**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА**

**Навчально-науковий інститут фізико-технічних та комп’ютерних наук**

**Кафедра комп’ютерних наук**

**Інформаційна технологія оптимізації ресурсоємності класифікаційних моделей на основі багатоетапної редукції зображень з конфігурованими параметрами**

**Кваліфікаційна робота**

**Рівень вищої освіти – другий (магістерський)**

***Виконав***:

Студент 2 курсу, 644 групи

**Максимович Микола Юрійович**

***Керівник***:

к.б.н., асистент **Талах М.В.**

***До захисту допущено***

***На засіданні кафедри***

***Протокол № 6 від 11.11.2024 р.***

***Зав. кафедрою проф. Ушенко Ю.О.***

**Чернівці – 2024**

**Міністерство освіти і науки України**

**Чернівецький національний університет імені Юрія Федьковича**

**Навчально-науковий інститут фізико-технічних та комп’ютерних наук**

**Кафедра комп’ютерних наук**

Рівень вищої освіти: магістр

Освітньо-професійна програма: Інтелектуальний аналіз даних в комп’ютерних інформаційних системах

Спеціальність: № 122 Комп’ютерні науки

**ЗАТВЕРДЖУЮ**

Завідувач кафедри комп’ютерних наук

Ю.О. Ушенко « » 2024 року

**ЗАВДАННЯ**

**НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ**

|  |
| --- |
| Максимовичу Миколі Юрійовичу |

*(прізвище, ім’я, по батькові)*

1. Тема роботи «**Інформаційна технологія оптимізації ресурсоємності класифікаційних моделей на основі багатоетапної редукції зображень з конфігурованими параметрами**»

керівник роботи: Марія Талах, к.б.н., асистент

затверджена протоколом засіданні кафедри від 30 серпня 2024 року № 2

2. Строк подання студентом закінченої роботи 11.11.2024

3. Вхідні дані до роботи: набір готових датасетів: keras.datasets.mnist, keras.datasets.fashion\_mnist, keras.datasets.cifar10.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які треба розробити):

РОЗДІЛ 1. ОГЛЯД ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ ОБ’ЄКТА ДОСЛІДЖЕНН

* Редукція зображень у контексті оптимізації машинного навчанн
* Огляд аналогів системи
* Загальна ідея та необхідність додатку
* Висновки до розділу

РОЗДІЛ 2. ТЕОРЕТИЧНІ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

* Бізнес-ризики
* Теоретичне обґрунтування методів редукції зображень для впровадження в систему
* Обгрунтування вибору алгоритмів класифікації для оцінки оптимального ступеня редукції
* Експериментальні дослідження
* Висновки до розділу

РОЗДІЛ 3. ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ

* Моделювання предметної області
* Проектування логічної структури системи
* Інженерія даних
* Впровадження моделей машинного навчання
* Результати та оцінка
* Висновки до розділу

РОЗДІЛ 4. АПРОБАЦІЯ РОБОТИ СИСТЕМИ

* Апробація системи
* Інструкція для користувача
* Інструкція для програміста
* Висновки до розділу

5. Перелік графічного, наочного матеріалу:

1. Діаграма потоків даних системи
2. Діаграма взаємодії системи
3. Діаграма класів
4. Діаграма пакетів
5. Діаграма модулів
6. Компонентна діаграма
7. Діаграма послідовності
8. Діаграма станів
9. Графіки з відношенням точності, швидкодії до кількості латентних векторів(ступеня зжаття)

6. Консультант(и) дипломної роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| РОЗДІЛ 1. ОГЛЯД ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ ОБ’ЄКТА ДОСЛІДЖЕНН |  | 01.09.2024 | 01.09.2024 |
| РОЗДІЛ 2. ТЕОРЕТИЧНІ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ | Дворжак В.В., к. т. н., асистент | 12.09.2024 | 12.09.2024 |
| РОЗДІЛ 3. ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ | Томка Ю.Я., к. ф.-м. н., асистент | 22.09.2024 | 22.09.2024 |

7. Дата видачі завдання: 30 серпня 2024 р

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів дипломної роботи | Термін виконання | Примітка |
|  | Вибір, погодження й затвердження теми, призначення наукового керівника, рецензента, консультанта | 01.09.2024 | виконано |
|  | Складання календарного плану й розгорнутого плану-конспекту роботи. Опрацювання джерел | 03.09.2024 | виконано |
|  | Аналіз предметної області, дослідження літератури та матеріалів | 12.09.2024 | виконано |
|  | Аналіз існуючих аналогів програмного забезпечення | 19.09.2024 | виконано |
|  | Постановка задачі за темою дипломної роботи | 26.09.2024 | виконано |
|  | Організація і проведення теоретичного, емпіричного (експериментального) дослідження | 2.10.2024 | виконано |
|  | Розробка основних складових частин додатку | 07.11.2024 | виконано |
|  | Оформлення пояснювальної записки | 16.11.2024 | виконано |
|  | Підготовка презентації та доповіді. | 19.11.2024 | виконано |
|  | Попередній захист. Обговорення роботи на кафедрі й рекомендації її до захисту. | 20.11.2024 | виконано |
|  | Коригування помилок, внесення виправлень, доопрацювання матеріалу. | 30.11.2024 | виконано |
|  | Подання роботи на рецензування. | 01.12.2024 | виконано |
|  | Захист дипломної роботи | 04.12.2024 | виконано |

Студент \_\_\_\_\_\_\_\_\_ Максимович М.Ю.

Керівник роботи \_\_\_\_\_\_\_\_\_\_ Талах М.В.

**АНОТАЦІЯ**

Максимович Микола Юрійович. «Інформаційна технологія оптимізації ресурсоємності класифікаційних моделей на основі багатоетапної редукції зображень з конфігурованими параметрами». Кваліфікаційна робота освітнього рівня – магістр, на правах рукопису. Спеціальність – 122 Комп’ютерні науки (Інтелектуальний аналіз даних в комп’ютерних інформаційних системах). – Чернівці, 2024.

У кваліфікаційній роботі проаналізовано підходи до редукції зображень у контексті оптимізації моделей машинного навчання, розглянуто аналоги системи для визначення її конкурентних переваг. Охарактеризовано основні бізнес-ризики, що виникають під час впровадження подібних систем. Доведено доцільність використання вибраних методів редукції зображень та алгоритмів класифікації для оцінки оптимального ступеня редукції.

В ході роботи досліджено експериментальні результати, описано процес проектування логічної структури системи та моделювання предметної області. Розроблено систему з використанням сучасних технологій обробки даних та моделей машинного навчання. Проведено апробацію системи, представлено інструкції для користувача та програміста.

**Ключові слова**: РЕДУКЦІЯ ЗОБРАЖЕНЬ, МАШИННЕ НАВЧАННЯ, ОПТИМІЗАЦІЯ, ІНЖЕНЕРІЯ ДАНИХ, АЛГОРИТМИ РЕДУКЦІЇ, SVD, АВТОЕНКОДЕРИ, КОНВЕЄР, ПАЙПЛАЙН ОБРОБКИ ДАНИХ.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

\_\_\_\_\_\_\_\_ М.Ю. Максимович

(підпис)

**ABSTRACT**

Maksymovych Mykola Yuriiovych. «Information Technology for Classification Model Resource Optimization Based on Multi-stage Image Reduction with Configurable Parameters» Master's Thesis, manuscript format. Specialty – 122 Computer Science (Intelligent Data Analysis in Computer Information Systems). – Chernivtsi, 2024.

In the qualification work, approaches to image reduction in the context of optimizing machine learning models are analyzed, and system analogs are examined to determine its competitive advantages. Key business risks associated with the implementation of such systems are characterized. The feasibility of using selected image reduction methods and classification algorithms for assessing the optimal degree of reduction is substantiated.

Experimental results are investigated, and the process of designing the system's logical structure and modeling the subject area is described. A system has been developed using modern data processing technologies and machine learning models. System testing has been conducted, and user and programmer instructions are provided.

**Keywords**: IMAGE REDUCTION, MACHINE LEARNING, OPTIMIZATION, DATA ENGINEERING, REDUCTION ALGORITHMS, SVD, AUTOENCODERS, PIPELINE, DATA PROCESSING PIPELINE.

The thesis contains the results of the author's own research. The use of ideas, findings, and texts from the scientific research of other authors is duly referenced to the appropriate source.

\_\_\_\_\_\_\_\_\_ M.Y. Maksymovych

(signature)

**ЗМІСТ**

[ВСТУП 5](#_Toc183985860)

[РОЗДІЛ 1. ОГЛЯД ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ ОБ’ЄКТА ДОСЛІДЖЕННЯ 8](#_Toc183985861)

[1.1 Редукція зображень у контексті оптимізації машинного навчання 8](#_Toc183985862)

[1.2 Огляд аналогів системи 12](#_Toc183985863)

[1.2.1 Аналоги для роботи з зображеннями 13](#_Toc183985864)

[1.2.2 Аналоги для конструювання пайплайнів 14](#_Toc183985865)

[1.3 Загальна ідея та необхідність додатку 15](#_Toc183985866)

[1.3.1 Мета та завдання розробки 15](#_Toc183985867)

[1.3.2 Основні функції додатку 16](#_Toc183985868)

[1.3.3 Вибір методів редукції даних 17](#_Toc183985869)

[1.3.4 Користувацький інтерфейс 17](#_Toc183985870)

[1.3.5 Нефункціональні вимоги 17](#_Toc183985871)

[1.3.6 Надійність та зручність 17](#_Toc183985872)

[1.3.7 Можливості розширення системи 18](#_Toc183985873)

[Висновки до розділу 18](#_Toc183985874)

[РОЗДІЛ 2. ТЕОРЕТИЧНІ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ 19](#_Toc183985875)

[2.1 Бізнес-ризики 19](#_Toc183985876)

[2.2 Теоретичне обґрунтування методів редукції зображень для впровадження в систему 20](#_Toc183985877)

[2.2.1 Singular Value Decomposition 20](#_Toc183985878)

[2.2.2 Principal Component Analysis 22](#_Toc183985879)

[2.2.3 Autoencoders 25](#_Toc183985880)

[2.2.4 Fast Independent Component Analysis 27](#_Toc183985881)

[2.2.5 t-Distributed Stochastic Neighbor Embedding 28](#_Toc183985882)

[2.2.6 Non-negative Matrix Factorization 29](#_Toc183985883)

[2.2.7 Підсумки обгрунтування вибору методів редукції 30](#_Toc183985884)

[2.3 Обгрунтування вибору алгоритмів класифікації для оцінки оптимального ступеня редукції 32](#_Toc183985885)

[2.3.1 Multi-Layer Perceptron Classifier 32](#_Toc183985886)

[2.3.2 Support Vector Classifier 33](#_Toc183985887)

[2.3.3 Decision Tree Classifier 34](#_Toc183985888)

[2.3.4 Random Forest Classifier 35](#_Toc183985889)

[2.3.5 Підсумки обгрунтування вибору алгоритмів класифікації 37](#_Toc183985890)

[2.4 Експериментальні дослідження 38](#_Toc183985891)

[2.4.1 Розробка кастомної моделі адаптивного автоенкодеру 38](#_Toc183985892)

[2.4.2 Експерименти з підбором оптимальних параметрів для окремих алгоритмів редукції 41](#_Toc183985893)

[2.4.3 Експерименти з підбором оптимальної архітектури конвеєра в залежності від характеристик зображень 46](#_Toc183985894)

[Висновки до розділу 48](#_Toc183985895)

[РОЗДІЛ 3. ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ 49](#_Toc183985896)

[3.1 Моделювання предметної області 49](#_Toc183985897)

[3.1.1 Архітектурний дизайн 52](#_Toc183985898)

[3.1.2 Функціональні компоненти та процеси системи 54](#_Toc183985899)

[3.2. Проектування логічної структури системи 56](#_Toc183985900)

[3.2.1 Загальні принципи проектування та декомпозиція системи 56](#_Toc183985901)

[3.2.2 Організація шару даних та попередньої обробки 60](#_Toc183985902)

[3.2.3 Ядро системи та конвеєр обробки 60](#_Toc183985903)

[3.2.4 Система оцінювання та експорту результатів 61](#_Toc183985904)

[3.2.5 Взаємодія компонентів та потоки даних 62](#_Toc183985905)

[3.2.6 Моделювання поведінки системи 62](#_Toc183985906)

[3.3 Інженерія даних 65](#_Toc183985907)

[3.3.1 Екстракція даних (Extract) 65](#_Toc183985908)

[3.3.2 Трансформація даних (Transform) 68](#_Toc183985909)

[3.3.3 Завантаження та збереження (Load) 70](#_Toc183985910)

[3.4 Впровадження моделей машинного навчання 72](#_Toc183985911)

[3.4.1 Визначення мір якості моделі 72](#_Toc183985912)

[3.4.2 Моделі редукції даних 73](#_Toc183985913)

[3.4.3 Моделі оцінки редукції 74](#_Toc183985914)

[3.4.4 Оптимізація алгоритму ML (підбір оптимальних параметрів) 75](#_Toc183985915)

[3.4.5 Навчання моделей 76](#_Toc183985916)

[3.5 Результати та оцінка 77](#_Toc183985917)

[Висновки до розділу 79](#_Toc183985918)

[РОЗДІЛ 4. АПРОБАЦІЯ РОБОТИ СИСТЕМИ 80](#_Toc183985919)

[4.1 Апробація системи 80](#_Toc183985920)

[4.1.1 Структура інтерфейсу та функціональні модулі 80](#_Toc183985921)

[4.1.2 Процедура апробації системи 81](#_Toc183985922)

[4.1.3 Розширення функціональності системи 81](#_Toc183985923)

[4.1.3.1. Інтеграція користувацьких моделей класифікації 81](#_Toc183985924)

[4.1.3.2. Впровадження нових методів редукції 82](#_Toc183985925)

[4.1.4 Результати апробації системи 82](#_Toc183985926)

[4.1.4.1 Завантаження та попередня обробка даних 82](#_Toc183985927)

[4.1.4.2. Конфігурація та виконання редукції 83](#_Toc183985928)

[4.1.4.3. Оптимізація параметрів та візуалізація 84](#_Toc183985929)

[4.1.5 Фінальна обробка та експорт результатів 85](#_Toc183985930)

[4.1.6 Тестування розширюваності системи 87](#_Toc183985931)

[4.2 Інструкція для користувача 88](#_Toc183985932)

[4.3 Інструкція для програміста 93](#_Toc183985933)

[Висновки до розділу 95](#_Toc183985934)

[ВИСНОВКИ 96](#_Toc183985935)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 98](#_Toc183985936)

[ДОДАТКИ 101](#_Toc183985937)

[ДОДАТОК А 101](#_Toc183985938)

# ВСТУП

**Актуальність роботи.** На сьогодні, обробка та аналіз великих обсягів зображень стали надзвичайно актуальними у багатьох галузях, від медичної діагностики до супутникового моніторингу. Зображення, порівняно з іншими типами даних, займають великий обсяг пам'яті та потребують значних обчислювальних ресурсів, що ускладнює обробку навіть на потужних комп'ютерах, а для персональних обчислювальних пристроїв є майже недосяжним завданням. У сучасному світі розвиток комп'ютерного зору, машинного навчання та нейронних мереж стимулює запит на оптимізацію методів роботи з великими обсягами зображень.

Тема даної роботи є актуальною, оскільки редукція зображень може значно зменшити необхідний обсяг даних без втрати критичної інформації, що спрощує їх обробку та аналіз. Це дозволяє ефективно використовувати обчислювальні ресурси, роблячи аналіз великих даних доступнішим, особливо для користувачів з обмеженими технічними можливостями.

**Мета роботи** – розробка інформаційної технології багатоетапної редукції зображень з конфігурованими параметрами для оптимізації ресурсоємності класифікаційних моделей машинного навчання при збереженні прийнятної точності класифікації.

Для досягнення цієї мети визначені наступні **завдання**:

1. Провести огляд та аналіз існуючих методів зниження розмірності зображень.
2. Розробити архітектуру додатка для редукції зображень, що включає інтеграцію з кінцевими алгоритмами класифікації
3. Спроектувати та реалізувати програмні компоненти для інтеграції методів редукції зображень та оцінки їх впливу на ефективність класифікаційних моделей.
4. Провести тестування додатка та оцінити його ефективність у різних задачах зменшення обсягу даних.

**Об'єктом дослідження** є процес оптимізації ресурсоємності класифікаційних моделей машинного навчання шляхом попередньої обробки вхідних зображень.

**Предмет дослідження** – моделі, методи та інформаційні технології конфігурованої багатоетапної редукції зображень для оптимізації обчислювальних ресурсів класифікаційних моделей машинного навчання.

**Методи дослідження.** Отримані результати дослідження базуються на використанні методів зниження розмірності даних (PCA, SVD, автоенкодери), методів машинного навчання для задач класифікації, методів оптимізації та статистичного аналізу для оцінки якості редукції зображень.

**Наукова новизна роботи**. Удосконалено метод оптимізації класифікаційних моделей, який, на відміну від існуючих, забезпечує інтеграцію різних алгоритмів редукції зображень (PCA, SVD, автоенкодери) в єдиний конфігурований користувачем конвеєр обробки, що дозволяє знизити обчислювальні витрати зі збереженням прийнятної точності класифікації. Також удосконалено метод багатоетапної редукції зображень, який, на відміну від існуючих, адаптується під потреби користувача через можливість вибору послідовності та налаштування параметрів алгоритмів редукції, що забезпечує гнучке балансування між ступенем стиснення даних та якістю збереження інформативних ознак.

**Практична значущість** отриманих результатів полягає в розробці інформаційної технології, що дозволяє оптимізувати ресурсоємність класифікаційних моделей машинного навчання при збереженні прийнятної точності класифікації. Розроблена технологія може бути застосована як у навчальних цілях для вивчення алгоритмів зниження розмірності, так і в системах комп'ютерного зору, медичної діагностики, аналізу супутникових знімків та інших галузях, де критичними є вимоги до обчислювальних ресурсів при роботі з великими наборами зображень. Впровадження запропонованої технології дозволяє значно знизити вимоги до обчислювальних ресурсів при збереженні якості класифікації, що підтверджено результатами експериментальних досліджень. Розроблене рішення особливо ефективне в проєктах, де швидкість обробки та оптимізація ресурсів при роботі з великими наборами зображень мають критичне значення.

**Публікації та апробація.** За темою дипломної роботи опубліковано статтю у фаховому виданні категорії Б та здійснено апробацію результатів дослідження на міжнародній конференції:

В публікаціях досліджено питання балансування між ефективністю обробки та збереженням точності при роботі з великими наборами даних, зокрема через впровадження поступового навчання для оптимізації ресурсоємності класифікаційних моделей. Окрему увагу приділено оцінці компромісів між ступенем стиснення даних та збереженням інформативних ознак у задачах класифікації.

Результати роботи представлено на VIII Міжнародній конференції з штучного інтелекту, медичної інженерії та освіти (AIMEE2024, Хуанши, Китай) та опубліковано у фаховому журналі "Оптико-електронні інформаційно-енергетичні технології" [26].

Всі роботи опубліковано у співавторстві.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (26 найменуваннь), 1 додаток. Загальний обсяг роботи становить 116 сторінок основного тексту, 43 рисунки та 1 таблиці.

# РОЗДІЛ 1. ОГЛЯД ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ ОБ’ЄКТА ДОСЛІДЖЕННЯ

## 1.1 Редукція зображень у контексті оптимізації машинного навчання

Редукція даних є фундаментальним процесом в галузі обробки та аналізу інформації, який полягає у трансформації вхідних даних до представлення меншої розмірності при збереженні їх ключових характеристик та інформативних ознак [23]. В контексті роботи з зображеннями редукція набуває особливого значення, оскільки візуальні дані характеризуються надзвичайно високою розмірністю та надлишковістю інформації. Як показано в дослідженні [20], навіть незначне зменшення розмірності може призвести до суттєвої економії обчислювальних ресурсів без значної втрати якості класифікації – у деяких випадках досягається зменшення обсягу даних на 70-80% при збереженні точності класифікації на рівні 95-97% від початкової.

Сутність редукції зображень полягає у пошуку такого представлення візуальних даних, яке б дозволило суттєво зменшити їх обсяг при збереженні інформативних характеристик, критичних для подальшої обробки, зокрема, для задач класифікації [16]. За результатами досліджень [1], навіть одне цифрове зображення середньої якості (наприклад, розміром 1000х1000 пікселів у форматі RGB) містить 3 мільйони значень, з яких, згідно з експериментальними даними, до 40-60% інформації може бути надлишковою для конкретних задач класифікації.

Основна ідея редукції базується на декількох ключових спостереженнях, підтверджених експериментальними дослідженнями [11]:

1. Більшість реальних зображень містить значну кількість надлишкової інформації - дослідження показують, що до 60-70% даних можуть бути надлишковими для конкретних задач класифікації;
2. Сусідні пікселі часто мають високу кореляцію - виявлено, що коефіцієнт кореляції може досягати 0.95 для сусідніх пікселів у природних зображеннях;
3. Не всі компоненти зображення однаково важливі для задачі класифікації - експерименти демонструють, що часто лише 20-30% ознак забезпечують 80-90% точності класифікації;
4. Існують ефективні математичні методи для виділення найбільш інформативних ознак.

За даними досліджень [20], необхідність застосування редукції зображень у задачах машинного навчання обумовлена низкою критичних факторів, що впливають на ефективність роботи класифікаційних моделей:

1. Обмеженість обчислювальних ресурсів:

* Оперативна пам'ять: експерименти показують, що для навчання моделей на наборах з 100,000 зображень високої якості може знадобитися 32-64 ГБ RAM
* Процесорний час: обробка повнорозмірних зображень може збільшувати час навчання в 3-5 разів
* Графічні прискорювачі: навіть сучасні GPU з 8-12 ГБ відеопам'яті можуть обмежувати розмір batch'у до неефективно малих значень

1. Оптимізація процесу навчання: Згідно з дослідженнями [7], ефективна редукція даних може суттєво покращити процес навчання:

* Зменшення часу навчання моделей: експерименти показують скорочення часу в 2.5-4 рази
* Можливість обробки більших наборів даних: редукція дозволяє збільшити розмір навчальної вибірки на 40-60%
* Зниження ризику перенавчання: дослідження демонструють зменшення розриву між навчальною та валідаційною точністю на 15-20%

1. Економічні аспекти: [16] провели детальний аналіз економічної ефективності редукції зображень:

* Зменшення вимог до апаратного забезпечення може знизити вартість обладнання на 30-50%
* Економія енергоспоживання досягає 40-60% при збереженні якості результатів
* Оптимізація використання хмарних ресурсів зменшує експлуатаційні витрати на 25-35%

У контексті машинного навчання редукція зображень набуває додаткового значення через специфіку роботи класифікаційних моделей. За даними досліджень [11], надмірна деталізація вхідних даних не тільки не покращує якість класифікації, але може навіть погіршувати результати через появу шуму в даних. Експерименти показують, що оптимальна розмірність вхідних даних часто становить 30-40% від початкової при збереженні 95-98% точності класифікації.

