дає змогу ефективно вирішувати різноманітні задачі оптимізації та пошуку в різних областях, від інженерії до біоінформатики. Вони виявляються особливо корисними в тих випадках, коли інші методи пошуку можуть застрягати в локальних мінімумах. Генетичні алгоритми продовжують залишатися активним напрямком досліджень у сфері штучного інтелекту та оптимізації завдяки їхній здатності до адаптації та широкому спектру застосувань.

ЗНМ є потужним інструментом у сфері машинного навчання. Вони виникли з бажанням моделювати біологічний процес візуального сприйняття і здатні відокремлювати характеристики з даних за допомогою структурної згортки.

Принцип роботи ЗНМ базується на кількох ключових компонентах, включаючи згортку, доповнення, крок, пулінг та функції активації. Згортка використовується для виділення ознак, доповнення для контролю розміру, крок

для керування щільністю, а пулінг для зменшення кількості параметрів і уникнення перенавчання.

ЗНМ представляють собою широкий клас архітектур, які постійно еволюціонують для покращення їхньої продуктивності та ефективності. Вони здатні автоматично витягати корисні ознаки з великих об'ємів даних, що робить їх вкрай корисними для багатьох завдань у сферах комп'ютерного зору, обробки зображень та розпізнавання образів.

Гіперпараметри згорткових нейронних мереж визначають ключові аспекти їх структури та навчання, впливаючи на їхню продуктивність та ефективність. Вибір оптимальних значень цих параметрів є критичним завданням у процесі розробки моделі, оскільки він визначає якість роботи мережі, її здатність до навчання та узагальнення на нові дані. Кожен з гіперпараметрів, такий як кількість шарів, кількість фільтрів, активація та інші, має велике значення для досягнення оптимальних результатів. Розуміння впливу кожного з цих параметрів дає змогу здійснювати більш ефективну оптимізацію моделі та покращувати її продуктивність. Однак, важливо також враховувати взаємодію між різними гіперпараметрами та їх вплив на весь процес навчання та використання моделі в практичних задачах.

ЗНМ знаходять широке застосування в сучасних технологіях, включаючи медичні дослідження, обробку природної мови та системи розпізнавання облич. Вони дають змогу робити швидкі та точні прогнози, включаючи аналіз медичних зображень для класифікації хвороб, розробку нових ліків та обробку тексту для виявлення спаму та категоризації тем. ЗНМ виявляються важливим інструментом для вирішення складних завдань у різних галузях.

РОЗДІЛ 3 АРХІТЕКТУРА ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору архітектур ГА та ЗНМ

Для побудови генетичного алгоритму було обрано наступні методи та оператори: шістнадцятковек кодування, відбір турніром, рівномірна мутація, одноточкове схрещщеня.

Система кодування у вигляді шістнадцяткового представлення вибрана через її простоту та компактність. Кодування параметрів індивіда у вигляді шістнадцяткових чисел дає змогу легко представляти та обробляти ці дані, що спрощує реалізацію алгоритму.

Відбір турніром обрано через його простоту та ефективність. Цей метод дає змогу обирати батьків з популяції на основі їхньої придатності, при цьому не вимагаючи сортування популяції. Турнірний відбір ефективний для задач оптимізації, де необхідно забезпечити широкий спектр різноманітності серед батьків.

Рівномірна мутація вибрана головним чином через свою універсальність та можливість застосування незалежно від важливості порядку символів у рядку. Навіть у випадках, коли порядок символів має значення, рівномірна мутація може ефективно впливати на кожен символ рядка окремо, зберігаючи при цьому його важливий порядок. Цей підхід дає змогу змінювати параметри рядка, враховуючи специфіку завдання, при цьому не порушуючи його основної структури. Ще однією важливою особливістю рівномірної мутації є можливість варіювати діапазон вибору чисел, що є критичним для оптимізації роботи.

Одноточкове схрещення було обране через його простоту та ефективність у виборі нових індивідів. Цей оператор дає змогу обмінюватись частинами генетичної інформації між батьками, створюючи нових потомків з комбінацією характеристик батьків. Одноточкове схрещення забезпечує простоту інтерпретації та реалізації, що є важливим у випадку реалізації алгоритму.

При застосуванні генетичного алгоритму для налаштування ЗНМ обрано такі параметри: кількість шарів, кількість фільтрів, розмір фільтра, активація, розмір пулінгу, ймовірність відсіву, крок згортки, доповнення, швидкість навчання, бета-1 та бета-2. Ці параметри вибрані для налаштування ЗНМ, оскільки вони безпосередньо впливають на здатність моделі до навчання, узагальнювання та ефективність виконання завдань розпізнавання зображень.

Для генерації та мутації популяцій генетичного алгоритму були застосовані обмеження для кожного гіперпараметра. Це обмеження було введено для забезпечення того, щоб гіперпараметри знаходилися в прийнятних межах, що сприяє збалансованому розвитку моделі та запобігає надмірної складності або неприпустимих значень.

Для моделі ЗНМ було обрано задача розпізнавання пухлин по МРТ головного мозку. Однаково для тренування та тестування нейронної мережі було використано зображення одного з видів пухлин: гліомна, менінгіомна, пухлина гіпофіза або зображення на якому пухлини відсутні.

3.2 Архітектура додатку

Для побудови додатку було використано наступні елементи: класи `CNNIndividual`, `GeneticAlgorithm`, функції `load_data`, `main`, `plot_fitness_dot`, `show_image`, `plot_generations_accuracy`, та два словника `padding_mapping` та `activation_mapping`.

Клас `CNNIndividual` представляє окремий індивід у рамках генетичного алгоритму, який використовується для оптимізації архітектур згорткових нейронних мереж. Цей клас включає всі необхідні методи для створення, тренування, оцінки, кодування/декодування та мутації окремого індивіда.

Ініціалізація класу здійснюється через метод `__init__`, який приймає параметри, що визначають архітектуру ЗНМ: кількість згорткових шарів, кількість фільтрів у кожному згортковому шарі, розмір фільтра, тип активаційної функції, розмір пулінгу, ймовірність відсіву, крок згортки, тип доповнення, швидкість навчання, а також параметри оптимізатора Adam – бtта-1 та бета-2. Метод `__repr__` надає текстове представлення індивіда, що дає змогу зручно відображати його параметри.

Метод `construct_model` будує та тренує модель на основі поточних параметрів індивіда. Модель складається з вхідного шару, вказаної кількості згорткових шарів із зазначеними параметрами, шарів пулінгу та відсіву (якщо задано), шарів вирівнювання (flatten) та вихідного шару з функцією softmax. Модель компілюється з використанням оптимізатора Adam та тренується на наданих тренувальних і валідаційних даних.

Метод `evaluate_fitness` оцінює придатність індивіда на основі точності моделі на валідаційних даних. Оцінка зберігається в атрибуті fitness.

Метод `encode` перетворює параметри індивіда в закодований рядок, що корисно для здійснення схрещування між індивідами. Закодований рядок включає всі параметри, представлені у вигляді шістнадцяткових чисел. Метод `decode` є класовим методом, який перетворює закодований рядок назад у параметри індивіда, що використовується для створення індивіда з закодованого представлення.

Метод `mutate` виконує випадкові зміни в параметрах індивіда з заданою ймовірністю мутації, що дає змогу генетичному алгоритму досліджувати нові конфігурації ЗНМ, уникаючи застою в локальних мінімумах.

Клас `GeneticAlgorithm` реалізує генетичний алгоритм, спеціально розроблений для оптимізації архітектур ЗНМ. Ініціалізація класу відбувається через метод `__init__`. Він приймає розмір популяції як аргумент і ініціалізує кілька атрибутів: розмір популяції, поточне покоління, список найкращих

індивідів кожного покоління та словник для збереження даних про придатність (fitness) різних параметрів.

Метод `initialize_population` створює початкову популяцію, яка складається з випадково згенерованих індивідів. Кожен індивід представляє архітектуру ЗНМ і визначається такими параметрами, як кількість згорткових шарів, кількість фільтрів, розмір фільтра, активаційна функція, розмір пулінгу, ймовірність відсіву (dropout probability), значення бета-1 та бета-2 для оптимізатора, крок, тип паддінгу і швидкість навчання.

Метод `selection` здійснює відбір батьків для наступного покоління за допомогою турнірного відбору.

Метод `crossover` реалізує одноточкове схрещування, яке створює нащадків шляхом комбінування закодованих представлень двох батьків.

Метод `evaluate_individual` оцінює придатність окремого індивіда за допомогою перевірочних даних. Після оцінки придатності, метод оновлює словник даних про придатність, додаючи інформацію про поточного індивіда.

Метод `evaluate_population_fitness` оцінює придатність всієї популяції паралельно за допомогою потоків (threading). Для кожного індивіда створюється окремий потік, що дає змогу оцінювати їх одночасно, що значно прискорює процес.

Метод `fit_population` тренує кожного індивіда в популяції також паралельно за допомогою потоків. Це включає навчання моделей на тренувальних даних і їх валідацію.

Метод populate керує процесом еволюції протягом кількох поколінь. Він починається з тренування початкової популяції, а потім виконує ітерації через наступні покоління. В кожному поколінні оцінюється придатність популяції, обираються батьки, здійснюється схрещування та мутація для створення нових нащадків, які замінюють стару популяцію. Метод також відстежує та виводить найкращу придатність для кожного покоління.

Функція `load_data` використовується для завантаження зображень і міток з заданого каталогу. Вона зчитує зображення, змінює їх розмір, перетворює в відтінки сірого, нормалізує та конвертує мітки в one-hot представлення.

Функція `plot_fitness_dot` відповідає за візуалізацію залежності точності від значень певного параметра. Вона будує графік розсіювання, де по осі *X* відкладаються значення параметра, а по осі *Y* – точність.

Функція `show_image` відображає одне зображення з його міткою.

Функція `plot_generations_accuracy` будує графік зміни точності кращого індивіда в кожному поколінні.

Основна функція `main` ініціалізує генетичний алгоритм з розміром популяції *n*. Завантажуються тренувальні та валідаційні дані. Виводиться кількість тренувальних та валідаційних прикладів, а також відображається випадкове тренувальне зображення з його міткою. Далі генетичний алгоритм запускається для оптимізації архітектур ЗНМ протягом *m* поколінь. Найкращий індивід кожного покоління зберігається, і будується графік точності найкращих індивідів за поколіннями. Після завершення оптимізації виводиться інформація про найкращого індивіда. Також будуються графіки залежності точності від різних параметрів.

Словники `activation_mapping` та `padding_mapping` використовуються для зіставлення числових значень з відповідними текстовими описами активаційних функцій та типів заповнення в архітектурі нейронної мережі. Це дає змогу зручно використовувати числові значення для вибору активаційних функцій та типу заповнення під час побудови моделі, а також для зберігання та передачі конфігурації моделі у компактному форматі. Структура додатку у вигляді дерева представлена на рис. 3.1.