Згідно з дослідженням [23], ефективність редукції зображень можна оцінювати за наступними критеріями:

1. Ступінь зменшення розмірності даних (типово досягається компресія 60-80%)
2. Збереження інформативних ознак (ключові характеристики зберігаються на рівні 90-95%)
3. Вплив на точність класифікації (допустиме зниження не більше 2-3%)
4. Обчислювальна складність процесу редукції (має бути менша за виграш від зменшення даних)
5. Можливість відновлення даних, якщо необхідно (точність реконструкції 85-90%)

Сучасні дослідження [7] демонструють, що комбінування різних методів редукції може дати синергетичний ефект. Наприклад, послідовне застосування PCA та автоенкодерів дозволяє досягти кращих результатів, ніж використання кожного методу окремо:

* Підвищення ступеня стиснення на 15-20%
* Збереження точності класифікації на рівні 97-98%
* Зменшення обчислювальної складності на 40-50%

Згідно з дослідженням [20], для зображень найбільш ефективними виявляються наступні методи редукції:

1. Principal Component Analysis (PCA): ефективний завдяки здатності зберігати до 90% варіативності даних при зменшенні розмірності на 70%, лінійній складності алгоритму та можливості відновлення даних з мінімальними втратами.
2. Автоенкодери для зображень: дозволяють зберігати просторову структуру, досягати коефіцієнту стиснення до 85% та адаптивно навчати найважливіші ознаки.
3. Singular Value Decomposition (SVD): оптимальне для матричних даних, дає можливість точного контролю ступеня стиснення та вибору важливих компонент.
4. Wavelet-перетворення: ефективно зберігає локальні особливості, дозволяє багаторівневу декомпозицію та добре працює з природними зображеннями.

Особливий інтерес представляють гібридні підходи, наприклад, комбінування PCA + Wavelets для збереження глобальних та локальних особливостей, або Автоенкодери + SVD для покращення якості стиснення на 10-15%.

За дослідженнями [19], ефективність застосування методів редукції суттєво залежить від специфіки вхідних даних та цільової задачі:

1. Вплив характеристик вхідних зображень:

* Для природних зображень Wavelet-перетворення показують найкращі результати (стиснення до 80%). Автоенкодери ефективні для подібних текстур. А PCA добре працює з однотипними об'єктами
* Для медичних зображень SVD забезпечує найкраще збереження діагностично важливих деталей. Гібридні методи дозволяють зберегти до 98% значущої інформації. Критична важливість контролю втрат при стисненні

1. Обчислювальні аспекти: За даними [11], практичні обмеження методів:

Таблиця 1.1 - Порівняння ресурсоємності методів редукції зображень

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **RAM (1000x1000 RGB)** | **Час обробки** | **GPU-прискорення** |
| PCA | 2-3 ГБ | 1-2 с | 2-3x |
| Автоенкодери | 4-6 ГБ | 10-15 с | 8-10x |
| SVD | 3-4 ГБ | 3-5 с | 4-5x |
| Wavelets | 1-2 ГБ | <1 с | 1.5-2x |

Згідно з [1], проблема навчання моделей на звичайному комп'ютері загострюється через:

1. Обмеженість обчислювальних ресурсів:

* Оперативна пам'ять: для навчання на 100,000 зображень може знадобитися 32-64 ГБ RAM
* Процесорний час: обробка повнорозмірних зображень збільшує час навчання в 3-5 разів
* Графічні прискорювачі: GPU з 8-12 ГБ можуть обмежувати розмір batch'у

1. Обмеження при роботі з великими наборами даних:

* Можливі збої через нестачу пам'яті
* Тривалий час обробки через великі об'єми даних
* Висока вартість обчислювальних ресурсів для масштабування

Ключовим є врахування специфіки вхідних даних та задачі при виборі методу редукції для оптимального балансу стиснення, збереження інформації та обчислювальної складності.

## 1.2 Огляд аналогів системи

На сьогодні існує ряд інструментів, які дозволяють виконувати обробку та редукцію зображень. Деякі з них можуть бути частково застосовні для вирішення задач, подібних до тих, що ставить перед собою наш сервіс. Розглянемо найбільш релевантні аналоги, порівняємо їхні можливості та обмеження в контексті нашої системи.

### 1.2.1 Аналоги для роботи з зображеннями

OpenCV OpenCV - це потужна бібліотека для обробки зображень і відео, яка надає широкий спектр функцій для попередньої обробки візуальних даних. Згідно документації [17] вона дозволяє виконувати базову редукцію, такі як зміна роздільної здатності або стискання зображень. Перевагою OpenCV є те, що вона добре інтегрується з іншими бібліотеками машинного навчання та забезпечує швидку обробку зображень.

Однак, OpenCV не надає комплексного рішення для редукції великих обсягів зображень, особливо якщо потрібно застосувати алгоритми зниження розмірності, такі як PCA або автоенкодери. Ці можливості, які є критично важливими для нашої системи, в OpenCV відсутні.

TensorFlow та PyTorch - це популярні платформи для машинного навчання, які дозволяють використовувати нейронні мережі, зокрема автоенкодери, для виконання складної редукції даних. Ці інструменти можуть бути корисними для реалізації етапів зменшення розмірності зображень в межах нашого сервісу [18,21].

Проте, використання TensorFlow та PyTorch вимагає глибоких знань у галузі глибокого навчання та значних обчислювальних ресурсів, особливо для налаштування та навчання моделей автоенкодерів. Це може бути проблематичним для користувачів, які мають обмежений доступ до потужних комп'ютерів.

Онлайн-сервіси стискання зображень Існують також онлайн-сервіси, такі як TinyPNG або JPEG-Optimizer, які пропонують спрощені рішення для стискання зображень. Вони можуть бути корисними для базового зменшення розміру файлів, але не надають можливості для більш складної редукції розмірності або оптимізації для задач машинного навчання [12,22].

Ці сервіси мають обмежений функціонал і не дозволяють гнучкого налаштування параметрів редукції, що робить їх непридатними для нашої системи, яка має забезпечувати контрольовану редукцію зображень для подальшого застосування в моделях класифікації.

### 1.2.2 Аналоги для конструювання пайплайнів

FiftyOne - це платформа для роботи з наборами даних зображень, що дозволяє виконувати попередню обробку зображень і гнучку інтеграцію з моделями машинного навчання. Згідно [6] – FiftyOne підтримує створення кастомних конвеєрів для підготовки, фільтрації та візуалізації даних.

Перевагою FiftyOne є можливість створення пайплайнів обробки зображень, що може бути корисним для нашої системи. Однак, FiftyOne акцентує увагу на анотуванні та підготовці даних для задач розпізнавання, а не на спеціальній редукції зображень для класифікації, що є ключовою потребою нашого сервісу [6].

MLflow - це інструмент для управління експериментами машинного навчання, який також можна використовувати для налаштування конвеєрів обробки даних, включаючи етапи редукції зображень. Згідно [15] MLflow пропонує простий інтерфейс для налаштування та відслідковування різних стадій обробки.

Проте MLflow не має спеціалізованої інтеграції або функціоналу, орієнтованого саме на редукцію зображень для їх подальшого використання у класифікаційних моделях [15]. Це обмежує його застосовність для наших потреб.

DVC (Data Version Control) - це інструмент для керування даними в ML-проєктах, який може підтримувати складні конвеєри для підготовки зображень, включаючи етапи редукції. Згідно [5] – DVC дозволяє відслідковувати різні версії наборів даних і конфігурацій обробки.

Перевагою DVC є можливість побудови повноцінних пайплайнів із редукцією даних, що може бути корисним для нашого сервісу [5]. Однак, для реалізації конкретних етапів редукції, орієнтованих на класифікаційні задачі, знадобиться додаткова інтеграція з бібліотеками на кшталт PyTorch або TensorFlow.

ClearML - це інструмент для управління експериментами, який може допомогти створити кастомні пайплайни з обробки даних та редукції зображень для задач машинного навчання. ClearML дозволяє будувати кастомні пайплайни для обробки та оптимізації даних, а також підтримує інтеграцію з TensorFlow та PyTorch для тренування моделей [4].

Перевагою ClearML є можливість створення гнучких пайплайнів, включаючи етапи редукції зображень, що може бути корисним для реалізації нашого сервісу [4]. Проте, як і в інших подібних інструментах, для повноцінного вирішення наших задач знадобиться додаткова інтеграція та налаштування.

Висновки: розглянуті аналоги демонструють, що наразі не існує єдиного рішення, яке б повністю відповідало вимогам нашої системи. Існуючі інструменти або надають лише базові можливості обробки зображень, або спрямовані на більш загальні задачі машинного навчання без спеціалізації на редукції даних для класифікації.

Ключовою відмінністю нашого сервісу має стати поєднання гнучкості конструювання пайплайнів обробки зображень, орієнтованих на задачі класифікації, з можливостями застосування різноманітних методів редукції даних, доступних навіть для користувачів з обмеженими обчислювальними ресурсами. Це дозволить створити унікальне рішення, що значно полегшить підготовку даних для задач комп'ютерного зору.

## 1.3 Загальна ідея та необхідність додатку

### 1.3.1 Мета та завдання розробки

Ключовою відмінністю нашого сервісу має стати поєднання гнучкості конструювання пайплайнів обробки зображень, орієнтованих на задачі класифікації, з можливостями застосування різноманітних методів редукції даних, доступних навіть для користувачів з обмеженими обчислювальними ресурсами. Це дозволить створити унікальне рішення, що значно полегшить підготовку даних для задач комп'ютерного зору.

На відміну від розглянутих аналогів, наш сервіс забезпечуватиме комплексне та гнучке управління конвеєром обробки зображень, з фокусом на ефективну редукцію даних для подальшого застосування в моделях класифікації. Це робить нашу систему перспективним інструментом для широкого кола користувачів, зацікавлених у вирішенні задач комп'ютерного зору.

Таким чином, метою роботи є створення гнучкої системи для конструювання конвеєрів обробки зображень з фокусом на редукцію даних для задач класифікації. Ключова особливість - доступність для користувачів з обмеженими обчислювальними ресурсами та можливість розширення функціоналу.

### 1.3.2 Основні функції додатку

1. Завантаження наборів зображень:

* Вибір з готових наборів даних
* Підтримка основних форматів зображень (JPG, PNG, TIFF, тощо)
* Можливість завантаження зображень з локальної файлової системи.
* Інтуїтивно зрозумілий користувацький інтерфейс для імпорту даних

2. Конструювання пайплайну обробки зображень

* Вибір кількості етапів редукції (1-3)
* Налаштування параметрів для кожного етапу
* Вибір методів класифікації для оцінки якості
* Представлення етапів обробки в зручному для користувача вигляді
* Можливість вибору та налаштування методів редукції даних на кожному етапі

3. Застосування методів редукції даних

* Підтримка широкого переліку алгоритмів зниження розмірності, наприклад, PCA та автоенкодери
* Гнучке налаштування параметрів редукції для оптимального балансу між якістю та розміром даних - встановлення мінімального та максимального ступеня стиснення, а також кроку зміни параметрів

4. Збереження результатів редукції

* Графічне представлення результатів підбору параметрів
* Відображення метрик якості редукції
* Експорт редукованих даних у CSV форматі

### 1.3.3 Вибір методів редукції даних

Застосування алгоритмів зниження розмірності охоплює кілька важливих аспектів. Спершу, використовуються методи лінійного зниження розмірності, наприклад, PCA. Також впроваджуються нелінійні підходи, засновані на автоенкодерах. Користувач має можливість налаштовувати архітектуру та гіперпараметри моделей редукції, що підвищує їх ефективність.

### 1.3.4 Користувацький інтерфейс

Особливості користувацького інтерфейсу зосереджені на забезпеченні зручності для користувача. Однією з ключових характеристик є інтерактивне представлення етапів обробки, яке дозволяє додавати, видаляти або змінювати їх. Наочне представлення результатів редукції включає як статистичну інформацію про ефективність редукції, так і візуалізацію метрик для алгоритмів зниження розмірності. Крім того, інтеграція з іншими компонентами забезпечується шляхом експорту оброблених даних у форматах, сумісних із популярними бібліотеками машинного навчання.

### 1.3.5 Нефункціональні вимоги

Нефункціональні вимоги охоплюють різні аспекти роботи системи. Щодо продуктивності, здійснюється оптимізація обчислювальних ресурсів, ефективне використання CPU та RAM для мінімізації навантаження на систему. Додатково забезпечується можливість паралельної обробки зображень, що сприяє підвищенню швидкодії. Швидкість обробки зображень досягається шляхом оптимізації алгоритмів, що дозволяє зменшити час обробки великих наборів даних навіть на комп'ютерах із обмеженими ресурсами.

### 1.3.6 Надійність та зручність

Вимоги до надійності передбачають збереження результатів редукції, що включає можливість безперервного збереження проміжних і кінцевих результатів обробки, а також відновлення роботи додатку після збоїв чи переривань без втрати даних. Для забезпечення масштабованості впроваджується модульна архітектура, яка полегшує додавання нових методів редукції, а також гнучка система налаштувань, що дозволяє розробникам інтегрувати нові можливості.

Зручність використання досягається завдяки інтуїтивно зрозумілому користувацькому інтерфейсу, який забезпечує зручне управління процесом редукції зображень. Простий і мінімалістичний дизайн орієнтований на полегшення роботи користувачів, навіть тих, хто має обмежені ресурси. Система дозволяє ефективно обробляти зображення на комп'ютерах із середніми характеристиками без надмірних вимог до апаратного забезпечення.

### 1.3.7 Можливості розширення системи

Перспективи розвитку системи охоплюють кілька напрямів. Інтеграція додаткових методів редукції, через модуль MyReductionModels.py, включає впровадження більш просунутих алгоритмів машинного навчання для зниження розмірності. А також додавання користувацьких моделей класифікації через модуль MyEvaluationModels.py для розширення набору доступних класифікаторів. Функціонал буде розширений для обробки відео, роботи з хмарними сховищами та іншими типами візуальних даних. Розвиток користувацького інтерфейсу спрямований на покращення зручності налаштування пайплайнів та візуалізації результатів. Планується впровадження рекомендаційних систем для вибору оптимальних методів редукції. Також розглядається інтеграція з іншими системами, що забезпечить можливість прямої передачі даних на вхід моделей машинного навчання та взаємодію з хмарними сервісами обробки зображень і комп'ютерного зору.

## Висновки до розділу

У результаті аналізу було визначено ключові підходи до редукції зображень, їх переваги та недоліки в контексті оптимізації машинного навчання. Проведено огляд існуючих аналогів системи, що дозволило сформулювати основні вимоги до розробки та виокремити її унікальні функціональні можливості. Розкрито необхідність створення додатку, який здатен зменшити обчислювальні витрати та підвищити ефективність навчання моделей

# РОЗДІЛ 2. ТЕОРЕТИЧНІ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

## 2.1 Бізнес-ризики

Технічні ризики:

* Опис: Складність розробки ефективної багатоетапної стратегії редукції даних може призвести до викликів у забезпеченні суттєвого підвищення продуктивності без критичних втрат точності.

Можливі наслідки: Затримки у розробці, втрата довіри користувачів через недоліки у функціонуванні продукту.

Ризик низького попиту:

* Опис: Низька поінформованість цільової аудиторії про переваги редукції даних або наявність альтернативних рішень може спричинити недостатній попит на розроблений продукт.

Можливі наслідки: Затримки у випуску продукту, низький рівень прибутковості, необхідність додаткових маркетингових зусиль.

Ризик швидкого розвитку конкурентних технологій:

* Опис: Швидкий розвиток конкуруючих технологій може призвести до втрати конкурентоспроможності розробленого продукту.

Можливі наслідки: Втрата ринкової позиції, зниження обсягів продажів, необхідність швидкої адаптації до нових тенденцій.

Ризик несумісності з існуючими системами клієнтів:

* Опис: Несумісність розробленого рішення з існуючими робочими процесами та системами клієнтів може ускладнити його впровадження та використання.

Можливі наслідки: Затримки у впровадженні, незадоволеність клієнтів, втрата довіри.

Ризики безпеки та конфіденційності даних:

* Опис: Недоліки у застосуванні стратегії редукції даних можуть створити загрози для безпеки та конфіденційності інформації користувачів.

Можливі наслідки: Виток конфіденційної інформації, штрафи за порушення законодавства про захист персональних даних.

Регуляторні ризики:

* Опис: Вимоги прозорості та пояснюваності моделей машинного навчання можуть стати перешкодою для впровадження продукту в певних галузях.

Можливі наслідки: Обмеження доступу до ринку, необхідність додаткових зусиль для відповідності регулятивним вимогам.

## 2.2 Теоретичне обґрунтування методів редукції зображень для впровадження в систему

### 2.2.1 Singular Value Decomposition

Для реалізації багатоетапної стратегії редукції зображень без значної втрати якості роботи основної моделі можна використовувати комбінації основних алгоритмів машинного навчання.

Одним із них є сингулярне розкладання (SVD, Singular Value Decomposition): SVD є потужним лінійним методом для виявлення найбільш значущих компонент у даних. Застосування SVD до матриці пікселів зображення дозволяє знайти найбільш інформативні сингулярні вектори, які представляють основні патерни та структури в даних. Зберігаючи лише найбільш значущі сингулярні вектори, можна досягти ефективного стиснення зображення [8].

Алгоритм розкладу сингулярних значень є одним із фундаментальних методів лінійної алгебри, що має широке застосування в численних наукових і технічних дисциплінах. SVD дозволяє поділити матрицю на кілька компонент, які зберігають інформацію про її геометричну структуру, що робить цей підхід надзвичайно корисним для аналізу даних, машинного навчання, обробки зображень та багатьох інших галузей.

Розклад сингулярних значень дозволяє представити будь-яку прямокутну матрицю A розміром m × n у вигляді добутку трьох інших матриць:

(2.1)

де:

- U — ортогональна матриця розміром m × m, стовпці якої є лівими сингулярними векторами матриці A.

- V — ортогональна матриця розміром n × n, стовпці якої є правими сингулярними векторами матриці A.

- Σ — діагональна матриця розміром m × n, що містить сингулярні значення матриці A вздовж головної діагоналі. Сингулярні значення є невід’ємними дійсними числами, що розташовані в порядку спадання.

Сингулярні значення відображають розмірність та орієнтацію матриці, вказуючи, наскільки «потужним» є кожний напрямок у просторі, що визначається відповідними сингулярними векторами.

Для обчислення розкладу SVD матриця A спочатку множиться на свою транспоновану матрицю A^T, що дозволяє знайти власні значення та власні вектори симетричної матриці. Ці власні вектори формують матриці U і V, а корені власних значень утворюють елементи діагональної матриці Σ.

Технічно, сингулярні значення матриці A дорівнюють квадратним кореням її власних значень, а стовпці матриць U і V є власними векторами для матриць AA^T і A^T A відповідно. Застосування сучасних числових методів, таких як QR-розклад, дозволяє значно прискорити обчислення SVD для великих матриць.

Завдяки своїй здатності забезпечувати унікальне представлення даних у просторі, розклад сингулярних значень широко застосовується в різних галузях:

* Зниження розмірності: SVD дозволяє обмежити кількість необхідних компонент, відкидаючи незначні сингулярні значення, що допомагає виділяти найбільш важливі ознаки та уникати надлишкових даних.
* Обробка зображень: SVD допомагає стиснути зображення, зберігаючи найбільш важливу інформацію, що дозволяє зменшити розмір файлів без значної втрати якості.
* Рекомендаційні системи: завдяки розкладу матриці користувачів та їхніх уподобань у рекомендаційних системах, SVD допомагає побудувати моделі, що здатні передбачати можливі оцінки відсутніх значень.
* Пошук інформації та аналіз тексту: SVD застосовується в латентно-семантичному аналізі для виявлення прихованих смислів у текстах, де допомагає знаходити зв'язок між словами та документами.

Однією з основних переваг SVD є його універсальність та застосовність до будь-якої матриці, незалежно від її форми чи рангу. SVD також забезпечує стійкість до шуму, оскільки дозволяє обмежити внесок малозначущих компонентів і таким чином відфільтрувати їх. Це особливо корисно в аналізі великих наборів даних, де важливо виділити основні тенденції та уникати дрібних коливань. SVD має також властивість точного відновлення початкової матриці (за відсутності відкидання сингулярних значень), що забезпечує високу точність представлення [2].

Однак, SVD має обмеження, оскільки він є лінійним методом і може втрачати важливі нелінійні патерни в даних. Також обчислення SVD є досить ресурсомістким процесом, особливо для великих матриць, що може потребувати значної кількості обчислювальних ресурсів і часу. Крім того, сам алгоритм не завжди є інтуїтивно зрозумілим, а результати розкладу вимагають подальшої обробки та інтерпретації, що може бути складним при високих розмірностях даних. SVD також може бути чутливим до точності обчислень у випадку роботи з великими значеннями, що може призводити до втрати точності при обробці надто великих або надто малих чисел. Тому доцільно доповнити його нелінійними методами.

### 2.2.2 Principal Component Analysis

Алгоритм головних компонент (Principal Component Analysis, PCA) є одним з основних методів зниження розмірності в аналітиці даних. PCA допомагає зменшити кількість змінних у великому наборі даних, зберігаючи при цьому максимум інформації. Використання цього методу дозволяє виділити найбільш важливі ознаки даних, що підвищує ефективність аналізу та полегшує візуалізацію.

Алгоритм PCA базується на пошуку нових координатних осей, які максимізують дисперсію даних. Спочатку PCA перетворює дані так, щоб їх середнє значення дорівнювало нулю, а потім визначає нові ортогональні осі, які відповідають напрямкам максимальної дисперсії. Ці осі називаються головними компонентами, і кожна наступна компонента буде максимально ортогональною попереднім.

Нехай дані представлені матрицею X розміром m × n, де m — кількість спостережень, а n — кількість ознак. PCA можна розділити на кілька основних етапів:

1. Центрування даних: Спочатку обчислюють середнє значення для кожної змінної, а потім віднімають це середнє від кожного значення змінної. Це дозволяє отримати середнє значення, що дорівнює нулю.
2. Обчислення коваріаційної матриці: Коваріаційна матриця відображає зв'язок між змінними. Для даних з нульовим середнім значенням коваріаційна матриця 𝐶 може бути обчислена як добуток транспонованої матриці 𝑋 на саму себе:

(2.2)

1. Обчислення власних значень і власних векторів: Власні вектори коваріаційної матриці визначають напрямки головних компонент, тоді як власні значення визначають, яку частку дисперсії пояснює кожна компонента. Власні значення впорядковуються за спаданням, і компоненти, що відповідають найбільшим власним значенням, мають найбільше значення для опису варіації даних.
2. Вибір головних компонент: Вибирають лише ті компоненти, які відповідають найбільшим власним значенням, що дозволяє скоротити кількість вимірів без втрати значущої інформації.
3. Проєкція даних на новий простір: Дані проєктуються на простір, створений обраними головними компонентами, що дає знижений за розмірністю набір даних.

Метод головних компонент (PCA) знаходить широке застосування в різноманітних галузях обробки та аналізу даних.

По-перше, він ефективно вирішує завдання зниження розмірності. Зменшення кількості ознак у наборі даних дає змогу суттєво скоротити обчислювальні витрати та уникнути ризику перенавчання в системах машинного навчання.

Крім того, PCA забезпечує ефективне стиснення даних. Цей метод дозволяє зберігати максимальну кількість інформації в мінімальному обсязі пам'яті, що особливо корисно при роботі з великими масивами даних.

Важливою перевагою є також можливість візуалізації багатовимірних даних. Проєктування складних наборів даних на двовимірний або тривимірний простір значно спрощує їх інтерпретацію та аналіз.

Окремо слід відзначити здатність PCA виділяти найбільш значущі ознаки. Метод допомагає ідентифікувати основні напрямки варіації, що має велике значення для побудови ефективних моделей та точної класифікації даних.

PCA є потужним інструментом, що дозволяє значно скоротити розмірність даних, зберігаючи основну інформацію. Його основні переваги включають:

* Зниження розмірності: PCA ефективно зменшує кількість змінних, що особливо корисно в роботі з великими наборами даних.
* Виділення найбільш значущих ознак: PCA дає можливість ігнорувати менш значущі компоненти, що покращує якість моделі.
* Зниження впливу шуму: Видаляючи менш важливі компоненти, PCA допомагає зменшити вплив шуму в даних.

Недоліки PCA включають:

* Лінійність: PCA працює лише з лінійними взаємозв'язками між змінними, що обмежує його ефективність для складніших, нелінійних задач.
* Чутливість до масштабування: Якщо змінні мають різний масштаб, PCA може дати спотворені результати, тому зазвичай перед використанням алгоритму дані необхідно нормалізувати.
* Втрата інформації: Зниження розмірності може призвести до втрати частини даних, особливо якщо використовуються лише кілька головних компонент.

### 2.2.3 Autoencoders

Автоенкодери — це тип нейронних мереж, які використовуються для навчання ефективного кодування даних, зазвичай без міток. Алгоритм автоенкодера складається з двох основних частин: енкодера та декодера. Енкодер приймає вхідні дані та стискає їх, створюючи стиснутий представлення, або кодування. Декодер, у свою чергу, використовує це кодування для відновлення вхідних даних.

Основні елементи автоенкодера:

* Енкодер — це частина мережі, яка приймає вхідні дані та трансформує їх у стислий формат. Це відбувається завдяки зменшенню розмірності даних, яка все ж таки повинна зберігати суттєві особливості для відновлення.
* Кодування (Latent Space Representation) — стислий, прихований шар автоенкодера, що містить суттєву інформацію про вхідні дані. Цей шар дозволяє представити дані в меншому обсязі, що є корисним для зберігання або обробки великих наборів даних.
* Декодер — це частина мережі, що приймає кодування як вхідні дані та намагається відновити оригінальні дані. Метою є мінімізувати різницю між відновленими та оригінальними даними.

Алгоритм автоенкодера навчається шляхом мінімізації різниці між виходом декодера та оригінальними вхідними даними. Це може бути виконано шляхом мінімізації функції втрат, такої як середньоквадратична похибка (Mean Squared Error, MSE), що вимірює відстань між вхідними та відновленими даними. Мережа оптимізується таким чином, щоб зменшити цю різницю, і, в результаті, автоенкодер навчається стискати інформацію.

Існують наступні типи автоенкодерів: класичний автоенкодер — найпростіший тип, де енкодер і декодер складаються з декількох шарів. Варіаційний автоенкодер (VAE) — генерує ймовірнісне представлення даних, що дозволяє генерувати нові дані, подібні до вхідних. Стискаючий автоенкодер (Sparse Autoencoder) — використовує обмеження, щоб уникати надмірної активності нейронів. Шумопригнічувальний автоенкодер (Denoising Autoencoder) — навчається відновлювати дані, пошкоджені шумом, що робить його корисним для очищення даних.

Переваги автоенкодерів:

* Здатність виявляти складні нелінійні патерни в даних.
* Зниження розмірності: Ефективно стискають інформацію та виділяють основні характеристики, корисні для подальшої обробки.
* Шумопригнічування: Можуть очищати дані від випадкового шуму, підвищуючи їхню якість.
* Генерація нових даних: Варіаційні автоенкодери здатні генерувати нові зразки даних, що є корисним для задач генерації зображень або синтетичних даних.
* Аналіз аномалій: Використовуючи здатність до реконструкції, автоенкодери можуть виділяти аномальні дані, що відрізняються від основного набору (популярно в задачах детекції шахрайства).
* Гнучкість архітектури: Існує багато типів автоенкодерів (наприклад, варіаційні, рекурентні), що дозволяє адаптувати їх до різних типів завдань, таких як обробка зображень, тексту або часового ряду.

Недоліки автоенкодерів:

* Складність у навчанні: Навчання автоенкодера може бути складним, особливо на великих наборах даних, оскільки архітектура часто вимагає великої обчислювальної потужності.
* Немає гарантій щодо латентного простору: Латентний простір може не відображати інтуїтивно зрозумілі ознаки даних, через що його важко використовувати для інтерпретації.
* Не працюють на нових типах даних: Автоенкодери погано узагальнюють на нові, відмінні від тренувальних, дані (низька здатність до генералізації), через що можуть видавати неточні результати для непередбачених даних.
* Висока чутливість до вибору гіперпараметрів: Налаштування автоенкодерів потребує ретельного підбору гіперпараметрів, таких як кількість нейронів у прихованих шарах, функції активації тощо, що може займати час і ресурси.

### 2.2.4 Fast Independent Component Analysis

Метод FastICA (Fast Independent Component Analysis) – це ефективний алгоритм для вирішення задачі незалежного компонентного аналізу (ICA), часто застосовуваний для розділення сигналів, особливо коли йдеться про виявлення незалежних джерел з сумішей. Його головна мета – знаходження таких лінійних комбінацій спостережуваних даних, які є статистично незалежними, що корисно для аналізу зображень, звуку, тексту та інших даних.

FastICA базується на ітераційному методі пошуку незалежних компонентів, який використовує підхід максимізації негентропії. На відміну від традиційних методів ICA, таких як градієнтний метод, FastICA має вищу швидкість збіжності завдяки використанню обмеженого другого порядку Тейлора та спеціальних нелінійних функцій, що робить алгоритм швидким та ефективним. Основний підхід включає обчислення окремих компонентів один за одним з наступною ортомалізацією.

Основні етапи роботи алгоритму включають:

1. Центрування даних: Зменшення середнього значення даних до нуля.
2. Сферування даних: Нормалізація дисперсії даних, що дозволяє зробити обробку стабільнішою [13].
3. Вибір нелінійної функції та максимізація негентропії. FastICA пропонує кілька варіантів функцій, таких як логарифмічна функція та функція g(x) = x^3, що дозволяє адаптувати алгоритм до різних типів даних.
4. Використання фіксованих точок для пошуку оптимальних значень незалежних компонентів, що мінімізує кількість необхідних ітерацій.

Переваги: FastICA є одним із найбільш ефективних і швидких методів ICA завдяки своїй збіжності за фіксованою кількістю ітерацій. Це робить його особливо привабливим для роботи з великими обсягами даних. Алгоритм забезпечує стабільність, оскільки його результат практично не залежить від початкових умов. Крім того, він може працювати без жорстких вимог до масштабування даних і має високу точність у відновленні незалежних компонентів, що підходить для різних типів завдань, таких як аналіз звукових сигналів і зображень.

Недоліки: попри свої переваги, FastICA не є універсальним інструментом. Алгоритм чутливий до вибору нелінійної функції, і невдалий вибір може призвести до втрати якості розділення сигналів. Крім того, метод не завжди працює ефективно з високозашумленими даними, особливо коли джерела сигналів сильно корельовані або містять лише малі відмінності. FastICA не гарантує найкращого рішення для складних сигналів, що може вимагати доповнення іншими методами.

### 2.2.5 t-Distributed Stochastic Neighbor Embedding

Алгоритм t-SNE (t-Distributed Stochastic Neighbor Embedding) є популярним методом для зменшення розмірності даних і візуалізації високовимірних структур у просторах, які можна легко інтерпретувати (зазвичай у двовимірних або тривимірних). Він особливо корисний для аналізу складних структур даних, таких як набори даних, де елементи мають приховані залежності або групування, що дозволяє побачити кластери чи взаємозв'язки між об'єктами.

Принцип роботи t-SNE полягає в тому, щоб зберегти локальну структуру даних, тобто об'єкти, які близькі у високовимірному просторі, повинні залишатися близькими й у низьковимірному. Цього досягають шляхом моделювання подібності між точками через гауссові розподіли у високовимірному просторі та Стьюдентові розподіли в низьковимірному просторі. Алгоритм мінімізує дивергенцію Кульбака-Лейблера між розподілами у високовимірному та низьковимірному просторах, досягаючи точного відображення локальної структури даних.

Процес роботи алгоритму включає:

1. Визначення подібності між точками у високовимірному просторі з використанням гауссового розподілу. Кожній точці призначається ймовірність близькості до інших точок.
2. Розподіл точок у низьковимірному просторі на основі розподілу Стьюдента, що дозволяє створювати мапу точок, які відображають структуру подібності у зменшеній розмірності.
3. Оптимізацію положення точок для мінімізації різниці між подібностями в обох просторах, що дозволяє зберегти локальні властивості даних і покращує візуалізацію.

Переваги: t-SNE є одним із найефективніших алгоритмів для виявлення та візуалізації кластерів у високовимірних наборах даних, оскільки він добре зберігає локальні структури, роблячи його особливо корисним для завдань класифікації та аналізу тексту, зображень або біологічних даних. Алгоритм демонструє високу точність у візуалізації природних груп у даних і допомагає краще зрозуміти внутрішню структуру великих і складних наборів.

Недоліки t-SNE: попри високу точність у збереженні локальних зв'язків, t-SNE має певні обмеження. Він чутливий до вибору параметрів, зокрема гіперпараметра "перплексія", і може давати значно різні результати при зміні цих значень. Алгоритм є обчислювально затратним і може бути повільним на великих наборах даних. Крім того, через фокус на локальних зв'язках, t-SNE може спотворювати глобальну структуру, через що іноді важко інтерпретувати відстані між кластерами.

### 2.2.6 Non-negative Matrix Factorization

Алгоритм NMF (Non-negative Matrix Factorization) є методом факторизації матриць, у якому оригінальна матриця розкладається на дві менші матриці, що містять тільки невід’ємні елементи. Це обмеження на невід’ємність робить NMF особливо корисним для аналізу даних, де негативні значення не мають сенсу, наприклад, в аналізі тексту, біоінформатиці, а також у зображеннях і звуках. Мета алгоритму полягає у представленні складної багатовимірної інформації через невід’ємні базиси, що дозволяє виявляти приховані структури і шаблони в даних.

Принцип роботи NMF полягає у тому, щоб знайти дві матриці – матрицю компонентів та матрицю ваг, – множення яких максимально наближено відтворює оригінальну матрицю. Це робиться шляхом мінімізації відстані (найчастіше – квадратичної похибки) між оригінальною матрицею і добутком двох нових матриць. Ключова особливість алгоритму в тому, що обмеження на невід’ємність змушує компоненти бути інтерпретованими як "частини" вихідних даних, що робить його корисним для завдань, де дані можна розкласти на складові частини (наприклад, теми у тексті).

Основні етапи роботи NMF:

1. Ініціалізація матриць компонентів та ваг випадковими невід’ємними значеннями.
2. Ітеративна оптимізація шляхом оновлення обох матриць, з метою зменшення похибки відновлення.
3. Завершення роботи після досягнення конвергенції або досягнення заданого порогу помилки.

Переваги: NMF є потужним інструментом для інтерпретованої декомпозиції даних, оскільки його факторизаційні компоненти можна трактувати як "будівельні блоки" оригінальних даних. Це особливо корисно для завдань класифікації, сегментації та кластеризації, зокрема, у таких областях як обробка природної мови (виділення тем), біоінформатика (аналіз експресії генів) та аналіз зображень (виявлення об'єктів). Крім того, алгоритм є відносно обчислювально ефективним, особливо для матриць середнього розміру, і легко масштабується.

Один з основних недоліків NMF – це його чутливість до початкових умов: оскільки іниціалізація матриць відбувається випадково, різні запуски можуть призводити до різних рішень, що може ускладнювати відтворюваність. Крім того, NMF часто потребує додаткового налаштування та вибору кількості компонентів, що є гіперпараметром алгоритму і може впливати на якість результатів. Алгоритм також може бути неефективним для даних із великою кількістю нулів, оскільки факторизація таких даних може призводити до великої похибки відновлення.

### 2.2.7 Підсумки обгрунтування вибору методів редукції

Вибір представлених методів (SVD, PCA, Autoencoders, FastICA, t-SNE, NMF) є стратегічно обґрунтованим з наступних причин:

1. Комплементарність підходів:

* SVD та PCA забезпечують надійну лінійну декомпозицію даних, що є фундаментальною основою для початкової редукції
* Autoencoders доповнюють лінійні методи здатністю виявляти складні нелінійні патерни
* FastICA, t-SNE та NMF привносять специфічні переваги для роботи з різними аспектами зображень

1. Збалансованість між ефективністю та якістю:

* SVD і PCA є обчислювально ефективними для початкової швидкої редукції
* Autoencoders можуть забезпечити більш точне збереження важливих ознак
* t-SNE та NMF дозволяють зберегти локальну структуру даних

1. Взаємна компенсація недоліків:

* Де лінійні методи (SVD, PCA) мають обмеження, нелінійні (Autoencoders, t-SNE) можуть забезпечити кращі результати
* Комбінування методів дозволяє знизити чутливість до шуму та покращити стабільність результатів

1. Масштабованість та адаптивність:

* Різні методи можуть бути застосовані залежно від розміру вхідних даних
* Можливість конфігурації параметрів кожного методу дозволяє точно налаштувати процес редукції

1. Практична реалізовність:

* Всі обрані методи мають добре документовані реалізації
* Існують ефективні бібліотеки для їх застосування
* Методи можуть бути інтегровані в єдиний конвеєр обробки

Саме такий набір методів створює комплексний інструментарій для багатоетапної редукції зображень, де кожен метод може бути застосований на відповідному етапі залежно від конкретних вимог до балансу між ступенем стиснення та збереженням значущої інформації. Це дозволяє досягти оптимального співвідношення між ресурсоємністю та якістю роботи класифікаційної моделі.

## 2.3 Обгрунтування вибору алгоритмів класифікації для оцінки оптимального ступеня редукції

### 2.3.1 Multi-Layer Perceptron Classifier

MLPClassifier (Multi-Layer Perceptron Classifier) є різновидом штучної нейронної мережі, що зазвичай застосовується для класифікації складних наборів даних, зокрема зображень. У задачах редукції даних він відіграє роль трансформатора вхідного простору, дозволяючи отримувати більш компактні, зменшені представлення, які все ще зберігають важливі ознаки початкових зображень. Це здійснюється через процес навчання, під час якого алгоритм проходить кілька ітерацій узгодження ваг нейронів, використовуючи пошарові перетворення вхідних даних.

У випадку застосування MLPClassifier для редукції зображень, мережа приймає на вхід високорозмірні дані, представлені пікселями зображення, і пропускає їх через приховані шари нейронів. Ці шари "вчаться" виявляти патерни і особливості, що є характерними для кожного класу зображень. Вихідний шар класифікує зображення за категоріями, спираючись на узагальнені ознаки, що виділяються мережею на попередніх рівнях. Під час редукції MLPClassifier адаптує параметри так, щоб зберегти ключові ознаки, при цьому мінімізуючи зайву інформацію, що не впливає на кінцеву класифікацію.

Основна перевага MLPClassifier полягає в його здатності опановувати нелінійні зв’язки між даними, що є надзвичайно корисним для зображень, які містять складні структури та текстури. Додатково, завдяки прихованим шарам мережа здатна виділяти як локальні, так і глобальні ознаки, що покращує якість редукції, оскільки навіть після зменшення розмірності даних залишається велика частина важливої інформації [2].

Проте, є й певні недоліки. По-перше, MLPClassifier вимагає великої кількості обчислювальних ресурсів для налаштування ваг мережі, особливо при обробці високорозмірних зображень. Це може бути проблематично при роботі з великими наборами зображень, де виникає потреба в оптимізації швидкості. По-друге, алгоритм схильний до перенавчання, особливо якщо навчальний набір є невеликим чи недостатньо збалансованим, що може призвести до зниження точності редукції. Також налаштування гіперпараметрів MLPClassifier, таких як кількість прихованих шарів або нейронів на кожному з них, є нетривіальним завданням, що вимагає глибокого розуміння даних і багато експериментів [2].

У підсумку, MLPClassifier є ефективним інструментом для редукції зображень завдяки своїй здатності виділяти суттєві ознаки навіть у високооб'ємних даних. Однак, його використання може бути обмежене високими обчислювальними вимогами і чутливістю до налаштувань, що слід враховувати при виборі його для конкретних застосувань у редукції даних.

### 2.3.2 Support Vector Classifier

Support Vector Classifier (SVC) – це алгоритм машинного навчання, що базується на методі опорних векторів (SVM) і широко використовується для класифікації зображень. Основна ідея SVC – знайти гіперплощину в багатовимірному просторі, яка оптимально розділяє дані за класами. У випадку задач редукції зображень SVC часто застосовується для стиснення інформації, де алгоритм перетворює високорозмірні зображення на компактнішу форму, утримуючи при цьому основні відмінні риси для подальшої класифікації.

При застосуванні до зображень SVC націлений на вибір невеликої кількості ключових ознак, званих опорними векторами, що найкраще розділяють класи. Для редукції це означає, що SVC не тільки зменшує обсяг даних, але й фільтрує малозначущі ознаки, залишаючи лише ті, що найбільш суттєво відрізняють класи зображень. Такий підхід підходить для обробки даних з чіткими класовими кордонами, наприклад, при класифікації зображень об’єктів, де структура і текстура дозволяють легко виділити ключові особливості.

Основною перевагою SVC є його висока ефективність у випадках, коли класи зображень мають явну межу розділення. Відповідно, для редукції це означає, що SVC часто надає високоточні результати навіть з невеликою кількістю ознак, що робить його ефективним в умовах обмежених обчислювальних ресурсів. Інша перевага полягає в можливості використовувати різні ядра (наприклад, лінійне, RBF, поліноміальне) для побудови складних меж розділення, що дозволяє гнучко налаштовувати алгоритм під специфіку даних.

Однак, є й недоліки. Алгоритм SVC зазвичай не працює належним чином при значній кількості класів і при високій розмірності зображень, оскільки обчислення оптимальної гіперплощини стає надто ресурсоємним. Це може бути критичним для обробки великих масивів зображень, де обчислювальні витрати швидко зростають. Крім того, при нерівномірному розподілі класів або накладенні даних, SVC може стати чутливим до шуму і давати менш точні результати редукції. Налаштування гіперпараметрів, таких як коефіцієнт регуляризації та тип ядра, також може вимагати великої кількості експериментів для досягнення оптимальних результатів.

У підсумку, SVC є потужним інструментом для редукції зображень, особливо коли дані мають чіткі границі між класами. Проте, його застосування обмежується високими обчислювальними вимогами та необхідністю вірного налаштування для досягнення стабільної точності, що слід враховувати у відповідних проектах.

### 2.3.3 Decision Tree Classifier

DecisionTreeClassifier — це алгоритм машинного навчання, заснований на побудові дерева рішень, яке використовує послідовність простих правил для класифікації даних. Кожне розгалуження дерева відповідає певному питанню або критерію, яке ділить набір даних на підгрупи, ґрунтуючись на обраній ознаці, що забезпечує оптимальну роздільність. У задачах редукції зображень DecisionTreeClassifier працює над виявленням найважливіших ознак, що можуть ефективно представляти зображення, скорочуючи при цьому загальну кількість необхідних даних.

У процесі роботи DecisionTreeClassifier оцінює ознаки вхідних даних і відбирає лише ті, що є найсуттєвішими для класифікації. У контексті зображень це означає, що дерево вибудовує правила для виділення певних характеристик, таких як колірні градієнти, контури, або текстури, що чітко розділяють класи об’єктів. Для редукції цей алгоритм дозволяє зменшити обсяг даних, залишивши при цьому найважливіші для класифікації властивості, що сприяє підвищенню швидкості обробки та ефективності класифікації.

Однією з основних переваг DecisionTreeClassifier є його інтуїтивність і простота у використанні. На відміну від інших алгоритмів, він не потребує нормалізації або масштабування ознак, що робить його зручним для застосування навіть при роботі з різноманітними зображеннями. Крім того, дерево рішень може легко адаптуватися до нових даних і визначати нелінійні зв’язки, що є корисним для складних зображень. Його побудова часто є досить швидкою, що робить його привабливим варіантом для редукції великих масивів зображень.

Однак, DecisionTreeClassifier має і певні недоліки. Зокрема, він схильний до перенавчання (overfitting), особливо на невеликих або нестійких наборах даних. Це може знижувати якість редукції, коли дані мають високий рівень шуму або накладення класів. Крім того, рішення, ухвалені деревом, можуть бути залежними від початкових виборів ознак, що обмежує точність при обробці особливо складних зображень. Для подолання цих обмежень часто використовується ансамблевий підхід, як-от випадковий ліс (Random Forest), який згладжує результати.

Таким чином, DecisionTreeClassifier є потужним і легким у використанні інструментом для редукції зображень, особливо коли необхідно швидко отримати результат і дані мають достатню роздільність. Проте, для досягнення стійкої точності потрібне ретельне налаштування та, за потреби, використання ансамблевих методів.

### 2.3.4 Random Forest Classifier

RandomForestClassifier — це ансамблевий метод машинного навчання, що об'єднує декілька дерев рішень, кожне з яких є незалежним класифікатором. Кінцеве рішення ухвалюється шляхом голосування або усереднення результатів від окремих дерев, що підвищує стабільність і точність класифікації. У задачах редукції зображень RandomForestClassifier дозволяє створити зменшену за обсягом і водночас інформативну модель, яка зберігає головні характеристики зображення, необхідні для точного розпізнавання.

У процесі роботи RandomForestClassifier генерує кілька дерев рішень, використовуючи різні підмножини ознак та прикладів з навчальної вибірки. Це дозволяє уникнути перенавчання (overfitting) і зменшити вплив шуму в даних, що особливо корисно для складних зображень, де окремі дерева могли б давати нестабільні результати. Кожне дерево аналізує різні аспекти зображення, наприклад, текстури, контури, або колірні градієнти, що підвищує шанси на виявлення найсуттєвіших ознак. В результаті, у редукції зображень RandomForestClassifier ефективно зменшує обсяг даних, залишаючи лише ті ознаки, які сприяють точній класифікації, що суттєво оптимізує процес обробки зображень.

Основною перевагою RandomForestClassifier є його висока точність та стійкість до перенавчання, оскільки об'єднання багатьох дерев дозволяє уникнути помилок, які могли б виникати в окремих дерев. Крім того, алгоритм менш чутливий до шуму в даних, що робить його підходящим для обробки зображень, де можливе накладання класів або наявність артефактів. Ще однією перевагою є автоматичний вибір найбільш значущих ознак, що дозволяє суттєво спростити моделі для редукції.