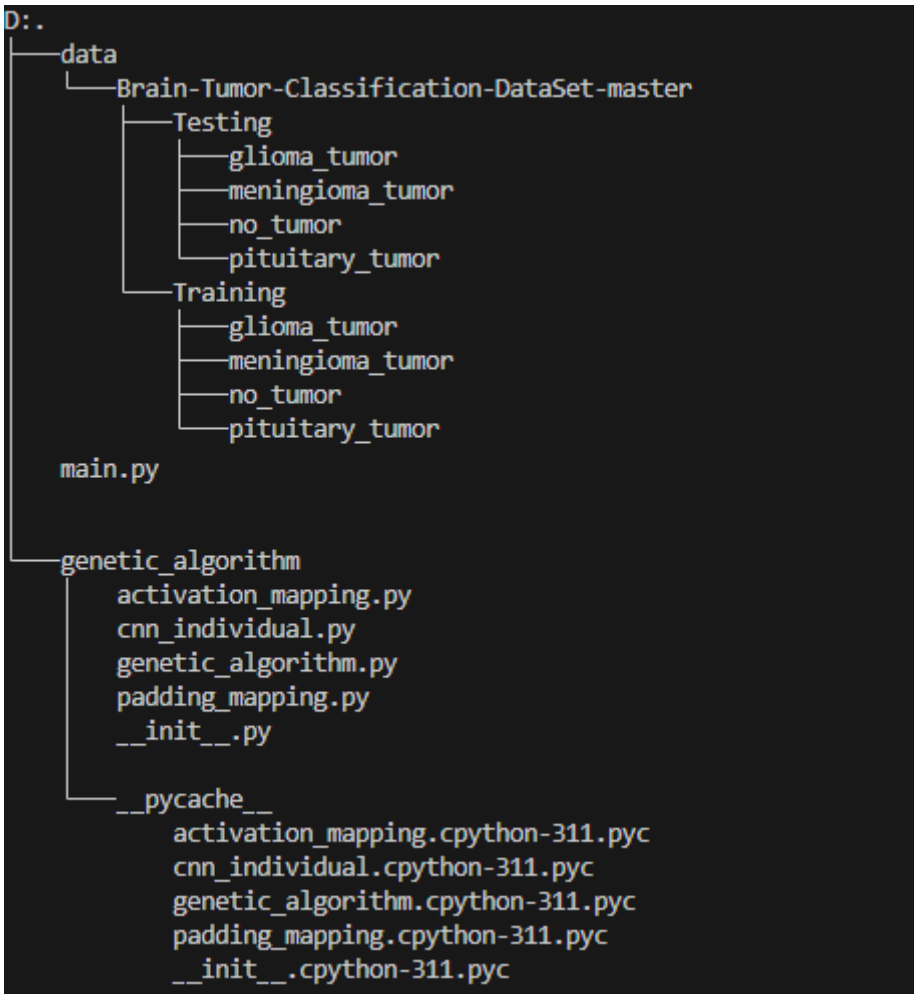

Рисунок 3.1 – Структура проекту

Варто зазначити, що через велику кількість даних, розміщених у каталозі data, на рис. 3.1 зображено лише дерево каталогів у шляху data без файлів, що містяться всередині них.

3.3 Використані пакети та алгоритми

Для реалізації програми було використано такі бібліотеки:

– numpy;

– os;

– random;
– tensorflow;
– matplotlib;
– cv2.

Модуль os використовується для роботи з файловою системою, зокрема для навігації по директоріях і отримання списків файлів і каталогів. cv2 (OpenCV) використовується для обробки зображень.

Модуль random було використано для ініціалізація випадкових значень. Пакетний модуль tensorflow використано для побудови та тренування моделі. numpy використано для зручної роботи з масивами та математичним операціями. Пакет matplotlib використовується для побудови графіків.

3.4 Оцінка результатів

Після запуску програми на екран виводиться приклад обробленого зображення, які використовуються для тренування ЗНМ моделі. На рис. 3.2 та рис. 3.3 зображено оригінальне та оброблене зображення, що показує, як вихідні дані трансформуються для підготовки до навчання моделі.

По закінченню ітерації ГА в консоль виводиться інформація про найкращу точність моделі в популяції (рис 3.4). Після завершення ітерація виводиться інформація про найкращу особину в популяції (рис. 3.5).

Після завершення роботи алгоритму на екран виводяться графіки результатів. Найвищу точність моделі з усієї популяції на кожній ітерації зображено на рис. 3.6.

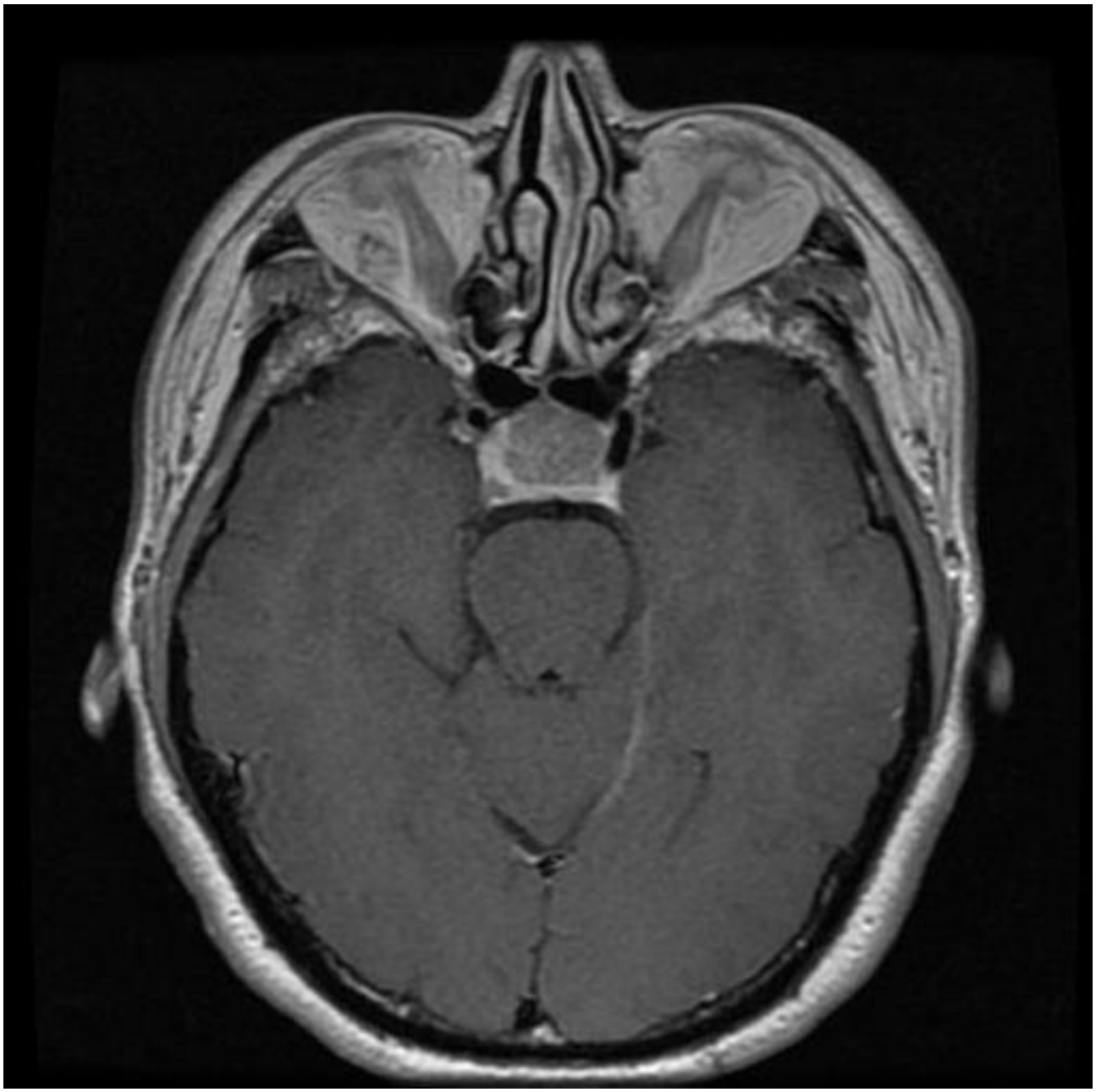

Рисунок 3.2 – Приклад оригінального зображення

На графіку (рис. 3.6) бачимо, що перші кілька поколінь ГА суттєво покращують точність, що свідчить про ефективність алгоритму в знаходженні кращих параметрів. Стабілізація після 10-го покоління може вказувати на досягнення оптимуму або наближення до нього.

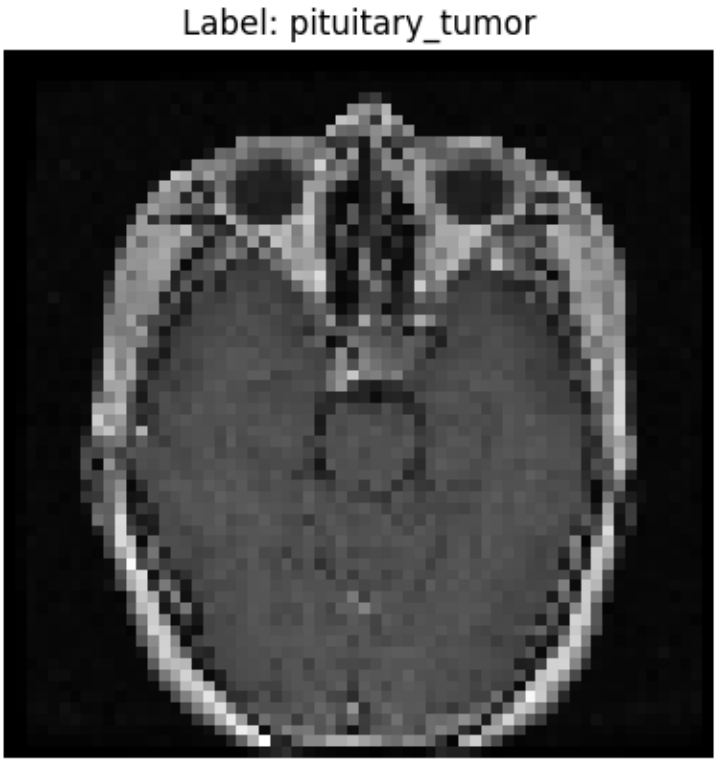

Рисунок 3.3 – Приклад обробленого зображення

```
Generation 20 of 30. Best fitness: 0.7741116881370544
Generation 21 of 30. Best fitness: 0.7741116881370544
Generation 22 of 30. Best fitness: 0.7741116881370544
Generation 23 of 30. Best fitness: 0.7741116881370544
Generation 24 of 30. Best fitness: 0.7741116881370544
Generation 25 of 30. Best fitness: 0.7741116881370544
Generation 26 of 30. Best fitness: 0.7741116881370544
Generation 27 of 30. Best fitness: 0.7741116881370544
Generation 28 of 30. Best fitness: 0.7741116881370544
Generation 29 of 30. Best fitness: 0.7741116881370544
Generation 30 of 30. Best fitness: 0.7741116881370544
```

Рисунок 3.4 – Інформація про найкращий результат ітерацій ГА

```
Best individual: CNNIndividual(num_conv_layers=1, num_filters=56, filter_size=3, activation=sigmoid, pooling_size=2, dropout_prob=0.0, stride=1, padding=valid, learning_rate=0.001, beta_1=0.991, beta_2=0.974)
```