Однак, RandomForestClassifier має і свої недоліки. Через велику кількість дерев, його обчислювальна складність може бути високою, що потребує більше ресурсів для тренування та передбачення. Це може стати критичним при обробці великих масивів зображень. Крім того, інтерпретованість результатів є обмеженою, оскільки кінцеве рішення є усередненим результатом багатьох дерев, і відстежити, яке саме дерево вплинуло на рішення, може бути складно. Така складність обмежує можливість повного аналізу важливості окремих ознак у високорозмірних даних.

Загалом, RandomForestClassifier є надійним і точним методом для редукції зображень, який забезпечує баланс між ефективністю та точністю. Проте для великих і складних даних він вимагає додаткових обчислювальних ресурсів та може потребувати компромісів між швидкістю обробки та інформативністю результатів.

### 2.3.5 Підсумки обгрунтування вибору алгоритмів класифікації

У контексті розробки інформаційної технології оптимізації ресурсоємності класифікаційних моделей, вибір набору методів класифікації для оцінки якості редукції є критично важливим етапом. Ефективна оцінка збереження значущої інформації після редукції вимагає використання різнопланових класифікаційних моделей, кожна з яких має свої унікальні характеристики та підходи до розпізнавання патернів у даних [3,9]. Саме тому було обрано набір класифікаторів, що представляють різні парадигми машинного навчання та дозволяють всебічно оцінити якість збереження інформативних ознак після процесу редукції. Вибір конкретних методів обґрунтовується наступними міркуваннями:

1. Різноманітність підходів до класифікації:

* MLPClassifier забезпечує нелінійне перетворення через багатошарову архітектуру
* SVC фокусується на пошуку оптимальних границь розділення класів
* DecisionTreeClassifier та RandomForestClassifier представляють різні підходи до дерево-базованої класифікації
* Комбінація цих методів дозволяє всебічно оцінити якість редукції

1. Взаємодоповнюючі характеристики:

* MLPClassifier ефективний для виявлення складних нелінійних патернів
* SVC оптимальний для чітких границь між класами
* DecisionTreeClassifier забезпечує прозорість рішень
* RandomForestClassifier підвищує стабільність через ансамблевий підхід

1. Різна чутливість до редукції даних:

* MLPClassifier чутливий до якості збережених ознак через багатошарову структуру
* SVC залежить від якості розділення в зменшеному просторі ознак
* Tree-based методи дозволяють оцінити збереження ієрархії важливих характеристик

1. Практичні аспекти реалізації:

* Всі методи мають ефективні реалізації в популярних бібліотеках
* Різна обчислювальна складність дозволяє оцінити компроміс між точністю та ресурсоємністю
* Можливість легкого порівняння результатів через стандартизовані метрики

1. Комплексна оцінка якості редукції:

* Різні моделі по-різному реагують на втрату інформації при редукції
* Порівняння результатів дозволяє виявити оптимальний ступінь стиснення
* Забезпечується надійна верифікація збереження значущих ознак

Такий набір класифікаторів створює комплексну технологію для оцінки якості редукції, де кожна модель оцінює різні аспекти збереження інформації. Це дозволяє знайти оптимальний баланс між ступенем редукції та збереженням класифікаційної здатності даних.

## 2.4 Експериментальні дослідження

### 2.4.1 Розробка кастомної моделі адаптивного автоенкодеру

Було впроведено серію експерементальних досліджень відносно архітектури автоенкодера.

Лістинг 2.1. - Архітектура автоенкодера v1.0

latent\_dim = 16

inputs = Input(shape=(784,))

encoded = Dense(128, activation='relu')(inputs)

encoded = Dense(latent\_dim, activation='relu')(encoded)

decoded = Dense(128, activation='relu')(encoded)

decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = tf.keras.Model(inputs, decoded)

autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.fit(x\_train\_flat, x\_train\_flat\_cut, epochs=10)

Результати для даної моделі наступні:

Час навчання моделі - 39.09 сек

Час затрачений для передбачення всіх x\_test(10000) - 0.043 сек

Розмір моделі в оперативній пам'яті: 48 байт

Розмір файлу-моделі на диску: 625.42 KB

Точність передбачення оцінюючої моделі: 96%

Лістинг 2.2. – Архітекрура автоенкодера v2.0

input\_shape = (28, 28, 1)

inputs = Input(shape=input\_shape)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(

inputs) # Output: 28x28 (avoid information loss)

x = MaxPooling2D((2, 2), padding='valid')(x) # Output: 14x14

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='valid')(x) # Output: 7x7

x = Flatten()(x)

encoded = Dense(self.lat\_dim\_ae)(x)

x = Dense(7 \* 7 \* 64)(encoded)

x = Reshape((7, 7, 64))(x)

x = Conv2D(64, (3, 3), activation='relu',

padding='same')(x) # Output: 7x7

x = UpSampling2D((2, 2))(x) # Output: 14x14

x = Conv2D(32, (3, 3), activation='relu',

padding='same')(x) # Output: 14x14

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(1, (3, 3), activation='sigmoid',

padding='same')(x) # Output: 28x28x1

self.autoencoder = tf.keras.Model(inputs, decoded)

self.autoencoder.compile(optimizer='adam', loss='mse')

Результати оцінювання на архітектурі автоенкодера другої версії:

Час навчання моделі - 41.66 сек

Час затрачений для передбачення всіх x\_test(10000) - 0.044 сек

Розмір моделі в оперативній пам'яті: 56 байт

Розмір файлу-моделі на диску: 625.42 KB

Точність передбачення оцінюючої моделі: 97%

Лістинг 2.3 – Архітекрура автоенкодера v3.0

inputs = Input(shape=self.input\_shape)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Flatten()(x)

encoded = Dense(self.lat\_dim\_ae)(x)

# Calculate the shape after flattening

conv\_output\_shape = (self.input\_shape[0] // 4, self.input\_shape[1] // 4, 64)

dense\_units = math.prod(conv\_output\_shape)

x = Dense(dense\_units)(encoded)

x = Reshape(conv\_output\_shape)(x)

# x = Dense(7 \* 7 \* 64)(encoded)

# x = Reshape((7, 7, 64))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(self.input\_shape[2], (3, 3), activation='sigmoid', padding='same')(x)

self.autoencoder = Model(inputs, decoded)

self.autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())

В підсумку модель другої версії розширилась, ускладнилась, добавилось вхідне розширення даних, що в подальшому дозволить розширити модель для обробки зображень різних розмірів і RGB. Із за цього стала вчитись на 2 секунди довше, час передбачення майже не змінився. На 18 байт збільшився розмір моделі в оперативній памяті. Але головне збільшилась точність оцінюючої моделі на 1%, що в подальшому нам дозволить виконувати редукцію даних без втрати точності оцінюючої моделі відносно оригінальних даних. Третя версія містить модифікований прихований шар, який автоматично адаптуватиметься відносно вхідного розміру зображення, що дозволяє автоенкодеру приймати зображення різної форми та з різною кількістю каналів.

Процес навчання моделей редукції на прикладі автоенкодера.

Спочатку вхідні зображення проходили попередню обробку та перетворення форми до формату, зручного для згорткових нейронних мереж у фреймворку Keras (зображення у відтінках сірого розміром 28x28 пікселів). Потім ці дані подавалися на вхід автоенкодера для навчання методом [24]:

self.autoencoder.fit(x\_train, x\_train, epochs=10)

Процес навчання автоенкодера був налаштований згідно з рекомендацій [10] та [24]: з розміром батчів 32 зображень та протягом 10 епох, що було визначено як оптимальне значення для забезпечення належної якості навчання, швидкості та уникнення перенавчання. Така кількість епох забезпечувала належну збіжність моделі без надмірного перенавчання. Динаміка зміни функції втрат демонструвала стабільне зменшення протягом навчання. Функція втрат, використана для навчання автоенкодера, - середньоквадратична помилка (MSE) між вхідними та реконструйованими зображеннями. Результати наведені нижче:

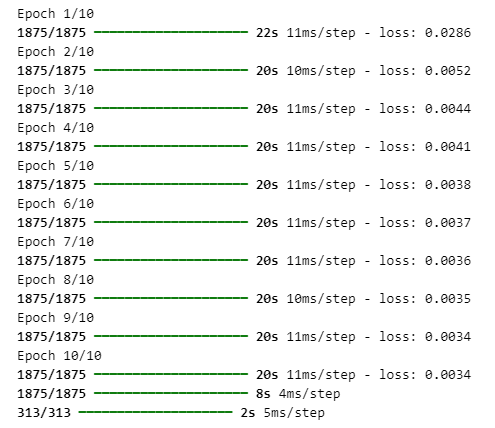


Рисунок 2.1 - Процес тренування автоенкодера

З рисунка 2.1 видно, що кожна епоха навчання в середньому займала 19 секунд, а загальний час навчання склав близько 195 секунд. Як результат, значення функції втрат становило 0.0033, що є прийнятно низьким показником.

### 2.4.2 Експерименти з підбором оптимальних параметрів для окремих алгоритмів редукції

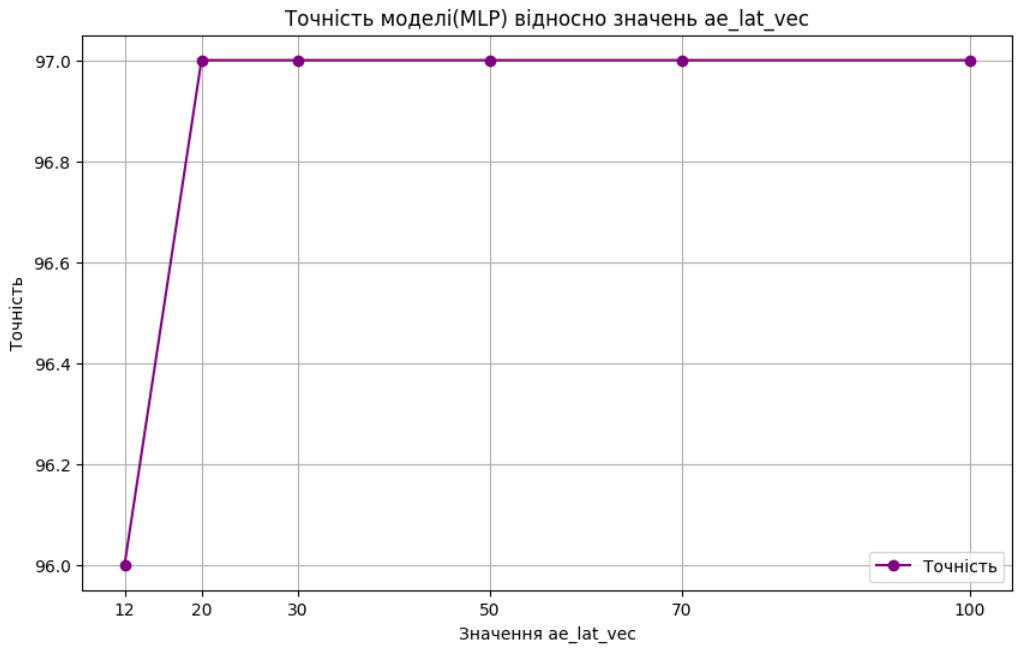


Рисунок 2.2 - Графік з відношенням точності до кількості латентних векторів

На рисунку 2.2 зображений графік який відображає точність моделі при налаштуванні параметру латентних векторів автоенкодера. Ми бачимо що всі значення, крім 12, мають хороший відсоток передбачення - 97%. Отже можна використовувати всі решта.

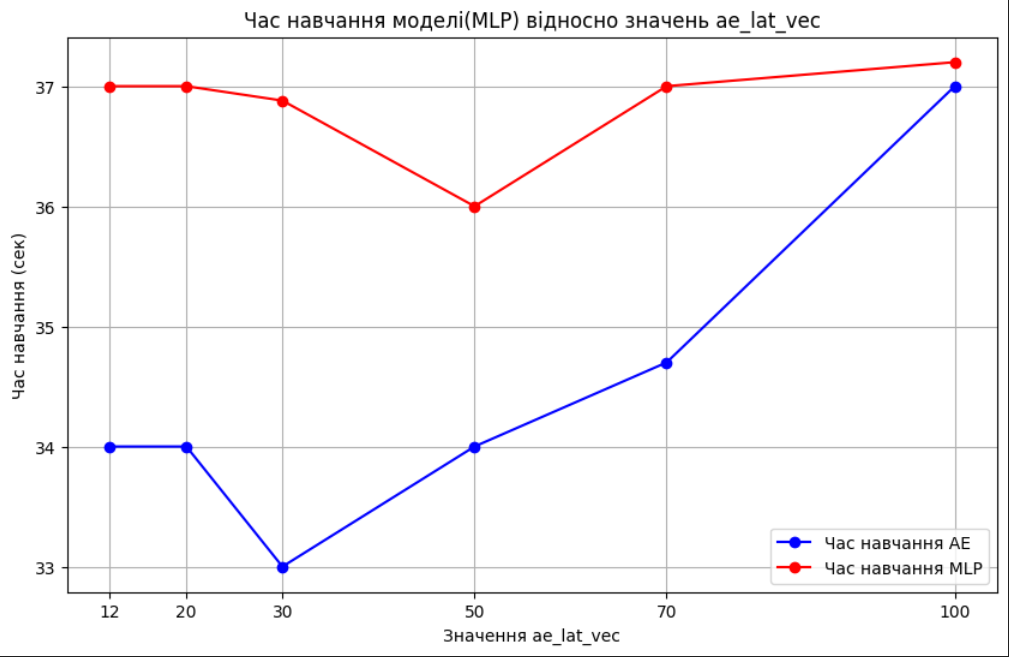


Рисунок 2.3 - Графік з відношенням часу навчання до кількості латентних векторів

На рисунку 2.3 проілюстровано взаємозв'язок між розмірністю латентного вектора автоенкодера та тривалістю навчання як самого автоенкодера (синя крива), так і допоміжної перевірної моделі (червона крива) на відновлених даних. Оскільки менша тривалість навчання є бажаною з точки зору ефективності, аналіз рисунка 3 дозволяє зробити висновок, що вибір розмірності латентного вектора 30 або 50 є оптимальним для мінімізації часу навчання обох моделей. Ця обставина набуває критичної ваги у випадках, коли необхідно часте перенавчання моделей. Варто зауважити, що більшу вагу слід надавати часу навчання перевірної моделі (червона крива), оскільки саме її продуктивність є індикатором якості редукції вихідних даних автоенкодером.



Рисунок 2.4 - Графік з відношенням часу передбачення до кількості латентних векторів

На рисунку 2.4 зображено відношення значень латентних векторів автоенкодера до часу затраченого на передбачення даних з тестової вибірки. Чим менший час навчання - тим краще. Цей параметр є важливим якщо модель буде використовуватись в режимі реального часу. А отже значення ближчі до 70 є кращими.

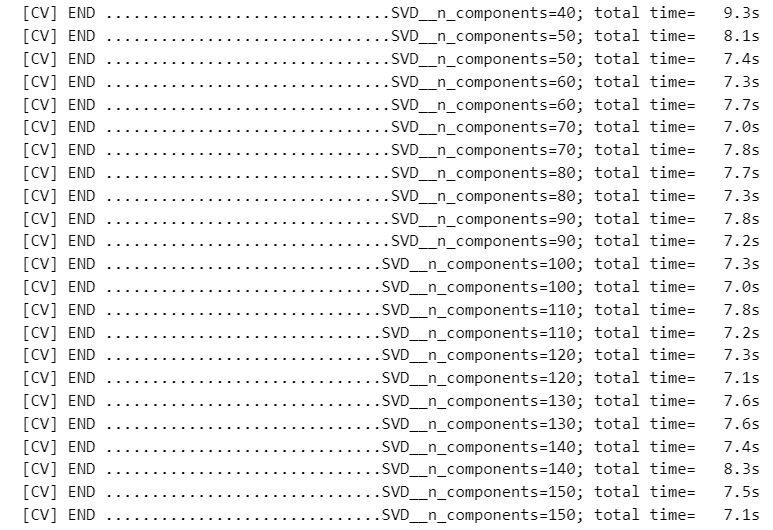


Рисунок 2.5 - Вивід в консоль при підборі параметрів.

Повний підбір параметрів, який запускався на центральному процесорі, та зайняв близько 260 секунд (рис. 2.5). Але час може відрізнятись в залежності від конфігурації комп’ютера.

Результат підбору параметрів наступний:

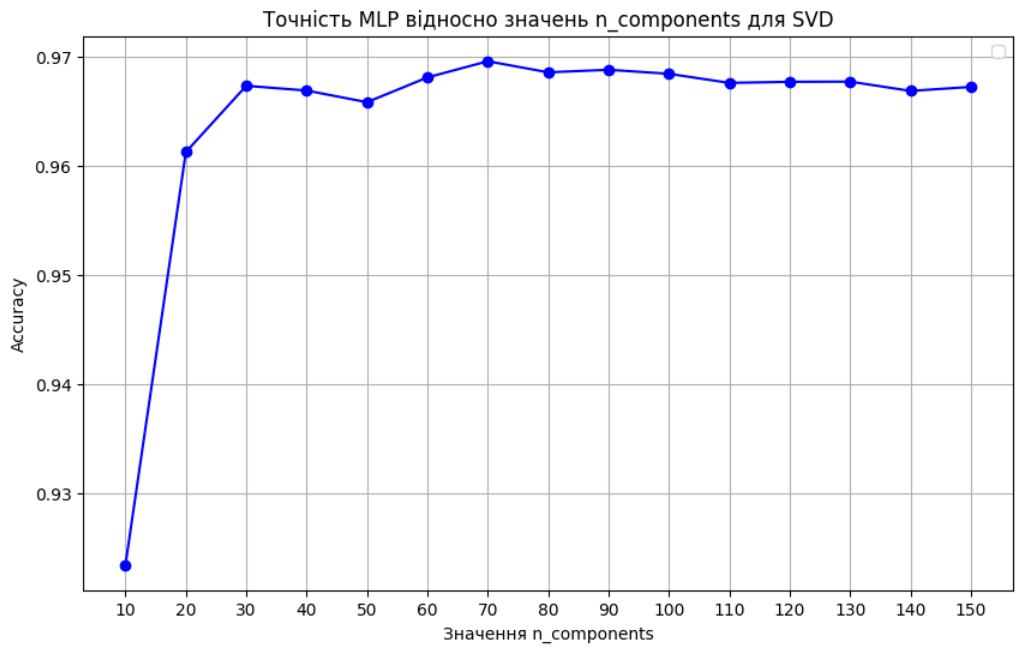


Рисунок 2.6 - Графік з відношенням точності до кількості компонентів SVD

На рисунку 2.6 зображено точність моделі MLP для заданих параметрів n\_components. Всі значення, окрім 15, можна вважати хорошими, адже вони мають точність вище 96%.

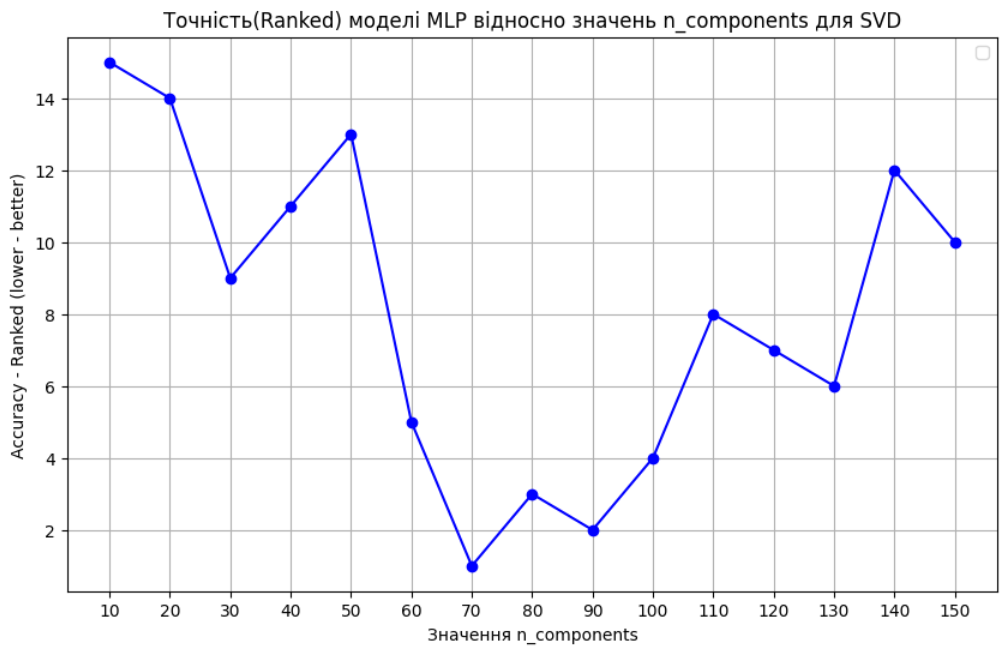


Рисунок 2.7 - Графік з відношенням точності до кількості компонентів SVD, але в іншому представлення

Рисунок 2.7 візуалізує ту ж саму точність що і на попередньому рисунку, але у форматі ranked. Тобто значення точності були перетворені у числа для кращого розуміння, яке із значень більше чи менше. Можна бачити що параметр 70 буде найкращим рішенням.

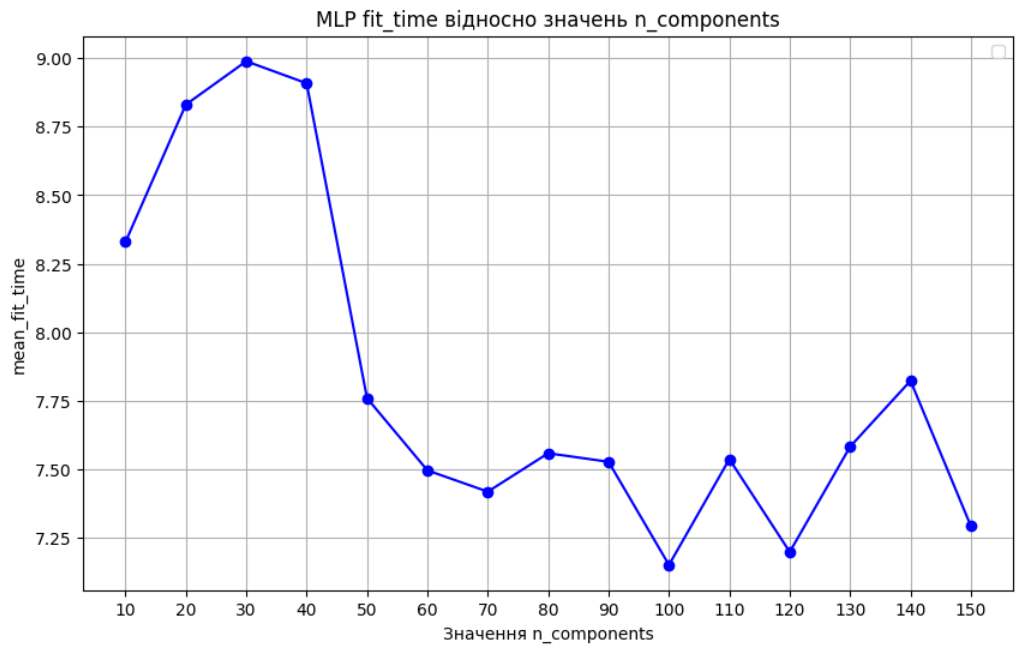


Рисунок 2.8 - Графік з відношенням часу навчання до кількості компонентів SVD

На рисунку 2.8 зображено графік з відношенням підібраних параметрів до часу навчання перевірочної моделі. Чим менше значення тим краще. Отже якщо модель для редукції даних планується перенавчати багато раз - то буде доцільно враховувати цей параметр і вибирати значення 100 або 120.