Рисунок 3.5 – Найкраща особина після завершення роботи ГА

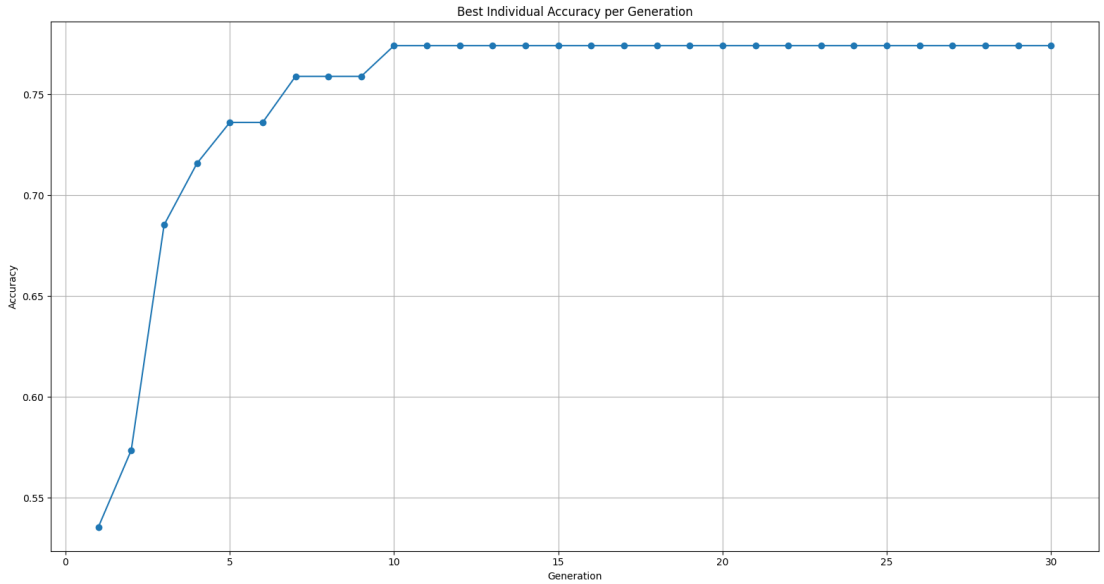

Рисунок 3.6 – Найвища точність моделей на кожній ітерації

Аналізуючи графік бета-1 (рис. 3.7) бачимо, що значення бета-1 від 0.90 до 1.00 демонструють високу концентрацію точності, нижчі значення в свою чергу демонструють ширший розкид і загалом нижчу точність. З цього можемо зробити висновок, що вибір значення beta_1 в діапазоні від 0.90 до 1.00 є оптимальним для досягнення високої точності.

На графіку бета-2 (рис 3.8) видно, що значення близькі до 0.97 мають високу точність з невеликою дисперсією. Значення ж наближені до 1 мають широкий розкид, через що важко зробити однозначні висновки.

Основуючись на графіку швидкості навчання (рис.3.9) моделі які наближені приблизно до 0.002 мають найбільшу точність. Моделі які мають приблизне значення альфа 0.001 теж мають доволі високу точність, хоча і більшу дисперсію.

Широкий розкид бачимо на графіках параметрів ймовірності відсіву, розмірів пулінгу та кроку згортки, розміру фільтрів, виду доповнення та функції активації (рис. 3.10 – 3.15). Через це та недостатність даних, ефективність цих параметрів на точність моделі важко оцінити, хоча вони можуть мати вирішальне значення на швидкість тренування моделі.

Роблячи висновок ґрунтуючись на графіку кількості фільтрів (рис. 3.16) можна зауважити, що найкращою кількістю фільтрів для моделі є 56 та 60.

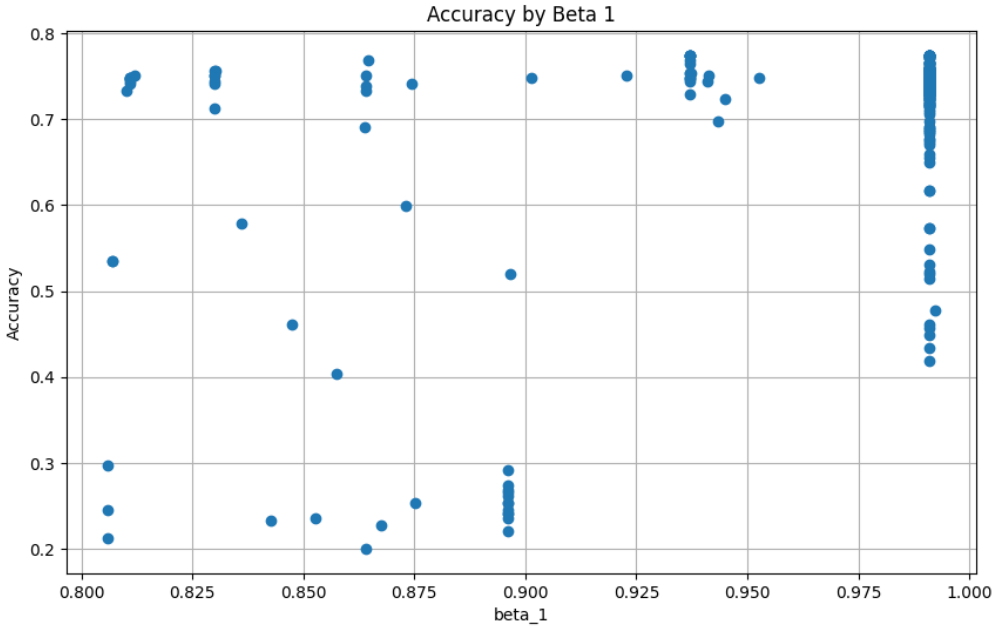

Рисунок 3.7 – Залежність точності моделі від бета-1

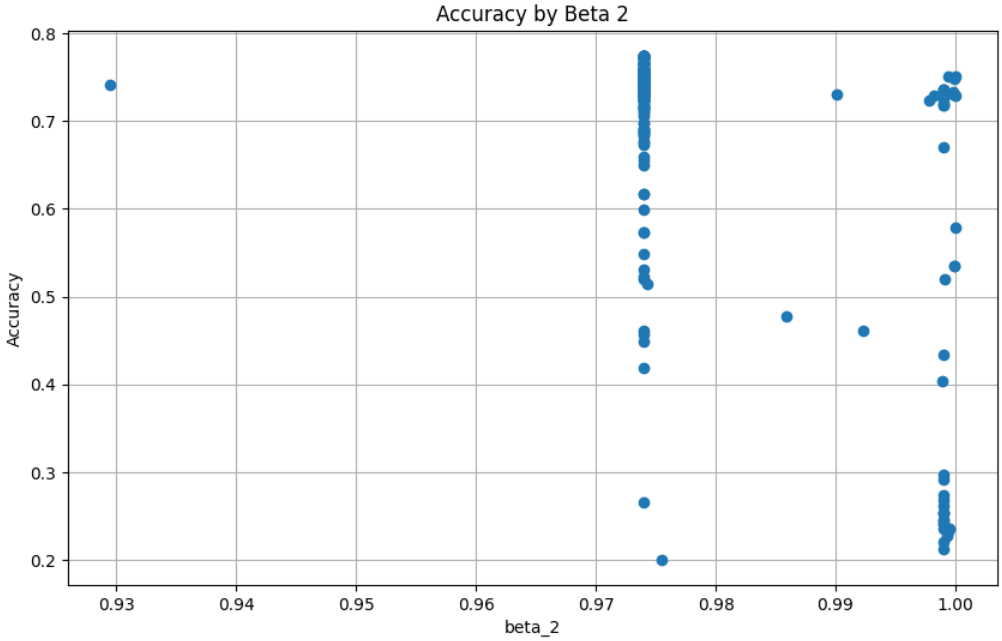

Рисунок 3.8 – Залежність точності моделі від бета-2

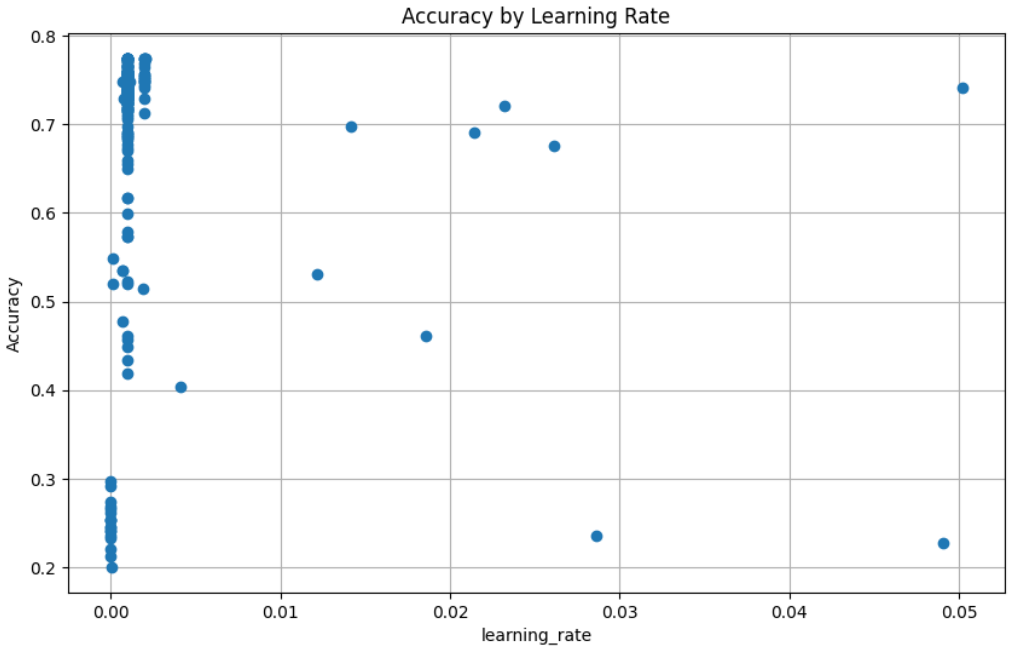

Рисунок 3.9 – Залежність моделі від параметра швидкості навчання альфа

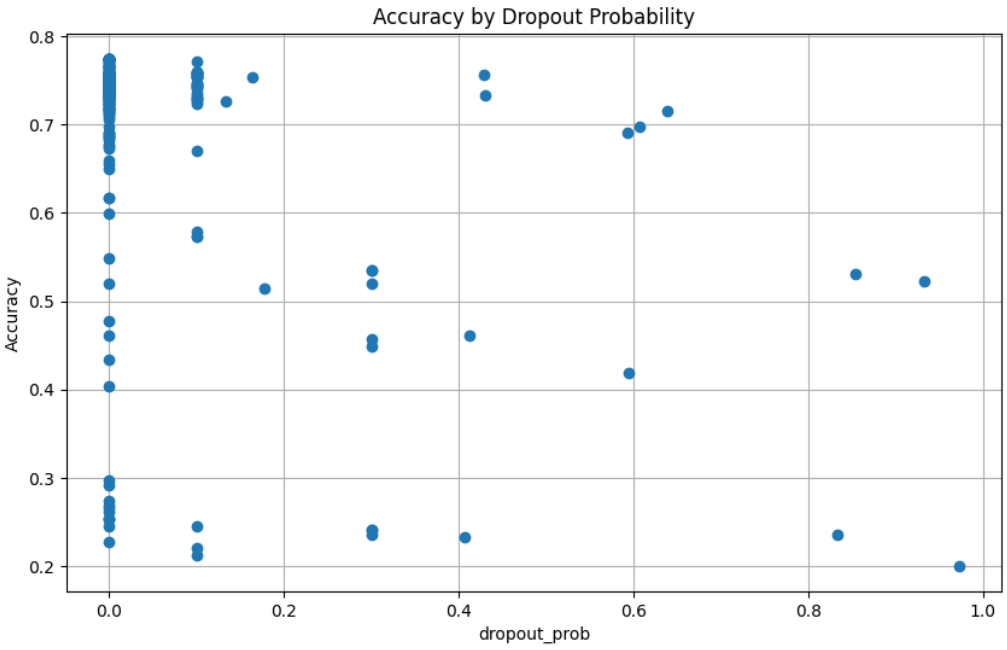

Рисунок 3.10 – Залежність точності моделі від ймовірності відсіву

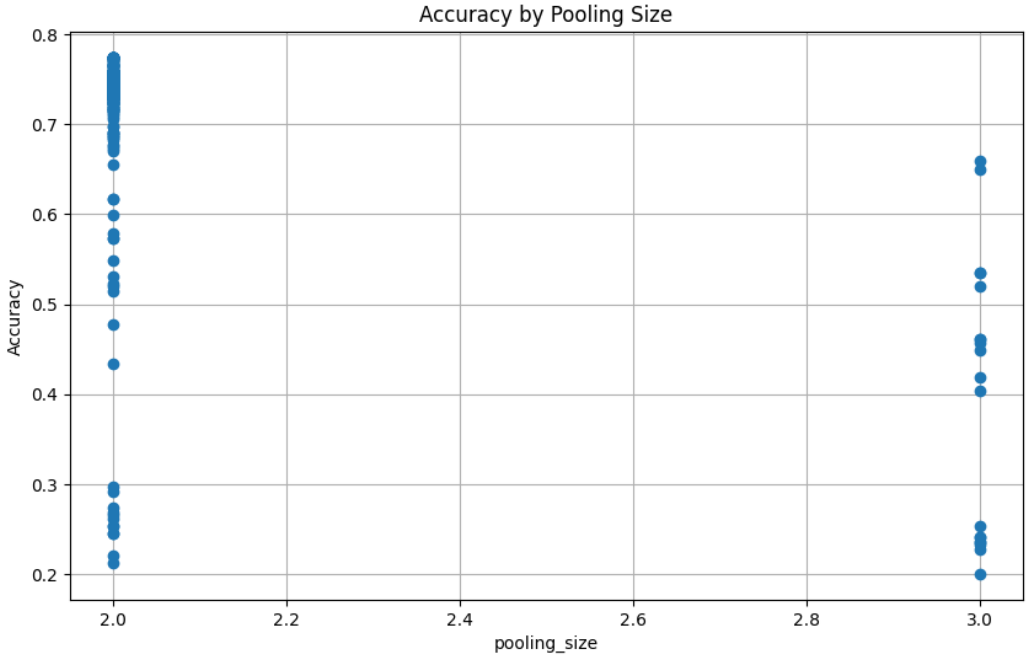

Рисунок 3.11 – Залежність точності моделі від розміру пулінгу

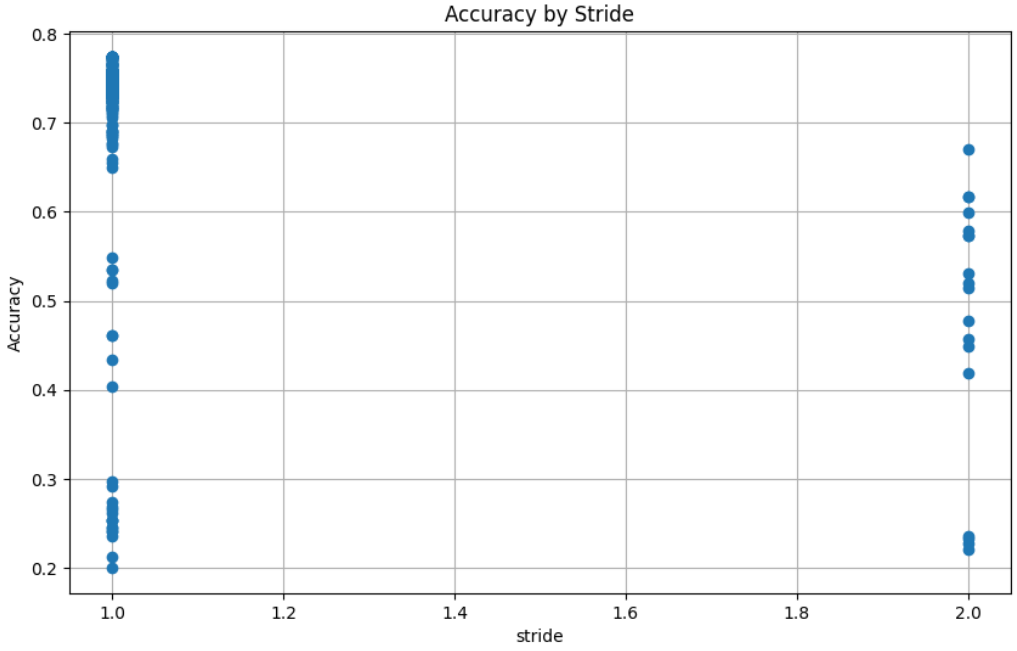

Рисунок 3.12 – Залежність точності моделі від кроку згортки

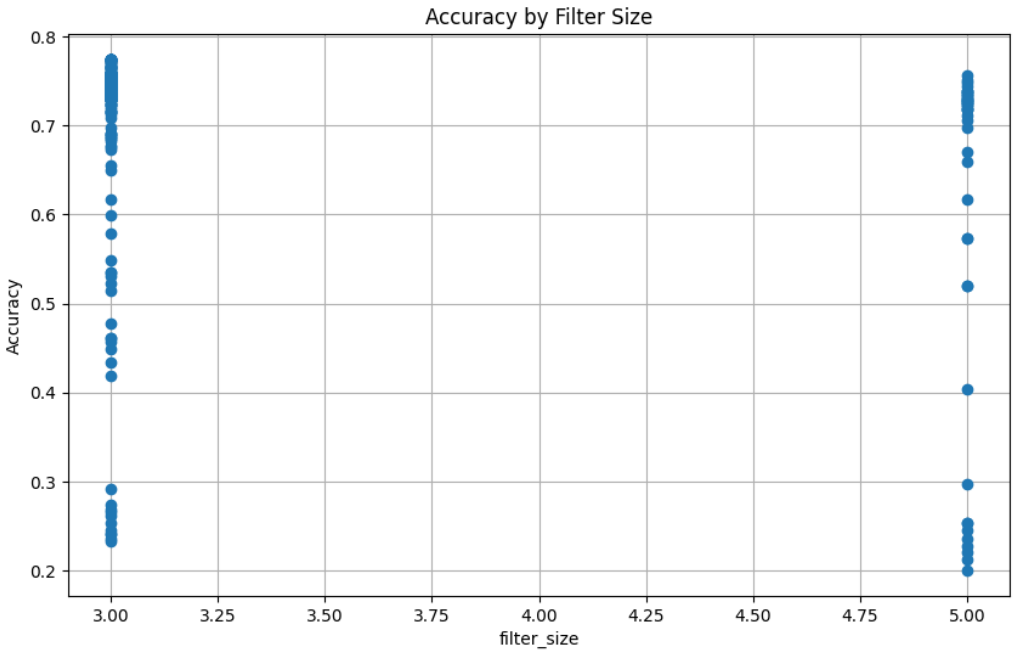

Рисунок 3.13 – Залежність точності моделі від розміру фільтрів

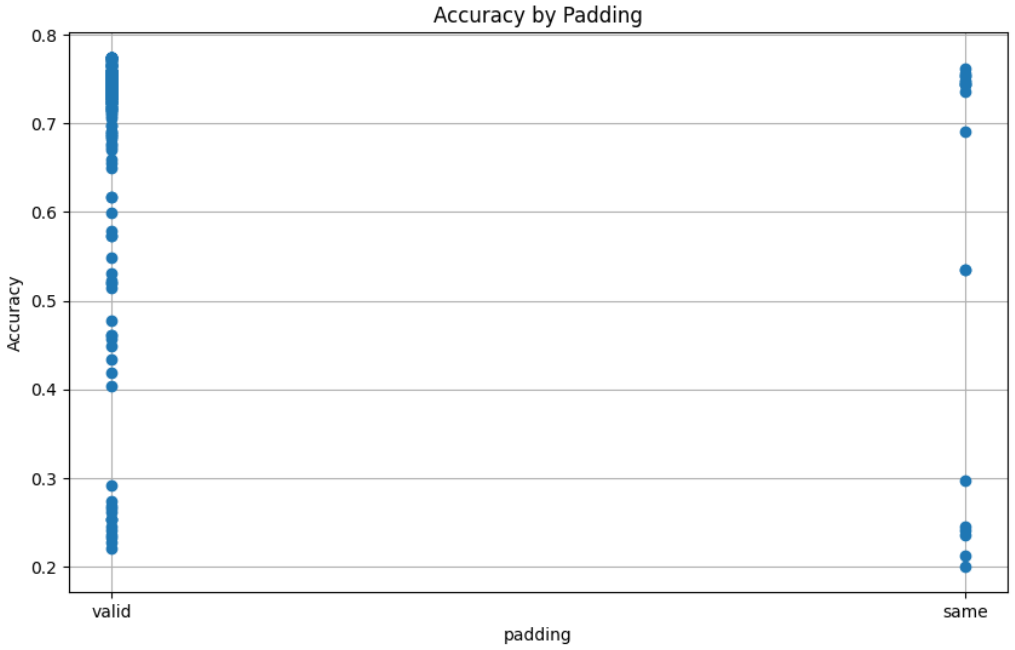

Рисунок 3.14 – Залежність точності моделі від виду доповнення

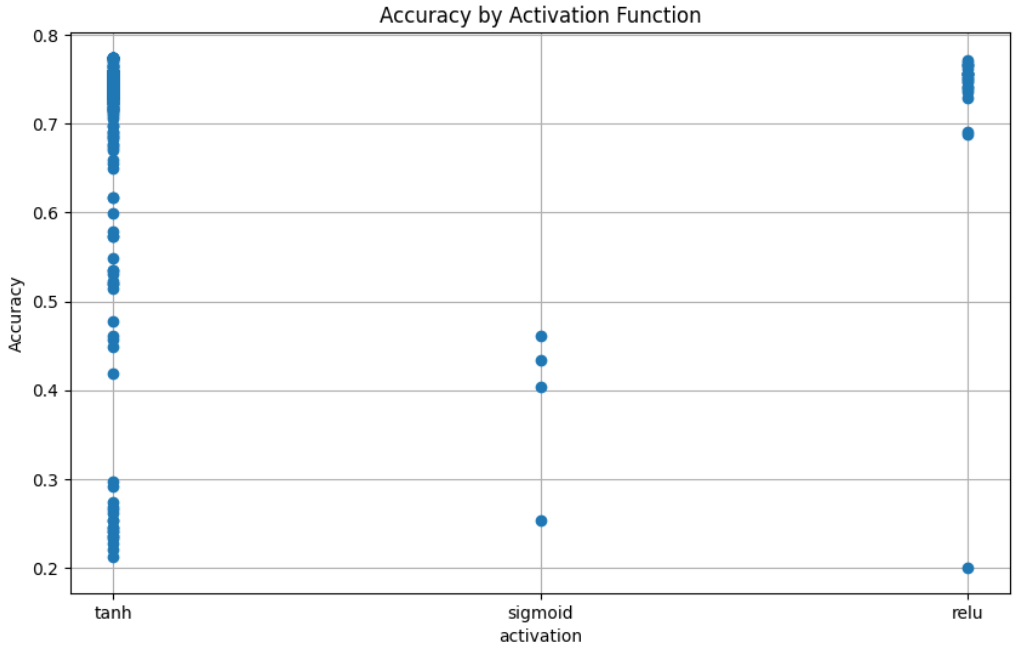

Рисунок 3.15 – Залежність точності від функції активації

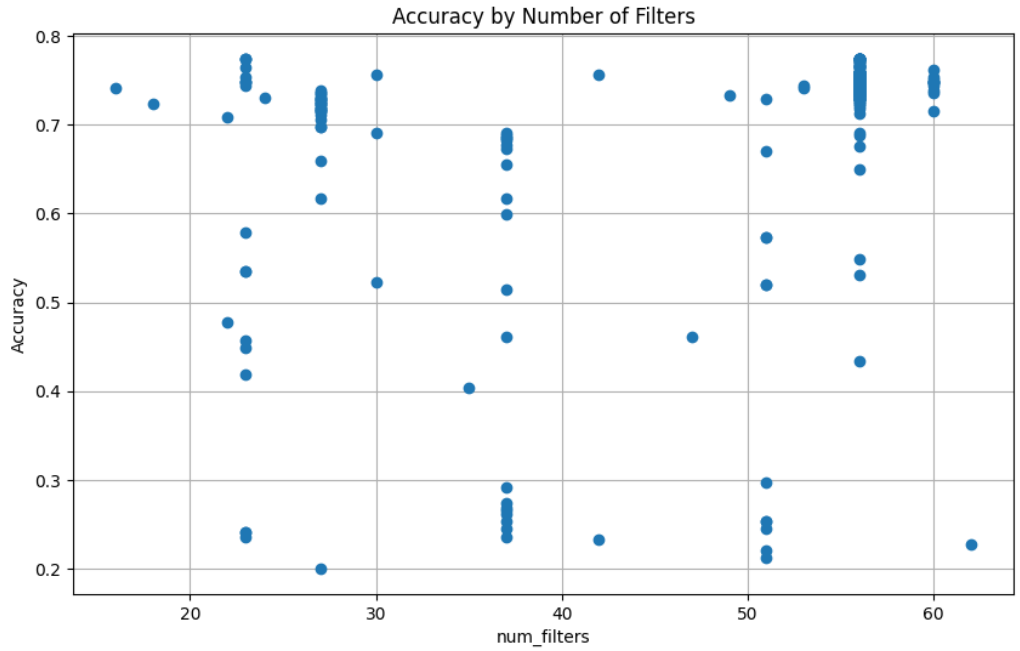

Рисунок 3.16 – Залежність точності моделі від кількості фільтрів

Висновки по розділу 3

Побудовано генетичний алгоритм з використанням шістнадцяткового кодування, відбору турніром, рівномірної мутації та одноточкового схрещення для оптимізації параметрів ЗНМ. Обрані методи та оператори відрізняються простотою та ефективністю, сприяючи оптимізації параметрів моделі.

Система кодування у вигляді шістнадцяткового представлення обрана через її простоту та компактність, що спрощує обробку даних та реалізацію алгоритму. Відбір турніром вибрано через його ефективність та можливість вибору батьків на основі придатності, не вимагаючи сортування популяції. Рівномірна мутація вибрана за свою універсальність та можливість варіювання параметрів незалежно від їхнього порядку. Одноточкове схрещення обране за простоту використання та ефективність у створенні нових потомків.

Параметри ЗНМ, які було оптимізовано, включають кількість шарів, кількість фільтрів, розмір фільтра, активацію, розмір пулінгу, ймовірність відсіву, крок згортки, доповнення, швидкість навчання, бета-1 та бета-2. Обмеження для кожного гіперпараметра було введено для забезпечення прийнятних меж параметрів та запобігання надмірної складності моделі.

Генетичний алгоритм застосовано для розв'язання задачі розпізнавання пухлин по комп'ютерній томографії головного мозку. Аналіз графіків показав, що алгоритм ефективно покращує точність моделі протягом перших кількох поколінь, але досягає стабільності після певної кількості поколінь.