Рисунок 2.9 - Графік з відношенням часу передбачення до кількості компонентів SVD

На рисунку 2.9 зображено графік з відношенням тих же параметрів до часу передбачення моделлю на тестовій вибірці. Чим менший час - тим краще. Якщо модель планується використовувати в режимі реального часу або важлива висока швидкість відповіді - варто вибирати параметр з найменшим часом відгуку, наприклад 20.

Варто звернути увагу, що вибір параметру залежить від мети завдання і іноді потрібно вибирати оптимальне значення базуючись на наведених вище графіках.

Таким чином, аналіз графіків залежності точності класифікації, тривалості навчання та часу передбачення від цих параметрів дозволив виявити оптимальні їх значення для забезпечення балансу між якістю редукції даних та обчислювальною ефективністю моделей.

Для автоенкодера оптимальним визнано значення розмірності латентного вектора 32, яке демонструє високу точність передбачення 97%, прийнятний час навчання та час передбачення. Для SVD найкращим виявився параметр n\_components = 70, що забезпечує точність вище 96% та задовільні показники навчання та передбачення.

Отже, вибір оптимальних параметрів здійснювався з урахуванням специфічних вимог задачі, зокрема необхідності часто перенавчати моделі або використовувати їх у режимі реального часу. Ретельна оптимізація гіперпараметрів є критично важливою для досягнення оптимального балансу між продуктивністю та якістю роботи систем машинного навчання в практичних застосуваннях.

### 2.4.3 Експерименти з підбором оптимальної архітектури конвеєра в залежності від характеристик зображень

На рисунку 2.10 зображено результати експериментів класифікації з використанням MLP-класифікатора на даних, редукованих за допомогою комбінації трьох алгоритмів: автоенкодера (AE), Truncated SVD, та PCA. Вісь X позначає конфігурації гіперпараметрів для кожного алгоритму редукції (латентна розмірність автоенкодера, кількість компонент SVD та PCA). Вісь Y розділена на три метрики: точність моделі (зелена крива), час навчання (fit\_time, синя крива), і час оцінки (score\_time, червона крива).

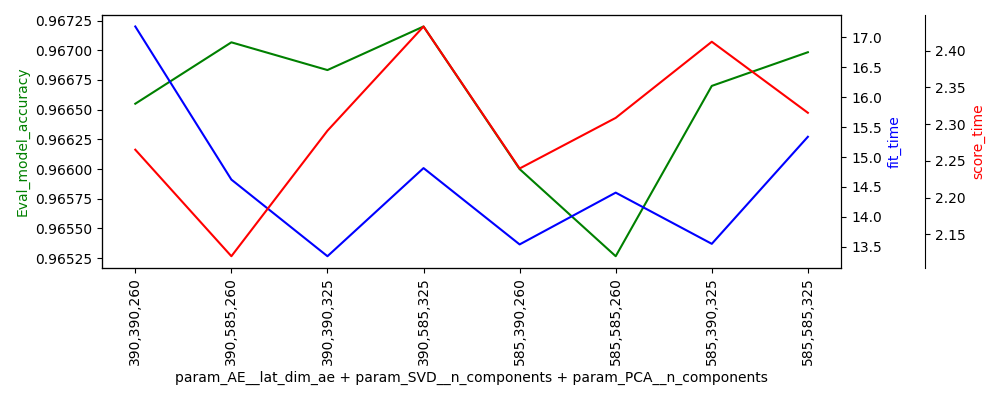


Рисунок 2.10 – Графік експерименту на моделях AE,SVD,PCA

З графіка видно, що точність моделі коливається у вузькому діапазоні (близько 0.9655–0.9672), тоді як час навчання та оцінки демонструють більшу варіативність залежно від параметрів. Найменший час навчання спостерігається при середніх параметрах, тоді як час оцінки є стабільнішим. Це свідчить, що оптимізація гіперпараметрів важлива для досягнення балансу між продуктивністю та швидкістю.

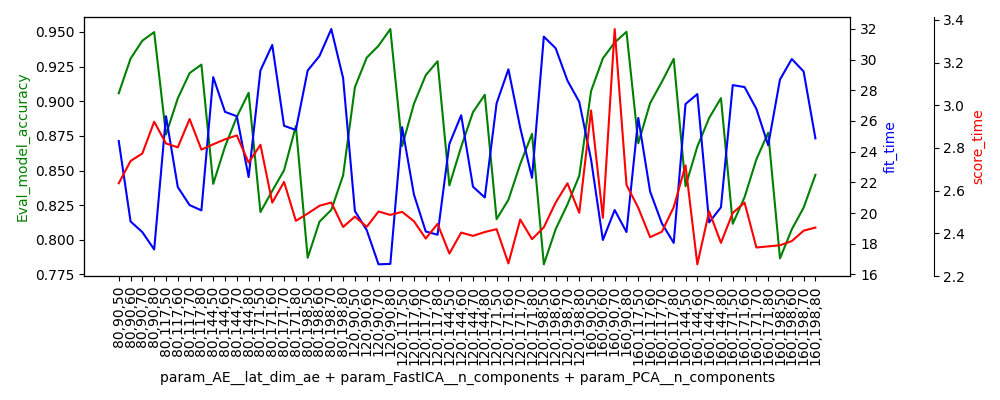


Рисунок 2.11 – Графік експерименту на моделях AE, FastICA, PCA

Графік на рисунку 2.11 демонструє високу варіативність точності моделі, яка коливається в діапазоні від 0.775 до 0.95. Час навчання значно змінюється між різними конфігураціями, досягаючи максимуму понад 30 секунд, тоді як час оцінки (score\_time) залишається відносно стабільним, варіюючись між 2.2 та 3.4 секунди. Ці результати показують складність знаходження оптимальної конфігурації гіперпараметрів для збереження як точності, так і продуктивності.

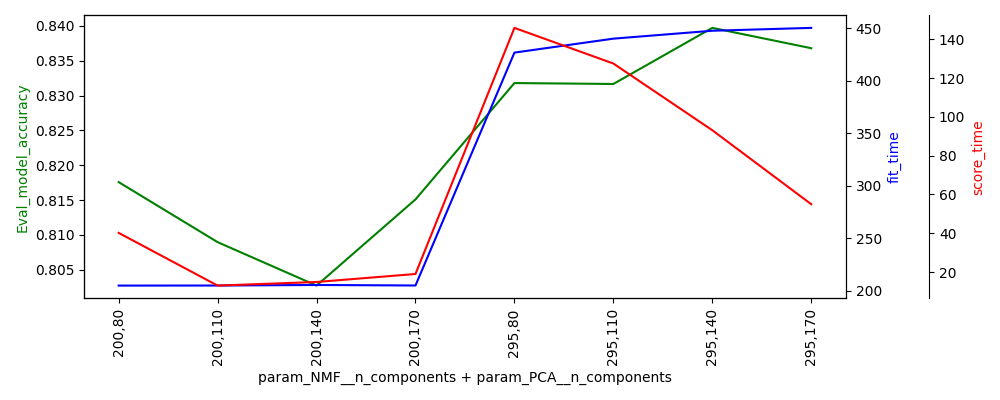


Рисунок 2.12 – Графік експерименту на моделях NMF, PCA

Результати показують, що точність моделі поступово покращується при збільшенні кількості компонентів, досягаючи піку приблизно на рівні 0.84. Час навчання (fit\_time) та час оцінки (score\_time) також значно зростають зі збільшенням параметрів, що вказує на компроміс між продуктивністю та якістю класифікації. Наочно видно кореляцію між числом параметрів для алгоритму NMF, точністю та часом відгуку. При збільшенні параметру, тобто чим менше ступінь стиснення – тим більше точність, але неймовірно зростає час навчання цієї моделі, що робить використання NMF недоцільним у рішеннях де потрібно часто створювати, перенавчати конвеєр редукції. Для досягнення балансу між точністю та швидкістю відгуку, варто обирати параметри з високою точністю (наприклад, 295,140), але при цьому час навчання буде дуже неймовірно великий.

## Висновки до розділу

Проведені теоретичні дослідження підтвердили ефективність вибраних методів редукції зображень для вирішення поставлених завдань. Обгрунтовано використання алгоритмів класифікації для оцінки оптимального ступеня редукції, що дозволило забезпечити високий рівень точності та швидкодії. Експериментальні результати продемонстрували кореляцію між рівнем редукції та продуктивністю моделей машинного навчання.

# РОЗДІЛ 3. ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ

## 3.1 Моделювання предметної області

В рамках розробки інформаційної технології оптимізації ресурсоємності класифікаційних моделей проведено детальний аналіз та моделювання предметної області. Система проектується як багатоетапний процес редукції даних з можливістю конфігурації параметрів та оцінки якості результатів.

Для візуалізації структури та взаємозв'язків компонентів системи розроблено дві ключові діаграми:

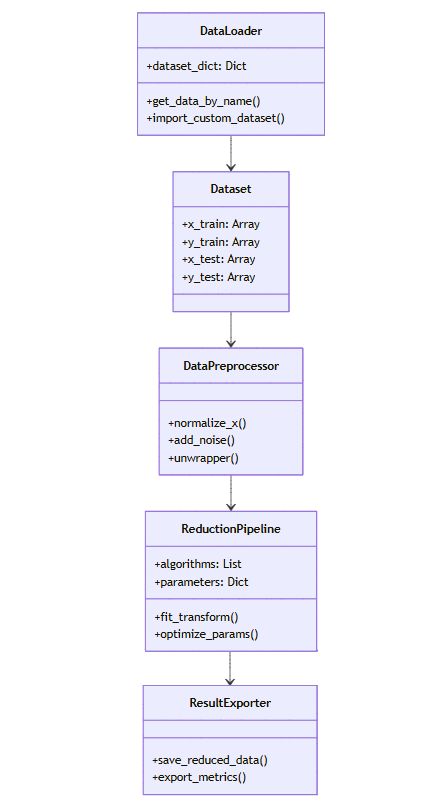


Рисунок 3.1 - Діаграма потоків даних системи

Діаграма на рис.3.1 відображає повний життєвий цикл обробки даних у системі, який складається з наступних етапів:

1. Підготовчий етап:
   * Завантаження вхідного набору даних (DataLoader).
   * Первинна обробка через Data Loader з формуванням навчальної та тестової вибірок.
   * Попередня обробка даних у Preprocessor, включаючи нормалізацію, шум, розгортання масиву у вектор.
2. Етап конфігурації, що включає три паралельних потоки користувацького введення:
   * Pipeline Configuration: визначення кількості етапів редукції (1-3)
   * Algorithms Selection: вибір конкретних алгоритмів для кожного етапу
   * Parameters Range: встановлення діапазонів параметрів оптимізації (мінімальне/максимальне значення та крок)
3. Етап оптимізації:
   * Формування пайплайну в Pipeline Constructor на основі користувацької конфігурації
   * Виконання процесу оптимізації параметрів (Parameter Optimization Process)
   * Візуалізація результатів оптимізації
   * Вибір користувачем фінальних параметрів на основі запропонованих оптимальних значень
4. Етап фінальної обробки:
   * Застосування Final Reduction Pipeline з обраними параметрами
   * Паралельна передача результатів на оцінку (Evaluation Model) та експорт (CSV Export)

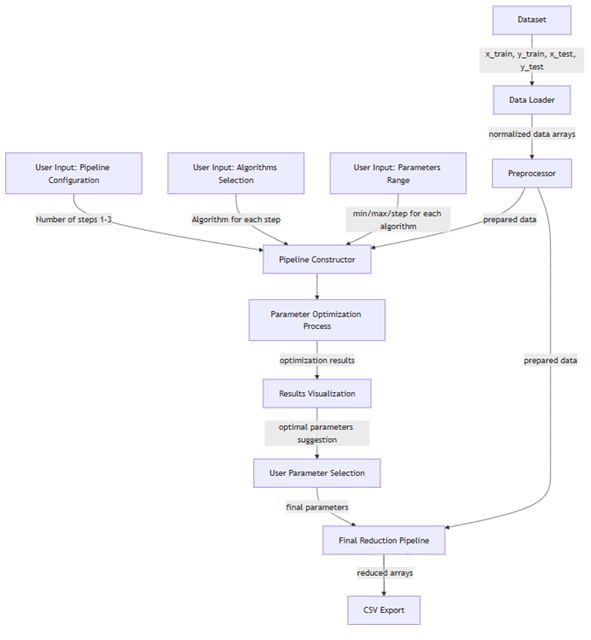


Рисунок 3.2 - UML діаграма взаємодії системи

Діаграма на рис.3.2 демонструє основні програмні компоненти системи та їх взаємозв'язки:

1. DataLoader:
   * Відповідає за завантаження даних
   * Містить словник доступних датасетів (dataset\_dict)
   * Реалізує методи get\_data\_by\_name() та import\_custom\_dataset()
2. Dataset:
   * Зберігає навчальну та тестову вибірки у вигляді масивів
   * Забезпечує уніфікований формат даних для подальшої обробки
3. DataPreprocessor:
   * Реалізує методи попередньої обробки даних
   * Включає функції нормалізації (normalize\_x), додавання шуму (add\_noise) та перетворення форматів (unwrapper)
4. ReductionPipeline:
   * Центральний компонент системи
   * Зберігає список алгоритмів та їх параметри
   * Забезпечує методи fit\_transform() та optimize\_params()
5. ResultExporter - Відповідає за збереження результатів у файл.

Розроблена модель предметної області забезпечує чітке розмежування відповідальності між компонентами системи, гнучкість у конфігурації параметрів та можливість розширення функціоналу. Використання патерну "Конвеєр" (Pipeline) дозволяє ефективно організувати послідовну обробку даних та оптимізацію параметрів алгоритмів редукції.

### 3.1.1 Архітектурний дизайн

Система реалізована як локальний Python-додаток для обробки та зменшення розмірності зображень, побудований на основі багаторівневої архітектури. В основу покладено принцип чіткого розмежування відповідальності між рівнями та компонентами системи.

Архітектура складається з наступних ключових рівнів:

1. Рівень представлення:

* Інтерфейс користувача на базі Tkinter
* Компоненти вибору та налаштування параметрів
* Візуалізація результатів та метрик

1. Рівень обробки даних:

* Модуль завантаження даних (підтримка Keras datasets та користувацьких наборів)
* Компоненти передобробки (нормалізація, векторизація)
* Конвеєр редукції даних на базі sklearn.pipeline

1. Рівень бізнес-логіки:

* Алгоритми зниження розмірності (PCA, SVD, t-SNE та інші)
* Оптимізація параметрів
* Оцінка якості редукції

В основу розробленої системи покладено багаторівневу архітектуру, що забезпечує ефективне вирішення задач зменшення розмірності даних. Для забезпечення ефективної обробки даних система реалізує послідовний потік операцій від завантаження даних до отримання кінцевих результатів.

Для деталізації взаємозв'язків між компонентами системи розроблено UML діаграму класів.

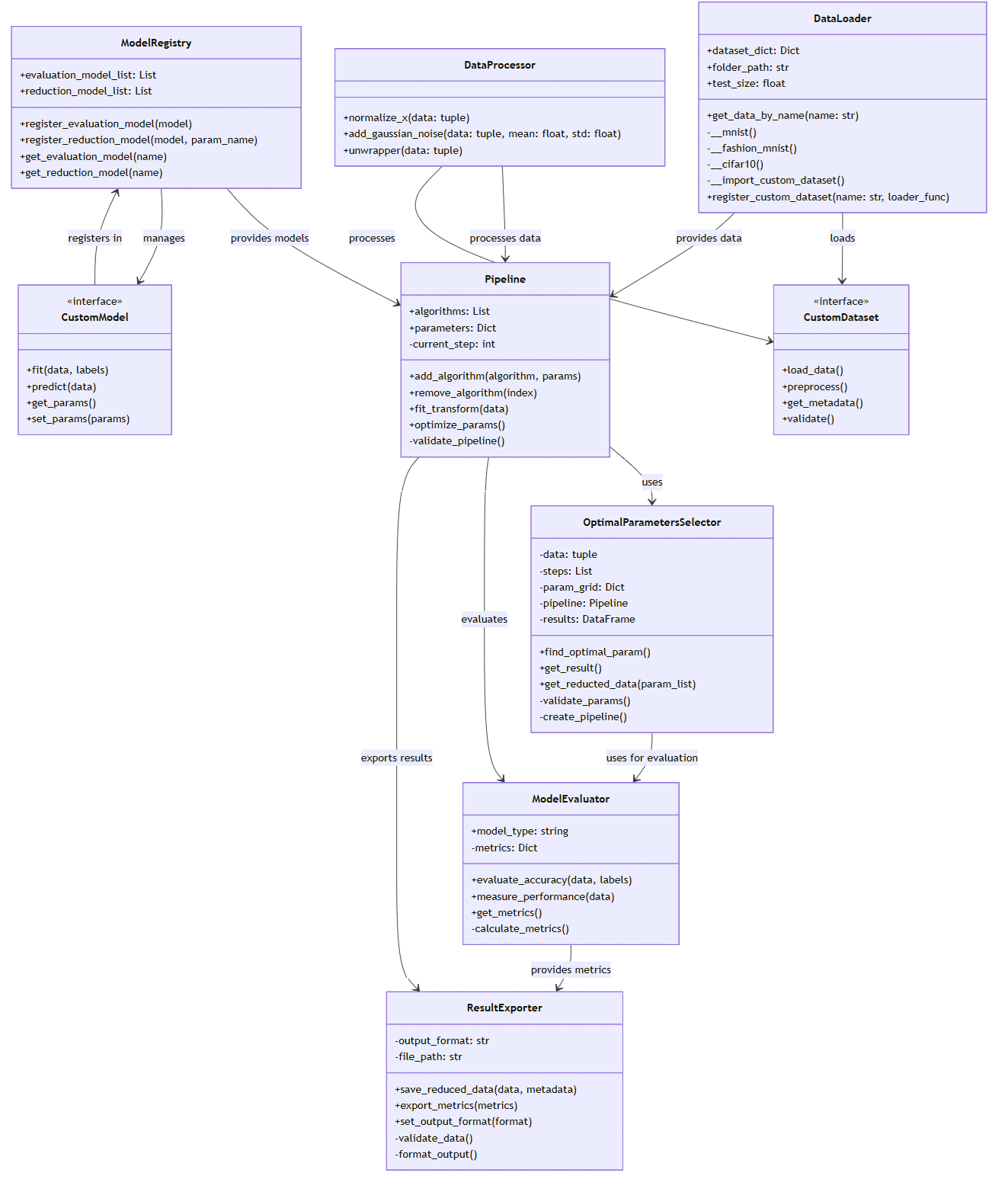


Рисунок 3.3 - UML-діаграма класів основних компонентів системи

Як видно з UML-діаграми класів на рис. 3.3, система реалізує складну взаємодію між компонентами через чітко визначені інтерфейси. Діаграма демонструє ключові класи та їх методи, що забезпечують функціональність на кожному рівні архітектури.

Ключовими особливостями архітектури є:

* Модульність та розширюваність через чітко визначені інтерфейси
* Використання патерну "Конвеєр" для гнучкої конфігурації алгоритмів
* Інкапсуляція деталей реалізації на кожному рівні
* Можливість незалежного розвитку та тестування компонентів

Така архітектурна організація забезпечує:

* Гнучкість при інтеграції нових алгоритмів та методів
* Чітке розділення відповідальності між компонентами
* Простоту розширення функціоналу
* Ефективне повторне використання коду

Методологічно архітектура системи базується на концепції багатошарової абстракції, де кожен рівень забезпечує специфічну функціональність, інкапсулюючи деталі реалізації від інших шарів. Така організація дозволяє досягти високого рівня гнучкості при інтеграції нових алгоритмів та методів обробки даних.

### 3.1.2 Функціональні компоненти та процеси системи

Розроблена система реалізує модульний підхід до організації функціональних компонентів, що забезпечує чітке розмежування відповідальності та гнучкість у розширенні функціоналу. Архітектурно система розділена на два основні функціональні рівні: клієнтський та обробний.

Клієнтська частина представлена інтерфейсом користувача, реалізованим на базі бібліотеки Tkinter, що забезпечує всі аспекти взаємодії з системою. Цей рівень включає компоненти для вибору та завантаження зображень, інтерфейс налаштування параметрів редукції даних, модулі конфігурації методів обробки (PCA, TruncatedSVD, TSNE тощо), а також компоненти для візуалізації результатів та метрик якості обробки.

Обробний рівень охоплює всю функціональність, пов'язану з безпосередньою обробкою даних. Він містить модулі зниження розмірності, компоненти для трансформації та обробки даних, підсистему збереження результатів. Важливою особливістю є реалізація модульної структури, яка забезпечує можливість легкого розширення функціоналу та додавання нових методів обробки.

Процес обробки даних в системі організований як послідовність взаємопов'язаних етапів. На етапі передобробки відбувається завантаження даних з Keras datasets або користувацьких джерел, виконується нормалізація значень пікселів до діапазону [0, 1] та здійснюється векторизація зображень для подальшої обробки.

Наступний етап - конструювання ознак - включає перетворення матриць пікселів у вектори фіксованої довжини, формування латентних векторів через алгоритми редукції та масштабування даних відповідно до вимог моделей. За цим слідує етап безпосередньої редукції розмірності, де застосовуються один або декілька алгоритмів (PCA, SVD, t-SNE), виконується їх конфігурація через sklearn.pipeline та послідовне застосування обраних методів.

Завершальний етап включає оцінку якості редукції та експорт результатів. Тут проводиться обчислення метрик якості, візуалізація результатів для порівняння та експорт оброблених даних у CSV формат.

Запропонований підхід має ряд ключових переваг. Перш за все, це ефективне зменшення розмірності при збереженні важливої інформації та виділення значущих патернів в даних. Також важливим є отримання некорельованих ознак, що суттєво покращує подальшу класифікацію, та можливість інтерпретації отриманих латентних векторів.

Реалізація системи характеризується низкою важливих особливостей. Забезпечується гнучкість у виборі джерел даних та можливість детальної конфігурації параметрів на кожному етапі обробки. Система підтримує збереження проміжних результатів, що важливо для аналізу та відлагодження процесу редукції. Особлива увага приділена візуалізації метрик для оцінки якості, що дозволяє користувачам ефективно контролювати та оптимізувати процес обробки даних.

Така організація системи створює надійну основу для ефективної обробки та редукції даних, забезпечуючи при цьому гнучкість у налаштуванні та розширенні функціоналу відповідно до конкретних потреб користувачів.

## 3.2. Проектування логічної структури системи

### 3.2.1 Загальні принципи проектування та декомпозиція системи

На етапі проектування логічної структури інформаційної технології оптимізації ресурсоємності класифікаційних моделей було застосовано багаторівневий підхід з чітким розділенням відповідальності між компонентами системи. Проектування здійснювалося з урахуванням вимог до гнучкості конфігурації, можливості розширення функціоналу та оптимізації взаємодії між компонентами.

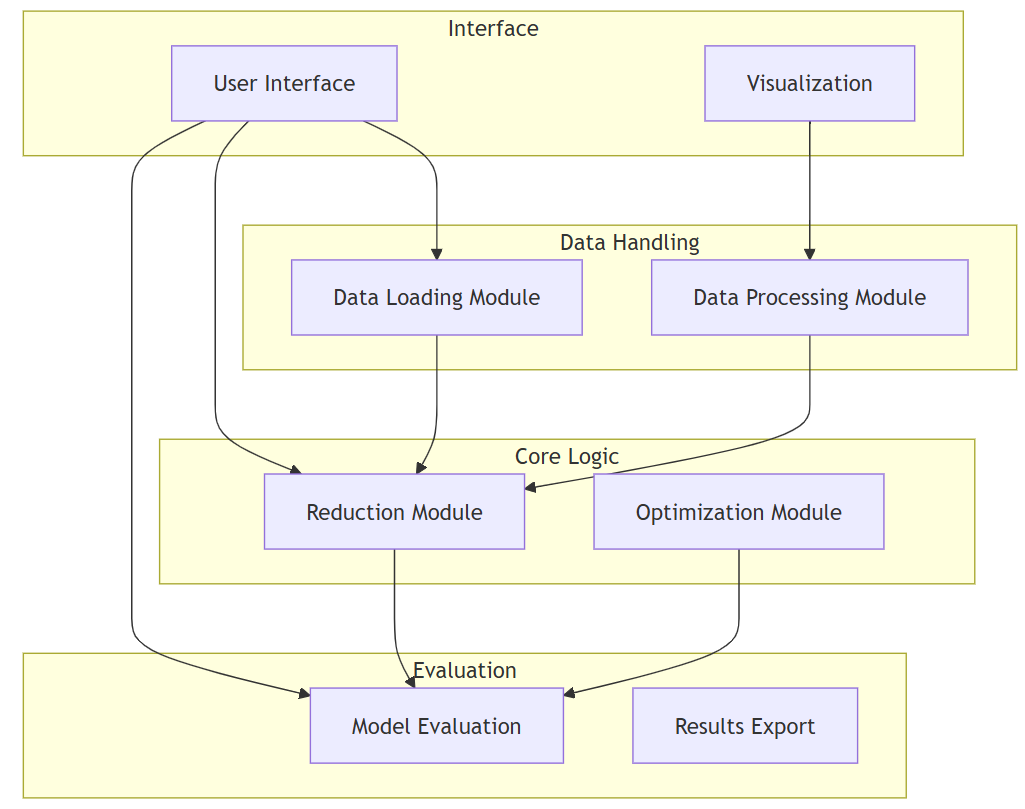


Рисунок 3.4 - Діаграма пакетів

Для забезпечення ефективної реалізації поставлених задач було розроблено чотирьохрівневу архітектуру системи (рисунок 3.4), яка включає:

1. Рівень інтерфейсу (Interface):

* Реалізує взаємодію з користувачем через графічні компоненти
* Забезпечує гнучке налаштування параметрів редукції
* Надає засоби візуалізації результатів оптимізації

1. Рівень обробки даних (Data Handling):

* Відповідає за уніфікований доступ до різних джерел даних
* Забезпечує попередню обробку та валідацію вхідних даних
* Реалізує перетворення форматів даних між компонентами

1. Рівень бізнес-логіки (Core Logic):

* Реалізує алгоритми редукції даних
* Забезпечує оптимізацію параметрів
* Координує процес багатоетапної обробки

1. Рівень оцінки (Evaluation):

* Здійснює оцінку якості редукції
* Забезпечує експорт та збереження результатів
* Формує метрики ефективності

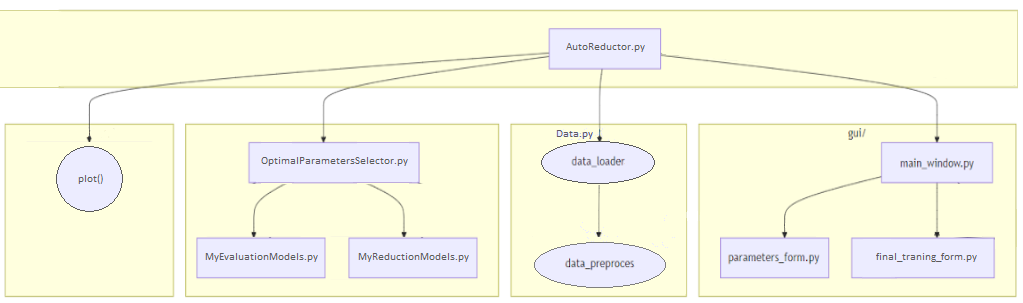


Рисунок 3.5 - Діаграма модулів системи

Фізична організація системи представлена через модульну структуру (рисунок 3.5), що забезпечує:

* Оптимальне розділення функціоналу між модулями
* Мінімізацію зв'язків між компонентами
* Можливість незалежної модифікації та тестування
* Спрощення процесу розробки та підтримки

Ключовими особливостями запропонованої архітектури є:

1. Гнучкість конфігурації:

* Можливість динамічного налаштування параметрів редукції
* Підтримка різних комбінацій алгоритмів
* Адаптивність до різних типів вхідних даних

1. Розширюваність:

* Можливість додавання нових алгоритмів редукції
* Підтримка різних методів оцінки якості
* Гнучкість у форматах експорту даних

1. Оптимізація взаємодії:

* Мінімізація накладних витрат при передачі даних
* Ефективне управління пам'яттю
* Оптимізація процесу обробки даних

1. Надійність:

* Валідація вхідних даних та параметрів
* Обробка виняткових ситуацій
* Збереження проміжних результатів

1. Масштабованість:

* Можливість паралельної обробки даних
* Підтримка різних об'ємів вхідних даних

Адаптивність до обчислювальних ресурсів

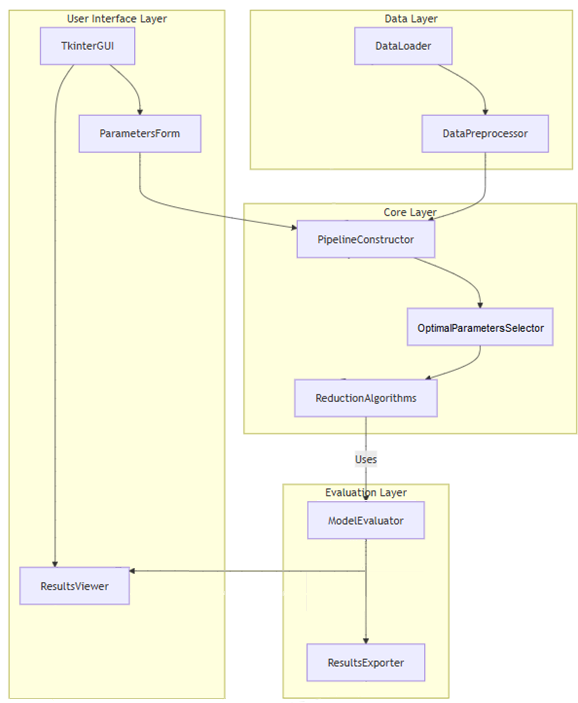


Рисунок 3.6 - Компонентна діаграма системи

Для деталізації взаємодії між компонентами системи розроблено компонентну діаграму (рисунок 3.6), яка демонструє:

* Механізми взаємодії через стандартизовані інтерфейси
* Потоки даних між компонентами
* Ієрархію залежностей компонентів

Запропонована логічна структура системи забезпечує ефективну реалізацію поставлених задач оптимізації ресурсоємності класифікаційних моделей через:

* Гнучке налаштування параметрів редукції
* Ефективну взаємодію компонентів системи
* Оптимальне використання обчислювальних ресурсів
* Можливість адаптації та розширення функціоналу

Реалізований підхід до проектування дозволяє досягти оптимального балансу між ефективністю редукції даних та обчислювальними витратами, що є критичним для задач оптимізації ресурсоємності класифікаційних моделей.

### 3.2.2 Організація шару даних та попередньої обробки

Шар даних реалізує комплексний підхід до управління вхідними даними через взаємодію компонентів DataLoader та DataProcessor. DataLoader виступає як уніфікований інтерфейс доступу до даних, підтримуючи як вбудовані набори даних (MNIST, Fashion-MNIST), так і користувацькі джерела через механізм розширення CustomDataset. DataProcessor реалізує науково обґрунтований підхід до попередньої обробки даних, включаючи:

* Нормалізацію даних з використанням адаптивних методів
* Статистичну обробку з урахуванням характеристик розподілу
* Валідацію та верифікацію якості перетворень

### 3.2.3 Ядро системи та конвеєр обробки

Центральним елементом архітектури виступає компонент Pipeline, який забезпечує технічну реалізацію процесу обробки даних. Реалізація включає:

1. Конфігурація Pipeline:

* Динамічне конструювання послідовності алгоритмів
* Моніторинг стану обробки
* Валідація вхідних та вихідних даних
* Обробка помилок та відновлення при збоях

1. OptimalParameterSelector:

* Імплементація багатокритеріальної оптимізації
* Реалізація адаптивного пошуку параметрів
* Механізми крос-валідації результатів

Процес обробки даних реалізується через наступні етапи:

1. Передобробка зображень:

* Нормалізація значень пікселів до діапазону [0, 1]
* Векторизація зображень для алгоритмів редукції
* Стандартизація форматів даних

1. Редукція розмірності:

* Використання одного або декількох алгоритмів (PCA, SVD, t-SNE)
* Конфігурація параметрів через sklearn.pipeline
* Послідовне застосування методів редукції

1. Оцінка якості:

* Застосування моделей класифікації (MLP, SVC, Decision Trees, Random Forest)
* Обчислення метрик:
  + Час навчання моделі
  + Швидкість передбачення
  + Розмір редукованих даних
  + Точність класифікації
* Візуалізація результатів для порівняння

1. Фінальна обробка:

* Налаштування оптимальних параметрів
* Експорт результатів у CSV формат
* Візуалізація ефективності редукції

### 3.2.4 Система оцінювання та експорту результатів

ModelEvaluator представляє науково обґрунтований підхід до оцінки якості зменшення розмірності, реалізуючи:

* Комплексну систему метрик якості
* Методи статистичної валідації
* Аналіз збереження топологічної структури даних

ResultExporter забезпечує не лише збереження результатів за пдопомогою функцій, але й повну відтворюваність експериментів через:

* Детальне протоколювання параметрів
* Збереження метаданих експерименту
* Форматування результатів для подальшого аналізу

### 3.2.5 Взаємодія компонентів та потоки даних

Взаємодія компонентів системи організована за принципом слабкого зв'язування через чітко визначені інтерфейси. Основний потік даних проходить через послідовність перетворень:

1. Завантаження та валідація через DataLoader
2. Попередня обробка в DataProcessor
3. Застосування алгоритмів редукції в Pipeline
4. Оптимізація параметрів через OptimalParameterSelector
5. Оцінка результатів у вигляді графіків через plot
6. Створення фінального ковеєра на основі підібраних оптимальних параметрів та редукція даних з допомогою OptimalParameterSelector.
7. Збереження даних та графіків з допомогою DataPreproces.save\_reducted\_data.

Така організація забезпечує ефективну обробку даних при збереженні можливості моніторингу та контролю якості на кожному етапі перетворень.

Розроблена архітектура демонструє науково обґрунтований підхід до створення систем зменшення розмірності даних, забезпечуючи як високу ефективність обробки, так і можливість розширення та адаптації до нових задач та методів аналізу даних.

### 3.2.6 Моделювання поведінки системи

У рамках проектування інформаційної технології оптимізації ресурсоємності класифікаційних моделей було проведено комплексне моделювання поведінки системи з використанням уніфікованої мови моделювання UML. Для забезпечення повноти аналізу застосовано два взаємодоповнюючих підходи: моделювання взаємодії компонентів через діаграму послідовності (рис. 3.7) та моделювання життєвого циклу через діаграму станів (рис. 3.8).

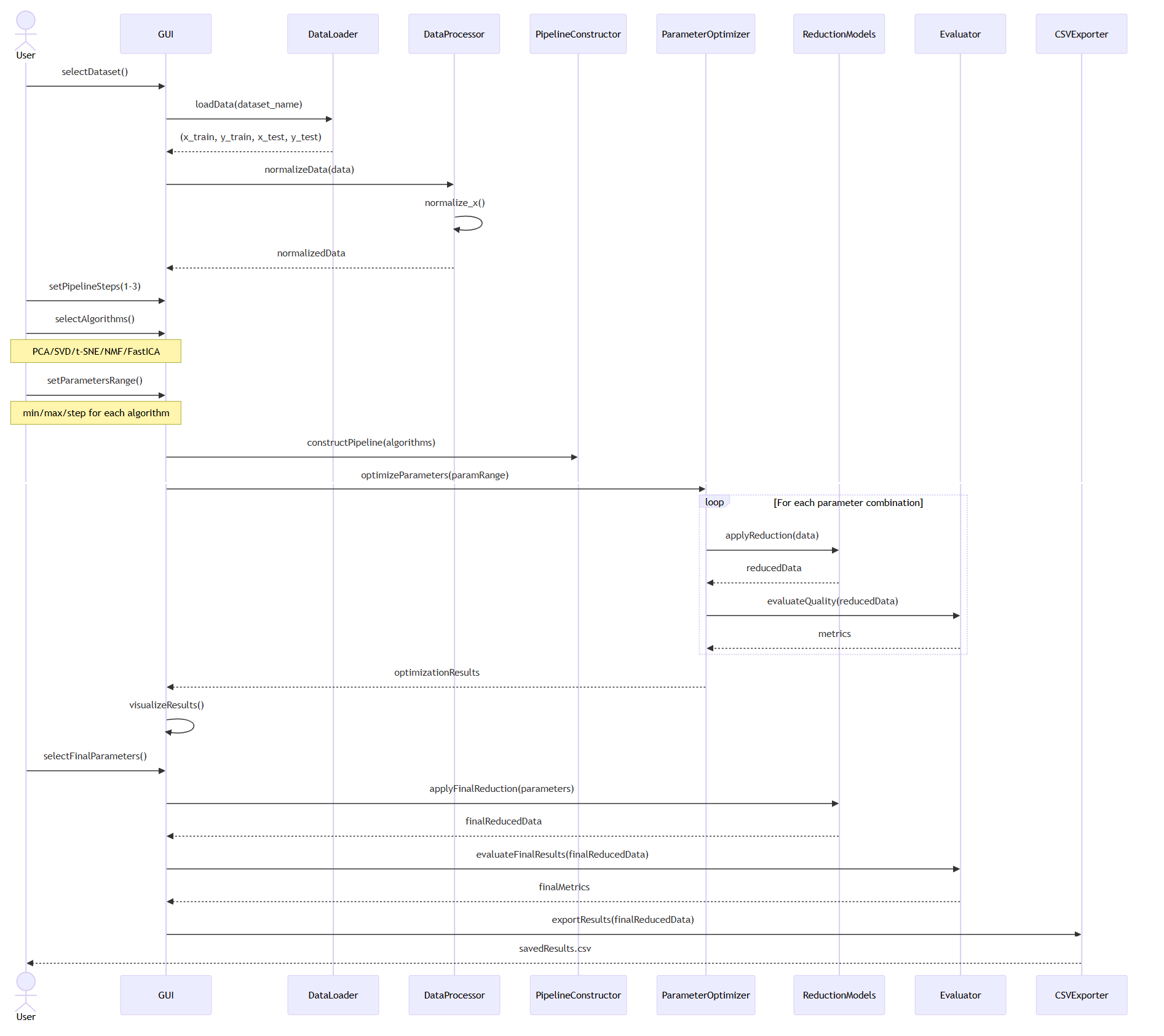


Рисунок 3.7 - Діаграма послідовності для процесу редукції

Діаграма послідовності на рис.3.7 демонструє взаємодію між головними компонентами системи: інтерфейсом користувача, завантажувачем даних, модулями попередньої обробки, конструктором пайплайну, оптимізатором параметрів, моделями редукції, оцінювачем та експортером результатів. Така декомпозиція дозволяє чітко відстежити процес передачі даних та управління між компонентами, починаючи з вибору користувачем датасету і закінчуючи збереженням оброблених даних.

Основний процес включає етапи завантаження та нормалізації даних, конфігурації пайплайну редукції з вибором алгоритмів та їх параметрів, оптимізації цих параметрів через ітеративний процес оцінки якості, та фінального застосування обраної конфігурації з подальшим збереженням результатів. Важливою особливістю є реалізація зворотного зв'язку через візуалізацію результатів та можливість користувача коригувати параметри на основі отриманих метрик.

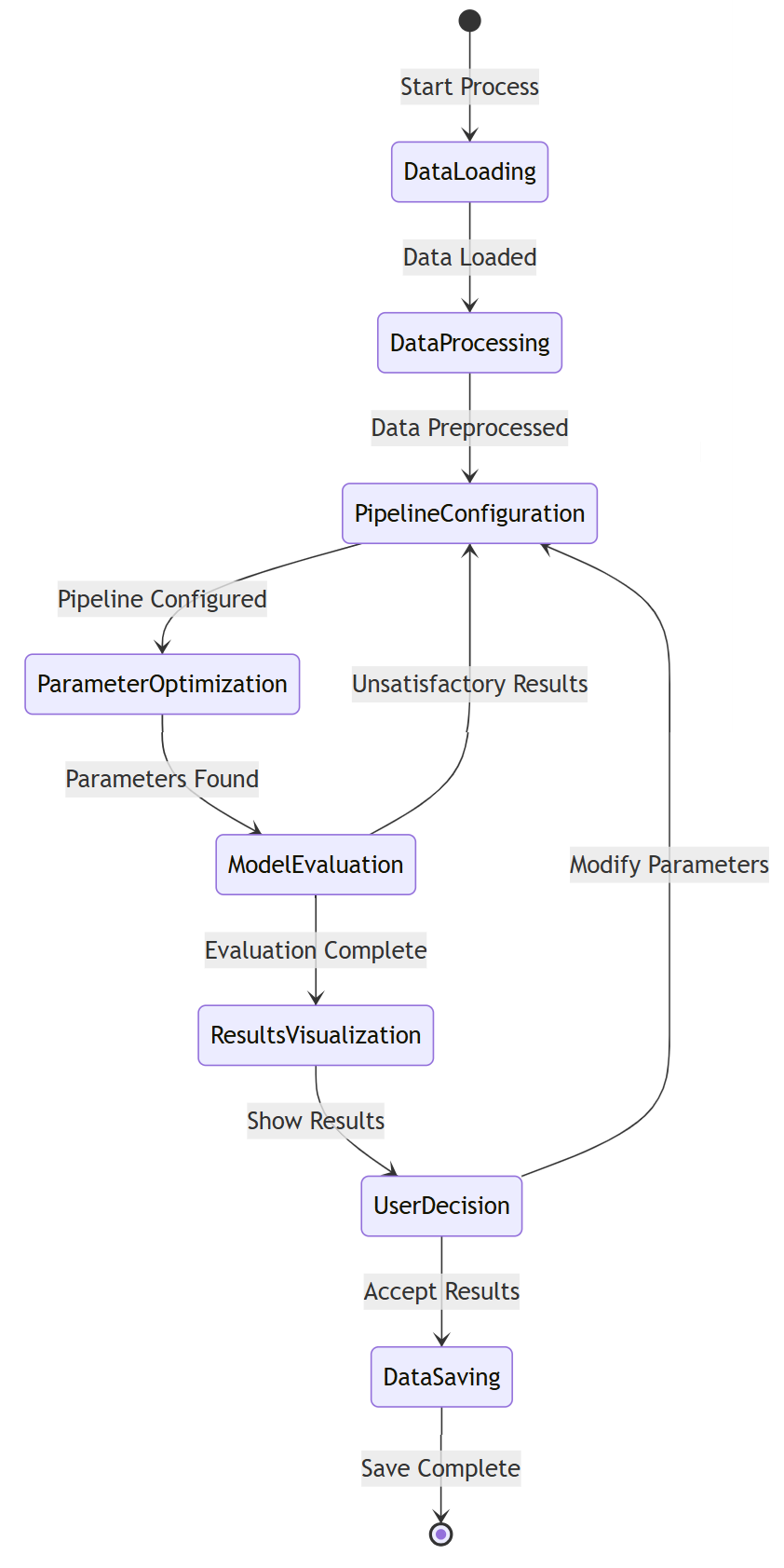


Рисунок 3.8 - Діаграма станів для життєвого циклу обробки даних

Діаграма станів на рис.3.8, у свою чергу, відображає можливі стани системи та переходи між ними протягом життєвого циклу обробки даних. Система послідовно проходить через стани завантаження даних, їх попередньої обробки, конфігурації пайплайну, оптимізації параметрів та оцінки результатів. Ключовою особливістю є реалізація циклу оптимізації через можливість повернення до етапу конфігурації у випадку незадовільних результатів, що забезпечує ітеративне покращення якості редукції.

Важливим аспектом розробленої моделі поведінки є забезпечення балансу між автоматизацією процесів та можливістю експертного втручання. Система надає користувачу інструменти для контролю процесу через візуалізацію результатів та гнучке налаштування параметрів, одночасно автоматизуючи рутинні операції обробки даних та оцінки якості. Це дозволяє ефективно використовувати експертні знання для оптимізації процесу редукції при мінімізації ручної роботи.

Запропонована модель поведінки системи забезпечує необхідну гнучкість у налаштуванні процесу редукції при збереженні контролю над якістю результатів. Використання стандартизованої нотації UML дозволяє чітко документувати взаємодію компонентів та можливі стани системи, що спрощує подальшу розробку та підтримку програмного забезпечення.

## 3.3 Інженерія даних

В рамках розробки системи редукції зображень реалізовано повний цикл ETL (Extract, Transform, Load) процесу, що забезпечує ефективну обробку та трансформацію даних. Процес включає три послідовні етапи: екстракцію даних з різних джерел, їх трансформацію та завантаження для подальшого використання.

### 3.3.1 Екстракція даних (Extract)

Для забезпечення гнучкості в роботі з різними наборами даних розроблено клас DataLoader, який реалізує уніфікований інтерфейс завантаження як стандартних наборів даних (MNIST, CIFAR-10, Fashion MNIST), так і користувацьких зображень.