Під час аналізу параметрів було виявлено, що високі значення бета-1 та бета-2 сприяють підвищенню точності моделі. Щодо швидкості навчання, оптимальні значення знаходяться приблизно в діапазоні від 0.001 до 0.002. Широкий розкид параметрів як ймовірність відсіву, розмір пулінгу, крок згортки та інші, свідчить про їх незначний вплив на точність моделі. Визначено, що оптимальні значення кількості фільтрів знаходяться приблизно в діапазоні 56-60.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в цьому дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дає змогу оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки методу генетичних алгоритмів для машинного налаштування згорткових нейронних мереж. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту такі:

- функціонування на персональних комп’ютерах із стандартним набором компонентів;
- швидкість обробки даних та навчання моделей;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту для підтримки прийняття рішень для торгівлі акціями. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) JavaScript.

Функція F_2:

а) Tensorflow.

б) Torch.

Функція F_3:

а) Vim.

б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

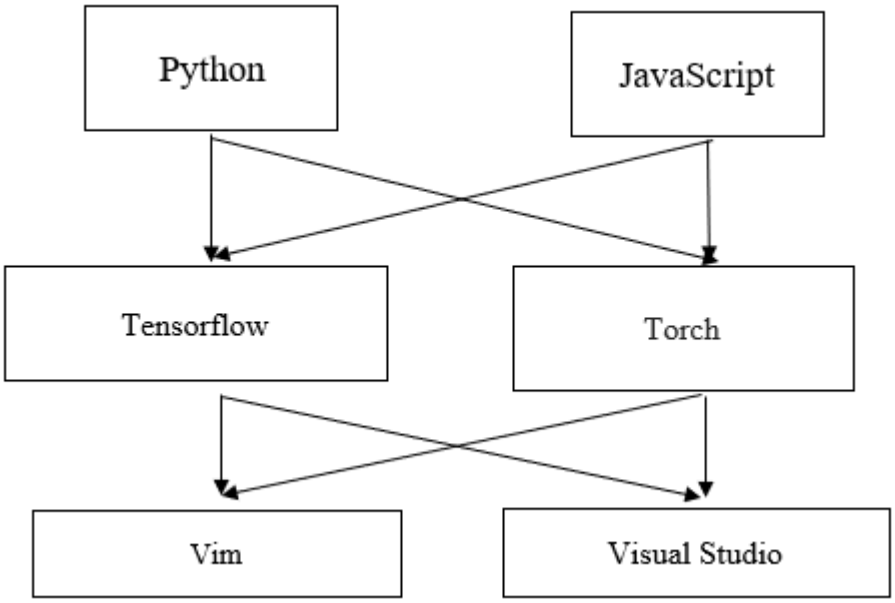

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій. Позитивно-негативна матриця показана в табл. 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	*А*	Простота вивчення та читання коду, багата екосистема бібліотек для різних задач програмування	Відносно низька швидкість роботи
	Б	Велика кількість інструментів та бібліотек	Не така ефективна для обчислювально інтенсивних завдань у порівнянні з іншими мовами
F_2	*А*	Масштабована та ефективна бібліотека для роботи з нейронними мережами та	Вимагає певного часу та зусиль для оволодіння
	Б	Швидкість та ефективність у виконанні обчислень, простота використання та гнучкість у розробці моделей	Менша популярність порівняно з іншими фреймворками, менше розширень та інтеграцій
F_3	*А*	Максимальна продуктивність для досвідчених користувачів завдяки потужним можливостям редактора та можливостям автоматизації	Вимагає значного часу та зусиль для оволодіння, не такий інтуїтивно зрозумілий для новачків
	Б	Має велику кількість розширень та інтеграцію з широким спектром інструментів	Може вимагати значних ресурсів системи

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу віддаємо А через спрощену синтаксичну конструкцію та наявності стандартних бібліотек для обчислення.

Функція F_2:

Обидва варіанти можна використовувати в розробці.

Функція F_3:

Віддаємо перевагу варіанту Б для спрощення роботи по написанню коду.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3б;

F_1а – F_2б – F_3б.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для обчислень та збереження даних;

X3 – тривалість однієї ітерації ГА;

X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позна-чення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	*X1*	оп/мс	10000	14000	19000
Об’єм пам’яті	*X2*	Мб	420	128	64
Тривалість однієї ітерації ГА	*X3*	сек	120	80	70
Потенційний об’єм програмного коду	*X4*	кількість рядків коду	600	500	300