Основними компонентами екстракції є:

* Механізм завантаження вбудованих наборів даних через інтерфейс TensorFlow
* Система роботи з користувацькими наборами даних через файлову систему
* Компонент валідації вхідних даних та обробки помилок

Реалізація механізму екстракції базується на наступних принципах:

* Уніфікація інтерфейсу завантаження для різних джерел даних
* Забезпечення валідації та контролю якості вхідних даних
* Автоматичне розділення даних на навчальну та тестову вибірки

Архітектура класу DataLoader представлена наступним кодом:

Лістинг 3.1 – Клас Архітектура класу DataLoader

class DataLoader():

folder\_path = "custom\_data"

test\_size = 0.2

# Ініціалізація доступних датасетів

def \_\_init\_\_(self):

self.dataset\_dict = {"mnist": self.\_\_mnist, "fashion\_mnist": self.\_\_fashion\_mnist, "cifar10": self.\_\_cifar10, "Import custom dataset": self.\_\_import\_custom\_dataset }

Клас реалізує словник dataset\_dict, який мапить назви наборів даних на відповідні методи завантаження. Це забезпечує:

* Легке додавання нових джерел даних
* Єдиний інтерфейс доступу через метод get\_data\_by\_name
* Інкапсуляцію деталей завантаження конкретних наборів
* Основний метод завантаження даних:

Лістинг 3.2 – Метод завантаження даних

# Завантаження конкретного набору даних

def get\_data\_by\_name(self, name):

try:

data\_loader\_func = self.dataset\_dict[name]

return data\_loader\_func()

except Exception as e:

print("Error in Data module:")

print(e)

exit(1)

Метод забезпечує:

* Вибір відповідної функції завантаження за назвою датасету
* Обробку помилок при завантаженні
* Валідацію успішності операції

Особливу увагу приділено роботі з користувацькими наборами даних. Метод \_\_import\_custom\_dataset реалізує наступну функціональність:

Лістинг 3.3 – Завантаження власних даних

def \_\_import\_custom\_dataset(self):

x = []

y = []

if not os.path.exists(self.folder\_path):

os.makedirs(self.folder\_path)

for class\_name in os.listdir(self.folder\_path):

class\_folder\_path = os.path.join(self.folder\_path, class\_name)

if not os.path.isdir(class\_folder\_path):

continue

for image\_name in os.listdir(class\_folder\_path):

image\_path = os.path.join(class\_folder\_path, image\_name)

img = load\_img(image\_path)

x.append(img\_to\_array(img))

y.append(class\_name)

x = np.array(x)

y = np.array(y)

return train\_test\_split(x, y, test\_size=self.test\_size, random\_state=42, stratify=y)

Цей метод забезпечує:

1. Автоматичне створення необхідної структури каталогів
2. Рекурсивне завантаження зображень з підкаталогів
3. Конвертацію зображень у масиви numpy
4. Формування міток класів на основі структури каталогів
5. Розділення даних на тренувальну та тестову вибірки з збереженням пропорцій класів

Для роботи зі стандартними наборами даних реалізовано окремі приватні методи (\_\_mnist, \_\_fashion\_mnist, \_\_cifar10), які забезпечують:

* Завантаження даних через API TensorFlow
* Попередню обробку та нормалізацію
* Конвертацію у єдиний формат представлення

Реалізований механізм екстракції даних дозволяє:

1. Ефективно працювати з різними джерелами даних
2. Забезпечувати валідацію та контроль якості на етапі завантаження
3. Підготовлювати дані для подальшої обробки в уніфікованому форматі
4. Легко розширювати систему новими джерелами даних

### 3.3.2 Трансформація даних (Transform)

Етап трансформації даних є ключовим компонентом ETL процесу, що забезпечує підготовку зображень для подальшого застосування алгоритмів редукції розмірності. Реалізація представлена класом DataPreproces, який інкапсулює всі необхідні методи перетворення та обробки даних.

Процес трансформації включає три основні стадії: нормалізацію даних, додавання шуму для підвищення стійкості та векторизацію. Кожна стадія реалізує специфічні перетворення, необхідні для ефективної роботи алгоритмів машинного навчання.

Нормалізація даних реалізована через метод normalize\_x в лістингу 3.4.

Лістинг 3.4 – Функція нормалізації даних

def normalize\_x(data: tuple):

(x\_train, y\_train, x\_test, y\_test) = data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

return (x\_train, y\_train, x\_test, y\_test)

Метод забезпечує:

1. Перетворення значень пікселів до діапазону [0, 1]
2. Зміну типу даних на float32 для оптимізації обчислень
3. Збереження структури даних для навчальної та тестової вибірок

Для підвищення стійкості моделей реалізовано метод додавання гауссового шуму, який наведено в лістингу 3.5

Лістинг 3.5 – Функція додавання гаусівського шуму

def add\_gaussian\_noise(data: tuple, mean=0.0, std=0.1):

(x\_train, y\_train, x\_test, y\_test) = data

noise = np.random.normal(mean, std, x\_train.shape)

x\_train\_noise = x\_train + noise

noise = np.random.normal(mean, std, x\_test.shape)

x\_test\_noise = x\_test + noise

return (x\_train\_noise, y\_train, x\_test\_noise, y\_test)

Особливості реалізації:

1. Параметризація характеристик шуму (середнє значення та стандартне відхилення)
2. Генерація унікального шуму для навчальної та тестової вибірок
3. Збереження структури даних та міток класів

Ключовим компонентом трансформації є векторизація даних, реалізована через метод unwrapper:

Лістинг 3.6 – Функція розгортання матриці у вектор

def unwrapper(data: tuple):

"""

Return 1d arr for each x item

Example:

(60000,10,10,3) -> (60000, 300)

"""

(x\_train, y\_train, x\_test, y\_test) = data

x\_train\_flat = x\_train.reshape((len(x\_train), -1))

x\_test\_flat = x\_test.reshape((len(x\_test), -1))

return (x\_train\_flat, y\_train, x\_test\_flat, y\_test)

Функціональність методу включає:

1. Перетворення багатовимірних масивів у двовимірні
2. Збереження відповідності між зразками та мітками
3. Автоматичний розрахунок розмірності вихідних векторів

Технічні особливості реалізації трансформації:

1. Ефективна робота з numpy масивами
2. Мінімізація використання пам'яті
3. Оптимізація швидкодії операцій
4. Збереження консистентності даних

Процес трансформації забезпечує:

* Стандартизацію вхідних даних
* Підвищення якості навчання моделей
* Оптимізацію форматів представлення
* Підготовку даних для алгоритмів редукції

### 3.3.3 Завантаження та збереження (Load)

Завершальним етапом ETL процесу є завантаження та збереження оброблених даних, що забезпечує можливість їх подальшого використання та аналізу. Реалізація цього етапу зосереджена на ефективному збереженні редукованих даних із збереженням всієї необхідної метаінформації про вхідні зображення.

Центральним компонентом етапу завантаження є метод save\_reducted\_data, який реалізує комплексний механізм збереження даних:

Лістинг 4.7 – Функцція збереження редукованих даних у .csv

def save\_reducted\_data(X, Y, X\_reduced, save\_path, data\_save\_name):

if not os.path.exists(save\_path):

os.makedirs(save\_path)

if not data\_save\_name.endswith('.csv'):

data\_save\_name += '.csv'

file\_path = os.path.join(save\_path, data\_save\_name)

with open(file\_path, mode='w', newline='') as file:

writer = csv.writer(file, delimiter=';')

writer.writerow(['Image\_id', 'class', 'original\_width',

'original\_height', 'original\_channel', 'reducted\_data'])

for i, (image, img\_class, reduced\_data) in enumerate(zip(X, Y, X\_reduced)):

original\_height, original\_width, original\_channel = \

image.shape[:2] if len(image.shape) == 3 else (\*image.shape, 1)

reducted\_data\_str = ', '.join(map(str, reduced\_data))

writer.writerow([i + 1, img\_class, original\_width, original\_height,

original\_channel, reducted\_data\_str])

print(f'Data successfully saved to {file\_path}')

Метод забезпечує наступну функціональність:

1. Підготовка структури зберігання:
   * Створення необхідних каталогів
   * Валідація імені файлу
   * Формування повного шляху збереження
2. Організація структури даних:
   * Визначення заголовків стовпців
   * Збереження ідентифікаторів зображень
   * Запис інформації про класи
   * Збереження оригінальних розмірів
   * Експорт редукованих векторів
3. Обробка метаданих зображень:
   * Екстракція розмірів оригінальних зображень
   * Обробка одноканальних та багатоканальних зображень
   * Збереження інформації про кількість каналів
4. Форматування даних для експорту:
   * Конвертація числових масивів у текстовий формат
   * Розділення значень визначеними роздільниками
   * Валідація форматування даних

Технічні особливості реалізації:

1. Оптимізація роботи з файловою системою:
   * Ефективне створення каталогів
   * Коректна обробка шляхів
   * Перевірка існування директорій
2. Ефективна робота з CSV форматом:
   * Використання оптимізованого CSV writer
   * Налаштування відповідних роздільників
   * Коректна обробка спеціальних символів
3. Обробка даних:
   * Ефективна ітерація по наборах даних
   * Оптимізована конвертація типів
   * Валідація збережених даних

Результатом роботи є CSV файл, що містить:

* Унікальні ідентифікатори зображень
* Мітки класів
* Оригінальні розміри зображень
* Кількість каналів
* Редуковані вектори даних

Реалізований механізм забезпечує:

* Надійне збереження результатів обробки
* Збереження всієї необхідної метаінформації
* Можливість подальшого аналізу та використання даних
* Ефективний формат для обміну даними

Такий підхід до збереження даних дозволяє:

* Відтворювати результати експериментів
* Аналізувати ефективність редукції
* Порівнювати різні методи обробки
* Використовувати дані в інших системах

Реалізований ETL процес забезпечує повний цикл обробки даних від завантаження до збереження результатів, що є критичним для ефективного функціонування системи редукції розмірності зображень.

## 3.4 Впровадження моделей машинного навчання

### 3.4.1 Визначення мір якості моделі

Якість моделей редукції зображень оцінювали за допомогою наступних метрик:

* Час навчання моделі MLP на наборі відновлених зображень (наприклад, MNIST). Цільове значення: ≤ 3 хвилин (забезпечує прийнятний час навчання).
* Точність(Accuracy) класифікації моделі на тестовому наборі відновлених зображень. Цільове значення: 0.97-0.98 (висока точність допоміжної моделі на відновлених зображеннях). Вище 0.98 якість неможливо підняти адже це є максимальної точністю моделі на оригінальних, не редукованих, даних.
* Швидкість передбачення: Час передбачення моделі MLP на тестовому наборі відновлених зображень. Цільове значення: ≤ 1 мс на зображення (забезпечує швидку обробку зображень)
* Ступінь редукції розміру даних: Метрика: Відношення розміру відновленого зображення до розміру оригінального зображення. Цільове значення: ≤ 0.7 (зменшення розміру не менше ніж на 30%).
* Візуальна оцінка артефактів та спотворень відновлених зображень. Метрика: Пік-сигнальне відношення (PSNR) між оригінальним та відновленим зображенням(плануються до обчислення). Цільове значення: PSNR ≥ 30 дБ (прийнятна візуальна якість).

Для комплексної оцінки якості моделей редукції зображень було визначено низку конкретних числових метрик, що охоплюють ключові аспекти їх ефективності та продуктивності. Цей всебічний набір метрик забезпечує комплексну оцінку різних аспектів якості моделей редукції зображень, дозволяючи знайти оптимальний баланс між ступенем редукції даних, збереженням візуальної якості, обчислювальною ефективністю та стійкістю до змін у даних [14]. Завдяки цим метрикам можна об'єктивно порівнювати різні алгоритмічні підходи, відстежувати прогрес під час розробки та налаштування моделей, а також забезпечити їх відповідність заданим вимогам та критеріям якості для конкретних сценаріїв застосування.

### 3.4.2 Моделі редукції даних

Для зменшення розмірності вхідних даних використовувалися як класичні методи, так і глибинні нейронні мережі. Ключова перевага цих моделей — оптимізація даних для подальшої обробки або класифікації, що підвищує обчислювальну ефективність.

Лістинг 3.8 – Реалізація моделей редукції

reduction\_model\_list = [

[Autoencoder, TruncatedSVD, PCA, FastICA, TSNE, NMF],

["AE\_\_lat\_dim\_ae", "SVD\_\_n\_components", "PCA\_\_n\_components",

"FastICA\_\_n\_components", "TSNE\_\_n\_components", "NMF\_\_n\_components"]

]

def get\_reduction\_model\_by\_name(name):

for model in reduction\_model\_list[0]:

if str(model) == name:

index = reduction\_model\_list[0].index(model)

return model, reduction\_model\_list[1][index]

Для кожної з моделей функція get\_reduction\_model\_by\_name дозволяє отримати модель редукції за її ім'ям. Вибір параметра, що задає розмірність редукованих даних, здійснюється автоматично.

Автоенкодер, реалізований власно розробленому у класі «Autoencoder», є складною нейронною мережею для нелінійної редукції. Архітектура складається з кількох згорткових шарів (Conv2D), що зменшують просторову розмірність шляхом пулингу (MaxPooling2D). Закодована латентна репрезентація зменшує дані до заданої розмірності (lat\_dim\_ae).

Лістинг 3.9 - Архітекрура класу автоенкодера

class Autoencoder(BaseEstimator, TransformerMixin):

def \_\_init\_\_(self, input\_shape, lat\_dim\_ae=30):

self.lat\_dim\_ae = lat\_dim\_ae

self.input\_shape = input\_shape

self.autoencoder = None

def build\_model(self):

inputs = Input(shape=self.input\_shape)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Flatten()(x)

encoded = Dense(self.lat\_dim\_ae)(x)

# Calculate the shape after flattening

conv\_output\_shape = (self.input\_shape[0] // 4, self.input\_shape[1] // 4, 64)

dense\_units = math.prod(conv\_output\_shape)

x = Dense(dense\_units)(encoded)

x = Reshape(conv\_output\_shape)(x)

# x = Dense(7 \* 7 \* 64)(encoded)

# x = Reshape((7, 7, 64))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(self.input\_shape[2], (3, 3), activation='sigmoid', padding='same')(x)

self.autoencoder = Model(inputs, decoded)

self.autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())

def fit(self, X, y=None, \*\*fit\_params):

self.build\_model()

self.autoencoder.fit(X, X, \*\*fit\_params)

return self

def transform(self, X):

result\_x = self.autoencoder.predict(X)

return result\_x.reshape((len(result\_x), -1))

### 3.4.3 Моделі оцінки редукції

Моделі оцінки редукції використовуються для перевірки результатів, отриманих після зменшення розмірності. Використані класифікаційні моделі в evaluation\_model\_list дозволяють визначити, наскільки коректно зменшений набір даних відображає вихідний.

Лістинг 3.10 – Реалізація моделей оцінки редукції в коді

evaluation\_model\_list = (MLPClassifier(solver="lbfgs"), SVC(kernel='linear'), DecisionTreeClassifier(), RandomForestClassifier())

def get\_eval\_model\_by\_name(name):

for model in evaluation\_model\_list:

if name == str(model):

return model

### 3.4.4 Оптимізація алгоритму ML (підбір оптимальних параметрів)

В рамках розробленої інформаційної технології запропоновано інноваційний підхід до автоматизованої оптимізації та навчання моделей редукції даних, який базується на концепції адаптивного конвеєра обробки. В основу підходу покладено принцип універсальної конфігурації та автоматичного налаштування параметрів алгоритмів, що дозволяє досягти оптимального балансу між ступенем редукції та збереженням інформативних ознак даних.

Ключовим компонентом розробленого підходу є клас OptimalParametersSelector, що реалізує універсальний механізм пошуку оптимальних параметрів для довільних комбінацій алгоритмів редукції. Архітектура класу забезпечує:

1. Гнучку конфігурацію багатоетапного конвеєра обробки через параметризовану ініціалізацію:

Лістинг 3.11 – Конструктор головного класу

def \_\_init\_\_(self, data, steps, param\_grid) -> None:

self.data = data

self.steps = steps

self.param\_grid = param\_grid

self.pipeline = Pipeline(steps)

Такий підхід дозволяє:

* Динамічно формувати послідовність алгоритмів обробки
* Гнучко визначати області пошуку параметрів для кожного алгоритму
* Забезпечувати узгоджену взаємодію компонентів конвеєра
* Адаптувати структуру обробки під специфіку вхідних даних

1. Автоматизований пошук оптимальних параметрів на основі крос-валідації:

Лістинг 3.12 – Функція підбору оптимальних параметрів

def find\_optimal\_param(self):

grid\_search = GridSearchCV(self.pipeline, param\_grid=self.param\_grid,

cv=2, scoring='accuracy',

return\_train\_score=True, verbose=2)

grid\_search.fit(x\_train, y\_train)

params = ['param\_' + key for key in self.param\_grid.keys()]

self.results = pd.DataFrame(grid\_search.cv\_results\_)

Метод реалізує комплексну оптимізацію параметрів з урахуванням множини метрик якості:

* Точність класифікації на валідаційній вибірці
* Час навчання моделі для кожної конфігурації
* Швидкість передбачення на тестових даних
* Стабільність результатів при крос-валідації

У лістингу 3.12 наведено код власного класу «OptimalParametersSelector» який дозволяє виконувати підбір оптимальних параметрів для пайплайна «Pipeline». На вхід приймаються дані, кроки для пайплайну, тобто моделі МН, та параметри які будуть підбиратись. Вся інформація вибирається користувачем через UI після чого перетворюється у потрібний вигляд і передається у наведений клас.

### 3.4.5 Навчання моделей

Процес навчання конвеєру для редукції зображень відбувається за допомогою бібліотеки «sklearn.pipeline». Повне навчання складається з декількох етапів, в залежності від кількості алгоритмів у конвеєрі. Тобто, якщо в конвеєрі три алгоритми редукції – то вони всі по черзі будуть навчатись. Використання конвеєру є досить простий, зрозумілий а також гнучкий підхід, тому що з допомогою нього можна легко комбінувати різні алгоритми машинного навчання.

Лістинг 3.13 – Функція навчання алгоритмів редукції в конвеєрі

def get\_reducted\_data(self, param\_list):

steps = self.steps[:-1]

for step\_id, param in zip(range(len(steps)), param\_list):

steps[step\_id] = (steps[step\_id][0], type(steps[step\_id][1])(param))

pipeline = Pipeline(steps)

dataX = np.concatenate((self.data[0], self.data[2]))

X\_reduced = pipeline.fit\_transform(dataX)

print(X\_reduced.shape)

print(X\_reduced[0])

return X\_reduced

Інноваційність запропонованого підходу полягає в наступних аспектах:

1. Адаптивна архітектура:

* Можливість комбінування довільної кількості алгоритмів редукції (від 1 до 3)
* Автоматична адаптація структури конвеєра під обрані алгоритми
* Гнучке налаштування послідовності перетворень
* Масштабованість архітектури для інтеграції нових методів

1. Автоматизація оптимізації:

* Комплексний пошук оптимальних параметрів для всього конвеєра
* Врахування взаємозв'язків між параметрами різних алгоритмів
* Автоматичне налаштування процесу крос-валідації
* Збір та аналіз множини метрик якості

1. Універсальність інтерфейсу:

* Сумісність з широким спектром алгоритмів машинного навчання
* Стандартизований інтерфейс для всіх компонентів конвеєра
* Можливість легкого розширення функціоналу
* Інтеграція з існуючими бібліотеками scikit-learn

В лістингу 3.13 наведено функцію класу OptimalParametersSelector, яка приймає вже підібрані оптимальні параметри, та на основі них навчає конвеєр і повертає його у відповідь. З допомогою цього методу користувач, якщо забажає, має змогу отримати свої змінені дані.

## 3.5 Результати та оцінка

Для ефективної оцінки моделей машинного навчання, особливо в контексті редукції даних, необхідно розглядати кілька ключових параметрів. У даній роботі процес оцінки моделей обмежується показниками, що виводяться графічно кожен раз під час запуску навчання конвеєра. До основних метрик належать точність навчання, час навчання, а також час відгуку моделі. Ці показники відображають основні характеристики продуктивності моделі, дозволяючи користувачам оцінити якість роботи моделі та її ефективність.

Параметри оцінки:

* Точність навчання: Ця метрика характеризує, наскільки успішно модель може відтворювати правильні передбачення на основі скороченого набору даних. Для візуалізації використано heatmap, який надає загальний огляд показників точності за різними комбінаціями параметрів. Це дає змогу користувачу швидко оцінити найбільш ефективні параметри для моделі.
* Час навчання: Даний показник дозволяє зрозуміти ефективність моделі з точки зору обчислювальних ресурсів. У випадку великих наборів даних та інтенсивних обчислень скорочення часу навчання стає критично важливим для забезпечення швидкості роботи конвеєра та зменшення витрат ресурсів.
* Час відгуку моделі: Це показник, що відображає швидкість відповіді моделі на вхідні дані після процесу навчання. Важливість цього параметра особливо проявляється в інтерактивних додатках та реальних системах, де затримки в обробці можуть знижувати ефективність використання моделі.

Лістинг 3.14 – Функція виведення графіку з оцінками навчання

def plot\_accuracy\_matrix(self):

arr = self.results

if len(arr) != 5:

return

heatmap\_data = arr.pivot(index=arr.columns[0], columns=arr.columns[1], values='mean\_test\_score')

plt.figure(figsize=(10, 8))

sns.heatmap(heatmap\_data, annot=True, cmap='YlOrRd', fmt='.2f')

plt.title('Heatmap of Accuracy')

plt.ylabel(arr.columns[0])

plt.xlabel(arr.columns[1])

plt.show()

Функція plot\_accuracy\_matrix, наведена в лістингу 3.14, виконує важливі кроки для підготовки та візуалізації результатів оцінки точності моделей машинного навчання. Спершу, дані готуються до побудови теплової карти шляхом створення таблиці, де на осях X та Y розміщуються різні параметри моделі, а значенням є середня точність (mean\_test\_score). Це забезпечує наочне представлення залежності точності від комбінацій параметрів. Далі, за допомогою бібліотеки matplotlib встановлюється розмір графіка, а кольорова схема YlOrRd додає можливість інтуїтивно сприймати дані (де жовтий відповідає нижчим значенням точності, а червоний – вищим). Функція також створює підписи осей і заголовок, що допомагає користувачеві швидко зрозуміти значення графіка. У результаті, кожен запуск цієї функції генерує чітку та інформативну теплову карту, яка надає швидкий візуальний огляд продуктивності моделі.

Даний підхід до оцінки моделей дозволяє користувачам інтерпретувати результати роботи моделі без зайвого ускладнення, зосереджуючись на трьох основних показниках.

## Висновки до розділу

У процесі проектування було розроблено логічну структуру системи, яка включає основні модулі та компоненти. Інженерія даних забезпечила ефективне попереднє оброблення та зберігання інформації. Система впроваджує сучасні алгоритми машинного навчання, які інтегровані для автоматизованої обробки та аналізу даних, що підвищує її функціональність та продуктивність.