За даними табл. 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

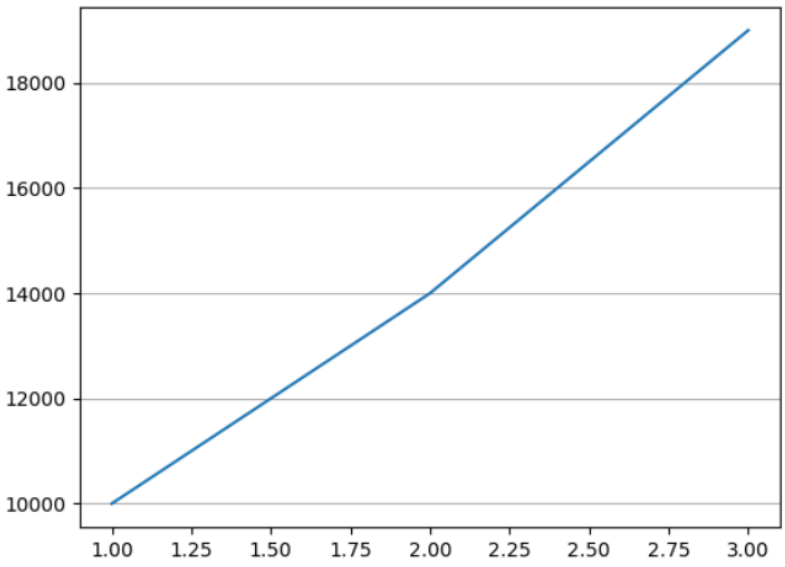

Рисунок 4.2 – *X1*, швидкодія мови програмування

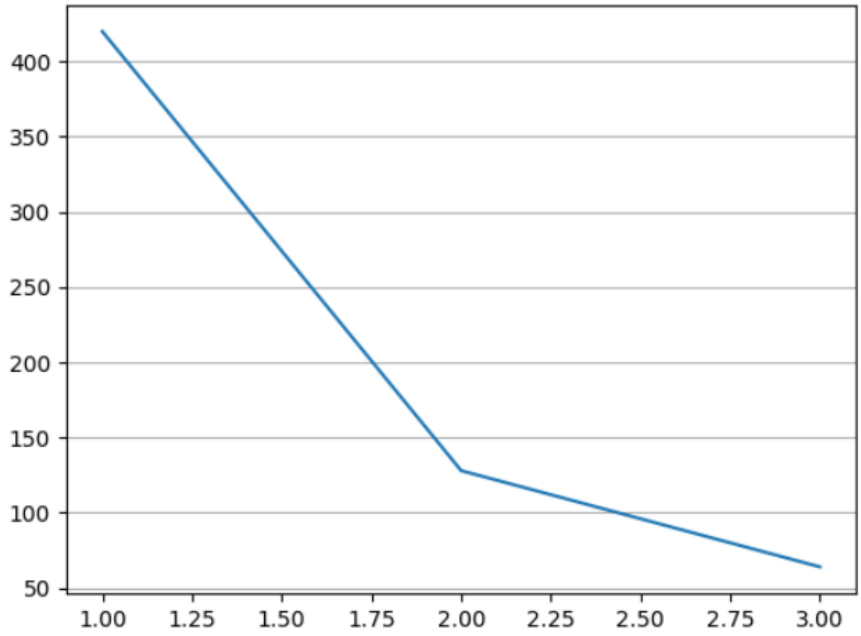

Рисунок 4.3 – *X2*, об’єм пам’яті

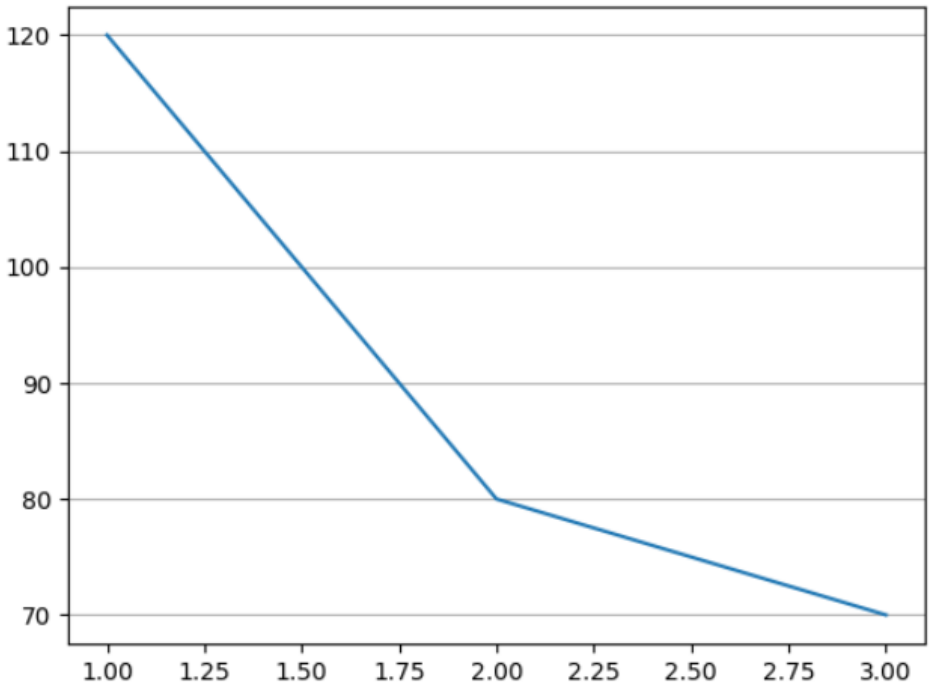

Рисунок 4.4 – *X3*, тривалість однієї ітерації ГА

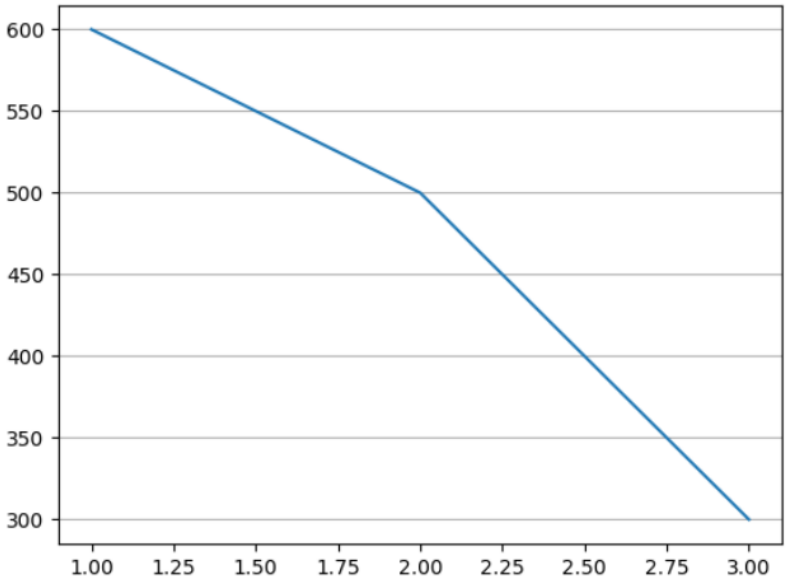

Рисунок 4.5 – *X3*, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у табл. 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							*Сума рангів* R_i	*Відхилення* Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	1	1	2	1	2	10	-7,5	56,25
X2	Об’єм пам’яті	Mb	3	4	3	3	3	4	3	23	5,5	30,25
X3	Тривалість однієї ітерації ГА	сек	4	3	4	4	4	2	4	25	7,5	56,25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	2	1	2	3	1	2	1	12	-5,5	30,25
	Разом		10	10	10	10	10	10	10	70	0	173

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 173. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 173}{7^2(4^3 - 4)} = 0{,}7 > W_k = 0{,}67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у табл. 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і *X2*	<	<	<	<	<	<	<	<	0,5
X1 і *X3*	<	<	<	<	<	<	<	<	0,5
X1 і *X4*	<	>	<	<	>	<	>	<	0,5
X2 і *X3*	<	>	<	<	<	>	>	<	0,5
X2 і *X4*	>	>	>	>	>	>	>	>	1,5
X3 і *X4*	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	*X2*	*X3*	*X4*	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X2	1,5	1	0,5	0,5	3,5	0,22	12,25	0,21	44,875	0,21
X3	1,5	1,5	1	1,5	5,5	0,34	21,25	0,35	77,875	0,36
X4	1,5	1,5	0,5	1	4,5	0,28	16,25	0,28	59,125	0,27
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X1* (швидкодія мови програмування), *X2* (Об’єм пам’яті), *X3* (Тривалість однієї ітерації ГА) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	*А*	*X1*	10000	3	0,16	0,48
F2	*А*	*X4*	400	6	0,27	1,62
	Б	*X4*	600	4	0,27	1,08
F3	*Б*	*X2*	128	5	0,21	1,05

За даними з табл. 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0{,}48 + 1{,}62 + 1{,}05 = 3{,}6,$$

$$K_{K2} = 0{,}48 + 1{,}08 + 1{,}05 = 3{,}06.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТМ}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 46 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та

вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 46 \cdot 1.7 \cdot 0.8 = 62{,}56 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (62{,}56 + 19.44 + 4.8 + 19.44) \cdot 8 = 845{,}44 \text{ людино-годин.}$$

$$T_{II} = (62{,}56 + 19.44 + 6.91 + 19.44) \cdot 8 = 866{,}32 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 18000 грн., один аналітик в області даних з окладом 20000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{M}{T_m \cdot t} \text{ грн,} \qquad (4.14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_Ч = \frac{18000 + 18000 + 20000}{3 \cdot 21 \cdot 8} = 109{,}12 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{ЗП} = C_ч \cdot T_i \cdot K_Д, \quad (4.16)$$

де $C_Ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_{ЗП} = 109{,}12 \cdot 845{,}44 \cdot 1.2 = 110705{,}2$ грн.

II. $C_{ЗП} = 109{,}12 \cdot 866{,}32 \cdot 1.2 = 113439{,}4$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_{ВІД} = C_{ЗП} \cdot 0.22 = 110705{,}2 \cdot 0.22 = 24355{,}14$ грн.

II. $C_{ВІД} = C_{ЗП} \cdot 0.22 = 113439{,}4 \cdot 0.22 = 958{,}67$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($C_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 18000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_Г = 12 \cdot M \cdot K_З = 12 \cdot 18000 \cdot 0{,}2 = 43200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + K_З) = 43200 \cdot (1 + 0.2) = 51840 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0.22 = 51840 \cdot 0{,}22 = 11404{,}8 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$C_A = K_{ТМ} \cdot K_A \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 20000 = 7000 \text{ грн,}$$

де $K_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_Р = K_{ТМ} \cdot Ц_{ПР} \cdot K_Р = 1.2 \cdot 20000 \cdot 0.08 = 1920 \text{ грн,}$$

де $K_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot K_В = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.8 =$$

$$= 1491{,}2 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В, Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491,2 \cdot 0,2 \cdot 0,2 \cdot 5,22 = 311,7 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13600 \text{ грн}.$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$$С_{ЕКС} = 51840 + 11404,8 + 7000 + 1920 + 311,7 + 13600 = 86076,5 \text{ грн}.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 86076,5 / 1491,2 = 57,72 \text{ грн/год}.$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M\text{-}Г} \cdot T, \quad (4.18)$$

I. $C_M = 57{,}72 \cdot 845{,}44 = 48221{,}6$ грн.

II. $C_M = 57{,}72 \cdot 866{,}32 = 50002{,}83$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $C_H = 110705{,}19 \cdot 0{,}67 = 74172{,}48$ грн.

II. $C_H = 113439{,}4 \cdot 0{,}67 = 76004{,}1$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_M + C_H, \quad (4.20)$$

I. $C_{ПП} = 110705{,}2 + 24355{,}14 + 48221{,}6 + 74172{,}48 = 257252{,}42$ грн.

II. $C_{ПП} = 113439{,}4 + 24956{,}67 + 50002{,}83 + 76004{,}4 = 264403{,}3$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{TEPj} = K_{Kj} / K_{Фj}, \quad (4.21)$$

$$K_{TEP1} = 3,6 / 257252,42 = 1,4 \cdot 10^{-5},$$

$$K_{TEP2} = 3,06 / 264403,3 = 1,157 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{TEP1} = 1,4 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{TEP} = 1,057 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– вибір програмного продукту – Python;

– використання моделей з великою ємністю;

– використання стандартного інтерфейсу візуалізації.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

Використання генетичних алгоритмів в задачі машинного налаштування згорткових нейронних мереж виявилося потужним інструментом для оптимізації параметрів моделей у сфері комп'ютерного зору та обробки зображень. Ці алгоритми дають змогу автоматизувати та прискорити процес налаштування, знаходячи оптимальні конфігурації мережі, що підвищує її точність та швидкість роботи.

Застосування генетичних алгоритмів дає змогу здійснювати ефективний пошук у просторі параметрів нейронних мереж, оптимізуючи їхню архітектуру та гіперпараметри. Це особливо важливо в умовах зростаючої складності завдань обробки зображень та збільшення обсягів даних.

Використання генетичних алгоритмів вказує на потенціал для розширення можливостей нейронних мереж у вирішенні реальних завдань обробки інформації. Оптимізація параметрів за допомогою генетичних алгоритмів може покращити якість результатів та зробити моделі більш адаптивними до різних умов та завдань.

Незважаючи на досягнуті успіхи, дослідження у цій області продовжуються, і є потреба у подальших дослідженнях для вдосконалення алгоритмів та їхнього використання. Такий підхід до машинного навчання відкриває широкі перспективи для розвитку нових технологій та застосування їх у різних сферах, включаючи медицину, автомобільну промисловість, аналітику даних та багато інших.

ПЕРЕЛІК ПОСИЛАНЬ

1. Khanzode, Ku. Chhaya A., and Ravindra D. Sarode. "Advantages and Disadvantages of Artificial Intelligence and Machine Learning: A Literature Review." International Journal of Library & Information Science (IJLIS), vol. 9, no. 1, Jan.-Apr. 2020, pp. 30-36. IAEME Publication, doi:10.17605/OSF.IO/GV5T4
2. Trabelsi, Mohamed. (2024). The impact of artificial intelligence on economic development. Journal of Electronic Business & Digital Economics. 3. 10.1108/JEBDE-10-2023-0022.
3. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville (2015), URL: http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf
4. Kennedy, J., and R. Eberhart. (1995) "Particle Swarm Optimization." Proceedings of the IEEE International Conference on Neural Networks, vol. 4. 10.1109/ICNN.1995.488968.
5. Holland, J. H. (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.
6. Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations, URL: https://arxiv.org/pdf/1412.6980
7. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE, URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7352306
8. David E. Goldberg (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. URL:

http://www2.fiit.stuba.sk/~kvasnicka/Free%20books/Goldberg_Genetic_Algorithms_in_Search.pdf

9. Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer-Verlag, NewYork. URL: http://web.ist.utl.pt/adriano.simoes/tese/referencias/Michalewicz%20Z.%20Genetic%20Algorithms%20+%20Data%20Structures%20=%20Evolution%20Programs%20%283ed%29.PDF
10. Chahar, Vijay & Katoch, Sourabh & Chauhan, Sumit. (2021). A Review on Genetic Algorithm: Past, Present, and Future. Multimedia Tools and Applications. 80. 10.1007/s11042-020-10139-6.
11. Lee, Joon-Yong & Kim, Min-Soeng & Kim, Cheol-Taek & Lee, Ju-Jang. (2007). Study on encoding schemes in compact genetic algorithm for the continuous numerical problems. 2694 - 2699. 10.1109/SICE.2007.4421447.
12. Sivanandam SN, Deepa SN (2008) Introduction to genetic algorithm, 1st edn. Springer-Verlag, BerlinHeidelberg
13. Fox B, McMahon M (1991) Genetic operators for sequencing problems, in Foundations of GeneticAlgorithms, G. Rawlins, Ed. Morgan Kaufmann Publishers, San Mateo,CA, Ed. 1991
14. Lee, Chang-Yong. (2003). Entropy-Boltzmann selection in the genetic algorithms. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society. 33. 138-49. 10.1109/TSMCB.2003.808184.
15. Soon, Gan & Tan, Tse Guan & On, Chin & Alfred, Rayner & Anthony, Patricia. (2013). A comparison on the performance of crossover techniques in video game. Proceedings - 2013 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2013. 493-498. 10.1109/ICCSCE.2013.6720015.
16. Desjardins, Ben & Falcon, Rafael & Abielmona, Rami & Petriu, Emil. (2017). Planning Robust Sensor Relocation Trajectories for a Mobile Robot with Evolutionary Multi-objective Optimization. 10.1007/978-3-319-47715-2_8.

17. Umbarkar, Dr. Anantkumar & Sheth, P.. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1). 6. 10.21917/ijsc.2015.0150.
18. Larranaga, Pedro & Kuijpers, Cindy & Murga, R. & Inza, I. & Dizdarevic, S.. (1999). Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review. 13. 129-170. 10.1023/A:1006529012972.
19. Li, Zewen & Liu, Fan & Yang, Wenjie & Peng, Shouheng & Zhou, Jun. (2021). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE Transactions on Neural Networks and Learning Systems. PP. 1-21. 10.1109/TNNLS.2021.3084827.
20. Neural Networks and Deep Learning: A Textbook by Charu C. Aggarwal (2018), URL: http://ndl.ethernet.edu.et/bitstream/123456789/88552/1/2018_Book_NeuralNetworksAndDeepLearning.pdf
21. Zhu, Wenbo & Yeh, Wei-Chang & Chen, Jianwen & Chen, Dafeng & Li, Aiyuan & Lin, Yangyang. (2019). Evolutionary Convolutional Neural Networks Using ABC. ICMLC '19: Proceedings of the 2019 11th International Conference on Machine Learning and Computing. 156-162. 10.1145/3318299.3318301.
22. Ayeni, Joshua. (2022). Convolutional Neural Network (CNN): The architecture and applications. Applied Journal of Physical Science. 4. 42-50. 10.31248/AJPS2022.085.
23. Vaz, Joel & Seetharaman, Balaji. (2021). Convolutional neural networks (CNNs): concepts and applications in pharmacogenomics. Molecular Diversity. 25. 10.1007/s11030-021-10225-3.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файл genetic_algorithm/activation_mapping.py:

```
activation_mapping  = {
    0: 'relu',
    1: 'sigmoid',
    2: 'tanh',
}
```

Файл genetic_algorithm/padding_mapping.py:

```
padding_mapping = {
    0: 'same',
    1: 'valid'
}
```

Файл genetic_algorithm/cnn_individual.py:

```
import random
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input,
Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.optimizers import Adam

from .activation_mapping import activation_mapping
from .padding_mapping import padding_mapping

class CNNIndividual:
    def __init__(
            self, num_conv_layers, num_filters, filter_size,
activation,
            pooling_size, dropout_prob, stride, padding,
```

```
learning_rate,
            beta_1, beta_2,
        ):
        self.num_conv_layers = num_conv_layers
        self.num_filters = num_filters
        self.filter_size = filter_size
        self.activation = activation
        self.pooling_size = pooling_size
        self.dropout_prob = dropout_prob
        self.stride = stride
        self.padding = padding
        self.learning_rate = learning_rate
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.encode_step = 0
        self.fitness = None
        self.model = None

    def __repr__(self):
        return
f"CNNIndividual(num_conv_layers={self.num_conv_layers},
num_filters={self.num_filters}, " \
               f"filter_size={self.filter_size},
activation={activation_mapping[self.activation]},
pooling_size={self.pooling_size}, " \
               f"dropout_prob={self.dropout_prob},
stride={self.stride}, padding={padding_mapping[self.padding]},
learning_rate={self.learning_rate}, " \
               f"beta_1={self.beta_1}, beta_2={self.beta_2})"
```

```
    def construct_model(self, x_train, y_train, x_val, y_val,
epochs=10, batch_size=32):
        model = Sequential()
        model.add(Input(shape=x_train.shape[1:]))

        for _ in range(self.num_conv_layers):
            model.add(Conv2D(self.num_filters,
self.filter_size,
activation=activation_mapping[self.activation],
strides=self.stride, padding=padding_mapping[self.padding]))

model.add(MaxPooling2D(pool_size=(self.pooling_size,
self.pooling_size)))
            if self.dropout_prob > 0:
                model.add(Dropout(self.dropout_prob))

        model.add(Flatten())
        model.add(Dense(y_train.shape[1],
activation='softmax'))

        optimizer = Adam(learning_rate=self.learning_rate,
beta_1=self.beta_1, beta_2=self.beta_2)
        model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

        model.fit(x_train, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(x_val, y_val),
```

```
verbose=0)
        self.model = model

        return model

    def evaluate_fitness(self, x_val, y_val):
        _, accuracy = self.model.evaluate(x_val, y_val,
verbose=0)

        self.fitness = accuracy

        return accuracy

    def encode(self, length=None):
        values = [
            self.num_conv_layers,
            self.num_filters,
            self.filter_size - 1,
            self.activation,
            self.pooling_size - 1,
            int(self.dropout_prob * 10),
            self.stride,
            self.padding,
            int(self.learning_rate * 1000),
            int(self.beta_1 * 1000),
            int(self.beta_2 * 1000),
        ]

        if length is None:
```

```
            max_len = max(len(hex(value)[2:]) for value in
values)
        else:
            max_len = length

        self.encode_step = max_len
        encoded_str = ''.join(hex(value)[2:].zfill(max_len)
for value in values)
        return encoded_str

    @classmethod
    def decode(cls, encoded: str, step: int = 2):
        if len(encoded) % step != 0:
            raise ValueError("Invalid length of the encoded
string")

        num_conv_layers = int(encoded[:step], 16)
        num_filters = int(encoded[step:2*step], 16)
        filter_size = int(encoded[2*step:3*step], 16) + 1
        activation = int(encoded[3*step:4*step], 16)
        pooling_size = int(encoded[4*step:5*step], 16) + 1
        dropout_prob = int(encoded[5*step:6*step], 16) / 10.0
        stride = int(encoded[6*step:7*step], 16)
        padding = int(encoded[7*step:8*step], 16)
        learning_rate = int(encoded[8*step:9*step], 16) /
1000.0
        beta_1 = int(encoded[9*step:10*step], 16) / 1000.0
        beta_2 = int(encoded[10*step:11*step], 16) / 1000.0
```

```
        return cls(num_conv_layers, num_filters, filter_size,
activation, pooling_size, dropout_prob, stride, padding,
learning_rate, beta_1, beta_2)

    # uniform mutation
    def mutate(self, mutation_rate=0.05):
        if mutation_rate < 0 or mutation_rate > 1:
            raise ValueError("Mutation rate must be between 0
and 1")

        if random.random() < mutation_rate:
            self.num_conv_layers = random.randint(1, 2)

        if random.random() < mutation_rate:
            self.num_filters = random.randint(16, 64)

        if random.random() < mutation_rate:
            self.filter_size = random.choice([3, 5])

        if random.random() < mutation_rate:
            self.activation = random.randint(0,
len(activation_mapping) - 1)

        if random.random() < mutation_rate:
            self.pooling_size = random.choice([2, 3])

        if random.random() < mutation_rate:
            dropout = random.uniform(-1, 1)
            if dropout > 0:
```

```
                self.dropout_prob = dropout
            else:
                self.dropout_prob = 0

        if random.random() < mutation_rate:
            self.stride = random.choice([1, 2])

        if random.random() < mutation_rate:
            self.padding = random.randint(0,
len(padding_mapping) - 1)

        if random.random() < mutation_rate:
            self.learning_rate = 10 ** (-random.uniform(1, 5))

        if random.random() < mutation_rate:
            self.beta_1 = random.uniform(0.8, 0.999)

        if random.random() < mutation_rate:
            self.beta_2 = 1 - 10 ** (-random.uniform(1, 5))
```

Файл genetic_algorithm/genetic_algorithm.py:

```
from threading import Thread
import random

from .cnn_individual import CNNIndividual
from .padding_mapping import padding_mapping
from .activation_mapping import activation_mapping

class GeneticAlgorithm:
    def __init__(self, population_size):
```

```
        self.population_size = population_size
        self.population = self.initialize_population()
        self.generation = 0
        self.best_individual_per_generation = []
        self.fitness_data = {
            'fitness_by_layer': [],
            'fitness_by_filter_num': [],
            'fitness_by_filter_size': [],
            'fitness_by_stride': [],
            'fitness_by_pooling_size': [],
            'fitness_by_dropout_prob': [],
            'fitness_by_beta_1': [],
            'fitness_by_beta_2': [],
            'fitness_by_learning_rate': [],
            'fitness_by_padding': [],
            'fitness_by_activation': []
        }


    def initialize_population(self):
        population = []
        for _ in range(self.population_size):
            num_conv_layers = random.randint(1, 2)
            num_filters = random.randint(16, 64)
            filter_size = random.choice([3, 5])
            activation = random.randint(0,
len(activation_mapping) - 1)
            pooling_size = random.choice([2, 3])
            dropout_prob =  random.uniform(-1, 1)
```

```
            beta_1 = random.uniform(0.8, 0.999)
            beta_2 = 1 - 10 ** (-random.uniform(1, 5))
            if dropout_prob < 0:
                dropout_prob = 0
            stride = random.choice([1, 2])
            padding = random.randint(0, len(padding_mapping) -
1)
            learning_rate = 10 ** (-random.uniform(1, 5))

            individual = CNNIndividual(
                num_conv_layers, num_filters,
filter_size,activation, pooling_size, dropout_prob,
                stride, padding, learning_rate, beta_1,
beta_2,
            )
            print(individual.__repr__())
            population.append(individual)

        return population

    # tournament selection
    def selection(self, tournament_size=3, p=0.75):
        selected_parents = []
        for _ in range(self.population_size):
            tournament = random.sample(self.population,
tournament_size)
            tournament.sort(key=lambda ind: ind.fitness,
reverse=True)
```

```
            for i in range(tournament_size):
                if random.random() < p * ((1 - p) ** i):
                    selected_parents.append(tournament[i])
                    break
            else:
                selected_parents.append(tournament[0])

        return selected_parents

    # one point crossover
    def crossover(self, parent1, parent2):
        parent1_encoded = parent1.encode()
        parent2_encoded = parent2.encode()

        step = max(parent1.encode_step, parent2.encode_step)
        if len(parent1_encoded) != len(parent2_encoded):
            parent1_encoded = parent1_encoded.extend(step)
            parent2_encoded = parent2_encoded.extend(step)

        crossover_point = random.randint(2,
(len(parent1_encoded) // step) - 1) * step

        child1_encoded = parent1_encoded[:crossover_point] +
parent2_encoded[crossover_point:]
        child2_encoded = parent2_encoded[:crossover_point] +
parent1_encoded[crossover_point:]

        child1 = CNNIndividual.decode(child1_encoded, step)
        child2 = CNNIndividual.decode(child2_encoded, step)
```

```
        return child1, child2

    def evaluate_individual(self, individual, x_val, y_val):
        individual.evaluate_fitness(x_val, y_val)
        fitness_categories = {
            'fitness_by_beta_1': 'beta_1',
            'fitness_by_beta_2': 'beta_2',
            'fitness_by_dropout_prob': 'dropout_prob',
            'fitness_by_learning_rate': 'learning_rate',
            'fitness_by_pooling_size': 'pooling_size',
            'fitness_by_stride': 'stride',
            'fitness_by_filter_size': 'filter_size',
            'fitness_by_filter_num': 'num_filters',
            'fitness_by_layer': 'num_conv_layers',
            'fitness_by_activation': 'activation',
            'fitness_by_padding': 'padding'
        }

        for category, attribute in fitness_categories.items():
            data = self.fitness_data[category]
            data.append({
                attribute: getattr(individual, attribute),
                'accuracy': individual.fitness
            })

    def evaluate_population_fitness(self, x_val, y_val):
        threads = []
        for individual in self.population:
```

```
            thread = Thread(target=self.evaluate_individual,
args=(individual, x_val, y_val))
            threads.append(thread)
            thread.start()
        for thread in threads:
            thread.join()

    def fit_population(self, population, x_train, y_train,
x_val, y_val, epochs, batch_size):
        threads = []
        for individual in population:
            thread = Thread(target=individual.construct_model,
args=(x_train, y_train, x_val, y_val, epochs, batch_size))
            threads.append(thread)
            thread.start()
        for thread in threads:
            thread.join()

    def populate(self, x_train, y_train, x_val, y_val,
epochs=10, batch_size=32, max_generations=30,
tournament_size=3, selection_p=0.75,elite_percentage=10):
        if elite_percentage > 100:
            raise ValueError("elite_percentage must be between
0 and 100")
        if batch_size < 1:
            raise ValueError("batch_size must be at least 1")
        if epochs < 1:
            raise ValueError("epochs must be at least 1")
        if max_generations < 1:
```

```
            raise ValueError("max_generations must be at least
1")

        self.fit_population(self.population, x_train, y_train,
x_val, y_val, epochs, batch_size)
        while self.generation + 1 < max_generations:
            self.evaluate_population_fitness(x_val, y_val)

            print(f"Generation {self.generation + 1} of
{max_generations}. Best fitness: {max(self.population,
key=lambda ind: ind.fitness).fitness}")

self.best_individual_per_generation.append(max(self.population
, key=lambda ind: ind.fitness))

            parents = self.selection(tournament_size,
p=selection_p)

            num_elite_parents = int(len(parents) *
elite_percentage / 100)
            elite_individuals = sorted(self.population,
key=lambda ind: ind.fitness, reverse=True)[:num_elite_parents]

            num_offspring_needed = self.population_size -
len(elite_individuals)
            num_offspring = num_offspring_needed +
num_offspring_needed % 2

            offspring = []
```

```
            for _ in range(num_offspring // 2):
                parent1, parent2 = random.sample(parents, 2)
                child1, child2 = self.crossover(parent1,
parent2)
                offspring.extend([child1, child2])

            offspring = offspring[:num_offspring_needed]

            for individual in offspring:
                individual.mutate()

            self.fit_population(offspring, x_train, y_train,
x_val, y_val, epochs, batch_size)

            self.population = offspring + elite_individuals
            self.generation += 1
```

Файл genetic_algorithm/__init__.py

Файл ./main.py

```
import os
import cv2
import numpy as np
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import random

from genetic_algorithm.genetic_algorithm import
GeneticAlgorithm
from genetic_algorithm.activation_mapping import
activation_mapping
```

```
from genetic_algorithm.padding_mapping import padding_mapping

dataset_dir = "data/Brain-Tumor-Classification-DataSet-master"
training_dir = os.path.join(dataset_dir, "Training")
testing_dir = os.path.join(dataset_dir, "Testing")

def load_data(directory):
    images = []
    labels = []
    classes = []
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for file in os.listdir(label_dir):
            image_path = os.path.join(label_dir, file)
            image = cv2.imread(image_path)
            image = cv2.resize(image, (64, 64))
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = np.expand_dims(image, axis=-1)
            image = image.astype(np.float32) / 255.0
            images.append(image)
            labels.append(label)
        classes.append(label)

    label_dict = {class_name: i for i, class_name in
enumerate(classes)}
    labels_one_hot = [label_dict[label] for label in labels]
    labels_one_hot = to_categorical(labels_one_hot,
num_classes=len(classes))
```

```
    return np.array(images), labels_one_hot, classes

def plot_fitness_dot(fitness_data, title, parameter_name):
    plt.figure(figsize=(10, 6))

    if parameter_name == 'activation':
        values =
[activation_mapping[data_point[parameter_name]] for data_point
in fitness_data]
    elif parameter_name == 'padding':
        values = [padding_mapping[data_point[parameter_name]]
for data_point in fitness_data]
    else:
        values = [data_point[parameter_name] for data_point in
fitness_data]


    accuracies = [data_point['accuracy'] for data_point in
fitness_data]

    plt.plot(values, accuracies, 'o', label=parameter_name)
    plt.title(title)
    plt.xlabel(parameter_name)
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.show()

def show_image(image, label):
    plt.imshow(image.squeeze(), cmap='gray')
```

```
    plt.title(f"Label: {label}")
    plt.axis('off')
    plt.show()

def plot_generations_accuracy(accuracy, title):
    plt.figure(figsize=(8, 6))
    generation_numbers = list(range(1, len(accuracy) + 1))
    plt.plot(generation_numbers, accuracy, marker='o')
    plt.title(title)
    plt.xlabel('Generation')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.show()

def main():
    genetic_algorithm = GeneticAlgorithm(population_size=10)

    x_train, y_train, classes = load_data(training_dir)
    x_val, y_val, _ = load_data(testing_dir)
    epochs = 10
    batch_size = 32

    print(f"Number of training examples: {len(x_train)}")
    print(f"Number of validation examples: {len(x_val)}")

    random_index = random.randint(0, len(x_train) - 1)
    random_image = x_train[random_index]
    random_label = classes[np.argmax(y_train[random_index])]
    show_image(random_image, random_label)
```

```
    genetic_algorithm.populate(x_train, y_train, x_val, y_val,
epochs, batch_size, max_generations=30)

    best_individual_per_generation =
genetic_algorithm.best_individual_per_generation

    plot_generations_accuracy([individual.fitness for
individual in best_individual_per_generation], "Best
Individual Accuracy per Generation")

    print(f"Best Individual:
{max(best_individual_per_generation, key=lambda x:
x.fitness).__repr__()}")

    fitness_data = {
        "Accuracy by Beta 1":
genetic_algorithm.fitness_data["fitness_by_beta_1"],
        "Accuracy by Beta 2":
genetic_algorithm.fitness_data["fitness_by_beta_2"],
        "Accuracy by Dropout Probability":
genetic_algorithm.fitness_data["fitness_by_dropout_prob"],
        "Accuracy by Learning Rate":
genetic_algorithm.fitness_data["fitness_by_learning_rate"],
        "Accuracy by Pooling Size":
genetic_algorithm.fitness_data["fitness_by_pooling_size"],
        "Accuracy by Stride":
genetic_algorithm.fitness_data["fitness_by_stride"],
        "Accuracy by Filter Size":
```

```
genetic_algorithm.fitness_data["fitness_by_filter_size"],
        "Accuracy by Number of Filters":
genetic_algorithm.fitness_data["fitness_by_filter_num"],
        "Accuracy by Number of Convolutional Layers":
genetic_algorithm.fitness_data["fitness_by_layer"],
        "Accuracy by Activation Function":
genetic_algorithm.fitness_data["fitness_by_activation"],
        "Accuracy by Padding":
genetic_algorithm.fitness_data["fitness_by_padding"]
    }

    for title, data in fitness_data.items():
        parameter_name = next(iter(data[0]))
        plot_fitness_dot(data, title, parameter_name)

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