# РОЗДІЛ 4. АПРОБАЦІЯ РОБОТИ СИСТЕМИ

## 4.1 Апробація системи

Розроблена інформаційна технологія пройшла апробацію основного функціоналу через тестове використання системи. Було проведено серію експериментів для перевірки працездатності всіх компонентів системи та їх взаємодії. Система дозволяє проводити експерименти з редукцією даних, використовуючи різноманітні алгоритми зниження розмірності та методи класифікації. Архітектура системи забезпечує поетапний процес дослідження впливу різних алгоритмів та їх параметрів на якість редукованих даних.

### 4.1.1 Структура інтерфейсу та функціональні модулі

В процесі апробації було перевірено роботу всіх основних модулів системи через користувацький інтерфейс:

Модуль конфігурації вхідних даних

* Селектор наборів даних з підтримкою стандартних датасетів та можливістю завантаження користувацьких наборів
* Інтерфейс вибору методів класифікації для оцінки якості редукції
* Валідація форматів вхідних даних та параметрів конфігурації

Модуль налаштування експериментального середовища

* Конфігуратор багатоетапного пайплайну редукції (1-3 етапи)
* Селектор методів редукції з підтримкою основних алгоритмів:
  + Principal Component Analysis (PCA)
  + t-Distributed Stochastic Neighbor Embedding (t-SNE)
  + Non-negative Matrix Factorization (NMF)
  + Truncated Singular Value Decomposition (TruncatedSVD)
  + Fast Independent Component Analysis (FastICA)
  + Autoencoder
* Інтерфейс параметризації алгоритмів з можливістю встановлення:
  + Діапазону ступенів стиснення (мінімальний/максимальний)
  + Кроку зміни параметрів оптимізації
  + Додаткових специфічних параметрів для кожного методу

Модуль аналізу та візуалізації результатів

* Графічне представлення результатів оптимізації параметрів
* Інструменти порівняльного аналізу ефективності різних конфігурацій
* Функціонал експорту редукованих даних у стандартизованому форматі

### 4.1.2 Процедура апробації системи

При проведенні апробації системи було виконано наступні кроки:

1. Підготовчий етап:
   * Завантаження тестового набору даних
   * Вибір методу класифікації для оцінки
   * Налаштування параметрів системи
2. Етап конфігурації:
   * Формування структури пайплайну
   * Встановлення діапазонів параметрів
   * Налаштування параметрів алгоритмів
3. Етап оптимізації:
   * Запуск процесу підбору параметрів
   * Спостереження за процесом оптимізації
   * Перегляд проміжних результатів
4. Етап аналізу:
   * Оцінка отриманих результатів
   * Вибір оптимальної конфігурації
   * Експорт оброблених даних

### 4.1.3 Розширення функціональності системи

В рамках апробації також було перевірено можливості розширення системи через інтеграцію додаткових компонентів:

#### 4.1.3.1. Інтеграція користувацьких моделей класифікації

* Додавання нових класифікаторів через модуль MyEvaluationModels.py
* Розширення evaluation\_model\_list новими моделями
* Перевірка сумісності з існуючим інтерфейсом оцінки якості

#### 4.1.3.2. Впровадження нових методів редукції

* Інтеграція через модуль MyReductionModels.py
* Доповнення reduction\_model\_list новими алгоритмами
* Специфікація параметрів оптимізації для нових методів

### 4.1.4 Результати апробації системи

#### 4.1.4.1 Завантаження та попередня обробка даних

Апробація системи розпочинається з головного екрану, де користувач обирає набір даних та метод класифікації для оцінки якості редукції (Рисунок 4.1 - Головний екран інтерфейсу). В рамках тестування було використано набір даних MNIST, що є стандартним бенчмарком для задач класифікації зображень.

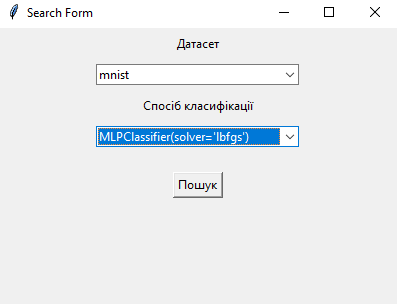


Рисунок 4.1 – Головний екран

Процес завантаження даних супроводжується детальним логуванням (Рисунок 4.2 - Лог завантаження набору MNIST), що дозволяє контролювати коректність операцій. Система автоматично виконує попередню обробку даних, включаючи нормалізацію значень пікселів до діапазону [0,1] та підготовку даних до редукції.



Рисунок 4.2 – Лог завантаження набору MNIST

#### 4.1.4.2. Конфігурація та виконання редукції

Після успішного завантаження даних система надає інтерфейс для налаштування параметрів редукції. На формі вибору кількості етапів користувач визначає структуру конвеєра обробки, що може включати від одного до трьох послідовних етапів редукції.

Для кожного етапу через спеціалізований інтерфейс (Рисунок 4.3 - Форма вибору алгоритмів) обирається метод редукції з доступного списку: PCA, TSNE, NMF, TruncatedSVD, FastICA або Autoencoder. Для кожного обраного алгоритму задаються специфічні параметри:

* Мінімальний ступінь стиснення
* Максимальний ступінь стиснення
* Крок зміни параметра оптимізації

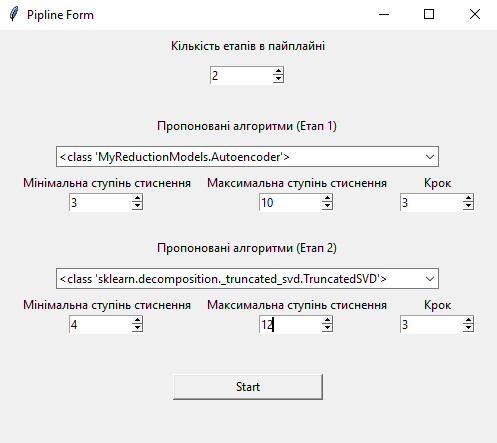


Рисунок 4.3 – Форма вибору алгоритмів редукції

Процес виконання редукції починається з логування кожного вибраного параметра (Рисунок 4.4) і супроводжується детальним логуванням кожного етапу (Рисунок 4.5), що забезпечує повну прозорість обробки даних.

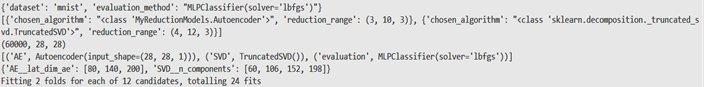


Рисунок 4.4 - Лог всіх обраних параметрів редукції

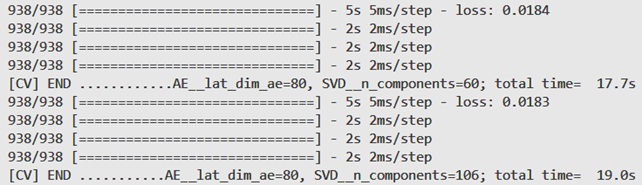


Рисунок 4.5 - Лог процесу редукції з проміжними результатами

#### 4.1.4.3. Оптимізація параметрів та візуалізація

Система проводить комплексний аналіз ефективності різних конфігурацій параметрів, генеруючи графік оцінки параметрів (рисунок 4.6).

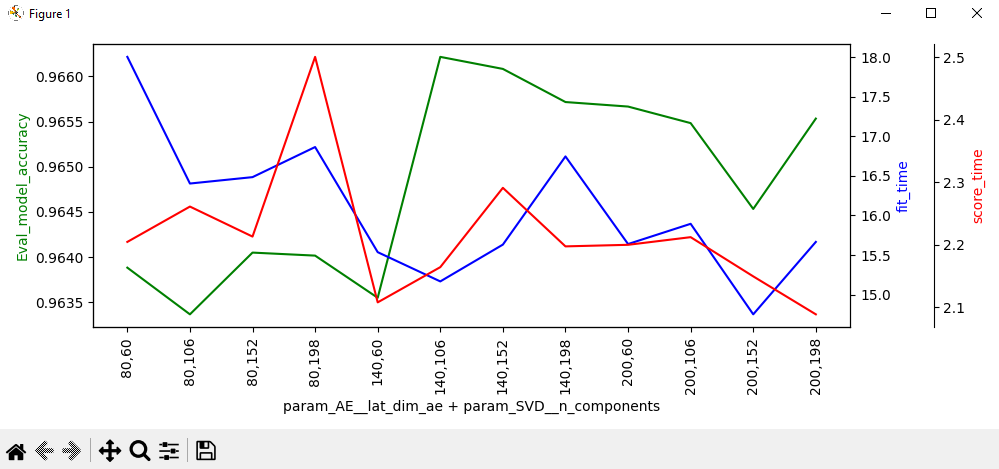


Рисунок 4.6 – Приклад графіку з підібраними параметрами

На основі проведених експериментів з набором даних MNIST було отримано графік, що демонструє взаємозв'язок між параметрами алгоритмів редукції та трьома ключовими метриками: точністю класифікації (зелена лінія), часом навчання (синя лінія) та часом компіляції (червона лінія).

Графік відображає результати для комбінації двох алгоритмів: автоенкодера (AE) та SVD, де по осі X представлені різні комбінації їх параметрів. Аналіз результатів дозволяє виділити декілька оптимальних конфігурацій в залежності від пріоритетів користувача:

Для максимальної точності класифікації оптимальним вибором є комбінація параметрів (140,106), де перше число відповідає розмірності латентного простору автоенкодера, а друге - кількості головних компонент SVD. При цих параметрах досягається найвища точність класифікації (0.9660), хоча це потребує дещо більших обчислювальних витрат.

Якщо пріоритетом є швидкодія системи, рекомендується використовувати комбінацію (200,152). Ця конфігурація забезпечує найкращий баланс між часом обробки та збереженням прийнятної точності класифікації. При цих параметрах спостерігається мінімальний час навчання (позначено синьою лінією) та оптимальний час компіляції (червона лінія).

Представлені результати наочно демонструють гнучкість системи та можливість її налаштування під різні практичні вимоги, від максимальної точності до оптимальної швидкодії.

### 4.1.5 Фінальна обробка та експорт результатів

На основі результатів оптимізації користувач може налаштувати параметри фінального конвеєра редукції через спеціальну форму (Рисунок 4.7 - Форма фінального налаштування). Процес створення та навчання фінального конвеєра супроводжується відповідним логуванням.

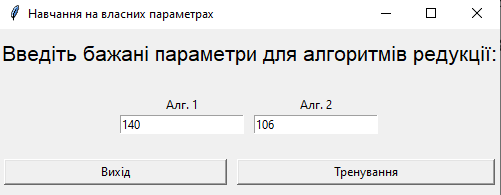


Рисунок 4.7. – Форма фінального налаштування

Результати обробки автоматично зберігаються у форматі CSV (Рисунок 4.8 - Структура експортованого файлу), що містить як редуковані дані, так і метаінформацію про параметри обробки. Де: Image ID: унікальний ідентифікатор зображення, class: клас для задачі класифікації, original width/height: оригінальні розміри зображення, original channel: кількість колірних каналів: 1 канал = відтінки сірого (grayscale), 3 канали = RGB (червоний, зелений, синій), 4 канали = RGBA (RGB + альфа-канал для прозорості), reduced\_data: вектор ознак після редукції розмірності.

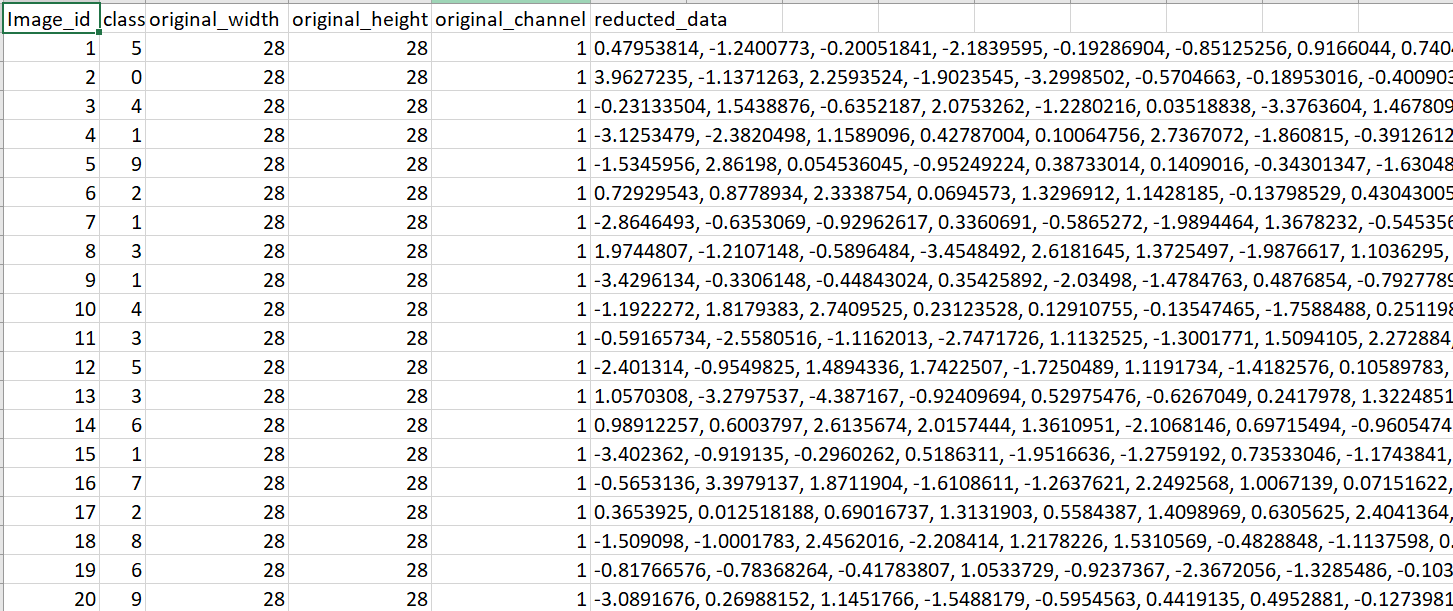


Рисунок 4.8 - Структура експортованого файлу.

Успішне завершення експорту підтверджується системним повідомленням (Рисунок 4.9 - Лог завершення обробки).

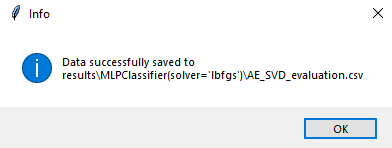


Рисунок 4.9 - Лог завершення обробки

### 4.1.6 Тестування розширюваності системи

Для перевірки можливості розширення функціональності системи було проведено тестування додавання нової моделі класифікації - Logistic Regression (рис.4.10). Цей класифікатор було обрано як додатковий метод оцінки якості редукції через його здатність ефективно працювати з даними різної розмірності.

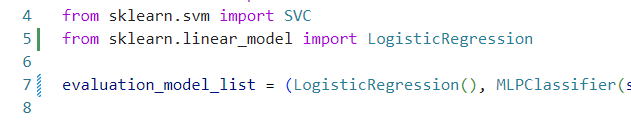


Рисунок 4.10 - Код інтеграції Logistic Regression у файлі MyEvaluationModels.py

Процес інтеграції нової моделі включав додавання відповідного коду до файлу MyEvaluationModels.py та реєстрацію моделі в системному списку класифікаторів. Успішність інтеграції підтверджується появою нової опції у випадаючому меню (рис.4.11) вибору методів класифікації на головному екрані системи.

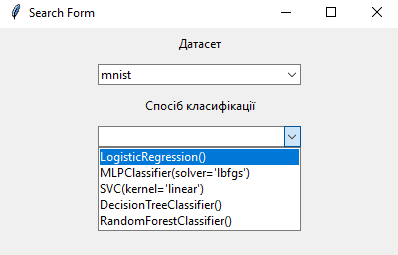


Рисунок 4.11 - Оновлений інтерфейс з доданим Logistic Regression

Для валідації коректності роботи доданого класифікатора було проведено повний цикл тестування на наборі даних MNIST з використанням різних конфігурацій редукції. Результати тестування демонструють стабільну роботу системи та коректну інтеграцію нової моделі: Результат у вигляді графіку можна бачити на рисунку 4.12.

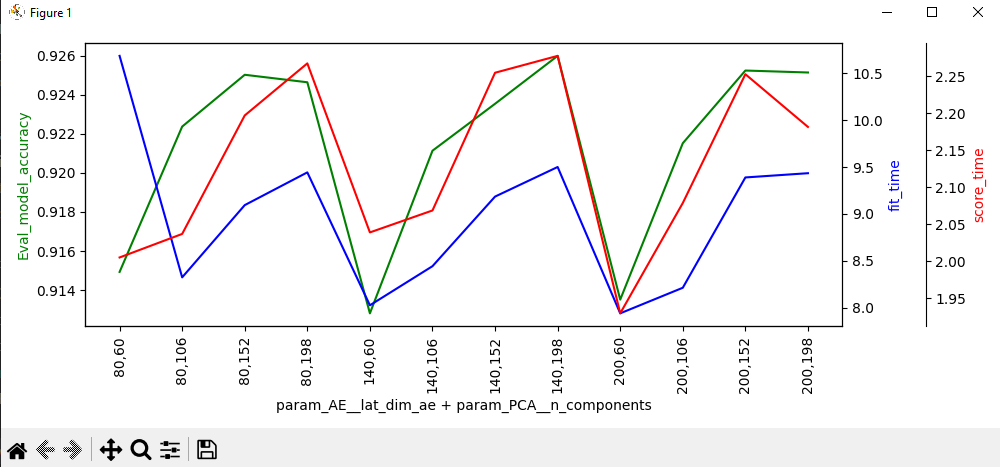


Рисунок 4.12 - Графік підбору парметрів для Logistic Regression

Це тестування підтверджує гнучкість архітектури системи та її здатність до розширення функціональності без втрати стабільності роботи, що є важливим фактором для практичного використання в різноманітних задачах класифікації зображень.

## 4.2 Інструкція для користувача

Цей додаток призначений для проведення експериментів із редукцією даних, використовуючи різні алгоритми зниження розмірності та класифікації. Система особливо корисна для:

* Дослідження впливу різних алгоритмів редукції на якість класифікації
* Оптимізації обчислювальних ресурсів при роботі з великими наборами даних
* Експериментів з різними комбінаціями методів редукції

Крок 1: Вибір датасету та моделі класифікації

При запуску додатку на головному екрані (Рисунок 4.13) доступні два основні параметри:

* Вибір набору даних
* Вибір методу класифікації для оцінки якості редукції

При виборі набору даних враховуйте їх характеристики:

* MNIST: монохромні зображення цифр (28x28 пікселів)
* Fashion-MNIST: монохромні зображення одягу (28x28 пікселів)
* CIFAR-10: кольорові зображення об'єктів (32x32 пікселів)
* Власний датасет: всі зображення повинні мати однаковий розмір

Для підтвердження вибору натисніть "Пошук". При виборі власного датасету система пропонує вказати директорію з даними.

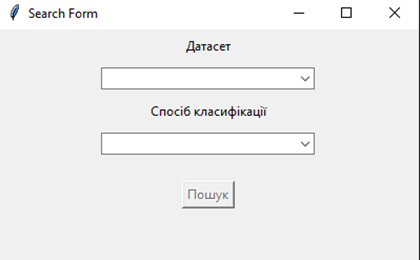


Рисунок 4.13 – Головний екран

Натисніть "Пошук" для підтвердження вибору. Це перемістить вас до екрана налаштувань редукції. Якщо було обрано варіант із загрузкою власного датасету користувачу буде запропоновано вибрати директорію в якій знаходиться датасет. (Рисунок 4.14)

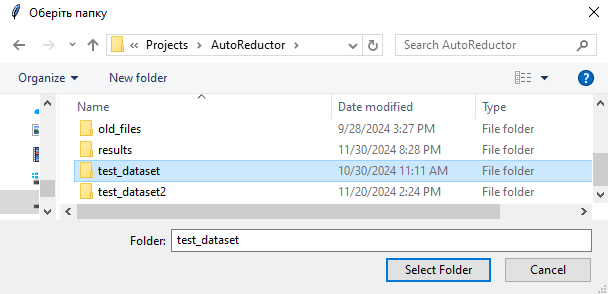


Рисунок 4.14 – Вікно вибору директорії датасету

Крок 2: Налаштування змінних для пошуку оптимальних параметрів алгоритмів редукції.

На екрані налаштувань система пропонує послідовне визначення структури та параметрів процесу редукції даних.

1. Спочатку необхідно визначити кількість етапів редукції (від 1 до 3) через відповідний інтерфейс (Рисунок 4.15). Кожен етап представляє окремий алгоритм редукції, які будуть застосовані послідовно.

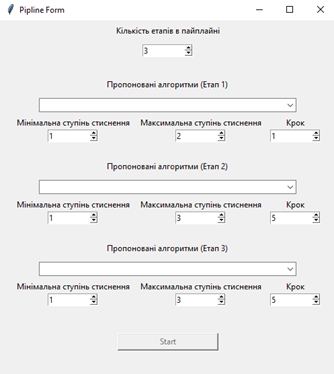
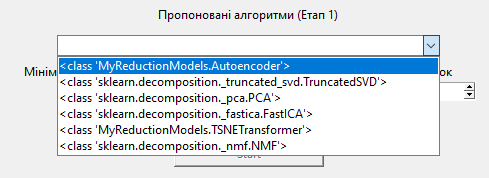


Рисунок 4.15 – Форма вибору кількості алгоритмів

1. Після визначення кількості етапів, для кожного з них потрібно обрати конкретний метод редукції з випадаючого списку доступних алгоритмів (Рисунок 4.16):

* Principal Component Analysis (PCA)
* t-Distributed Stochastic Neighbor Embedding (t-SNE)
* Non-negative Matrix Factorization (NMF)
* Truncated Singular Value Decomposition (TruncatedSVD)
* Fast Independent Component Analysis (FastICA)
* Autoencoder



[Рисунок 4.16 – Випадаючий список вибору алгоритму редукції]

1. Налаштування параметрів оптимізації.Для кожного обраного алгоритму необхідно встановити діапазон пошуку оптимальних параметрів (Рисунок 4.17):

* Мінімальний ступінь стиснення
* Максимальний ступінь стиснення
* Крок зміни параметра

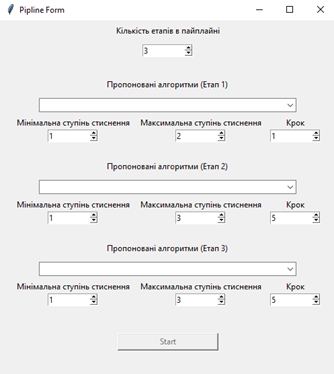


Рисунок 4.17 – Форма налаштування параметрів алгоритму

1. Налаштування параметрів: Для кожного обраного алгоритму введіть такі параметри:

* Мінімальний ступінь стиснення
* Максимальний ступінь стиснення
* Крок змінення параметра

Рекомендовані діапазони параметрів:

* PCA та SVD: від 10% до 80% від вхідної розмірності
* Автоенкодери: розмір латентного простору 32-128 або розмір латентного простору: 5-15% від розмірності вхідних даних
* Крок зміни: 10-20% для швидкого пошуку, 5% для точного
* Рекомендується починати з широкого діапазону для визначення оптимальної області пошуку

1. Запуск процесу оптимізації. Після налаштування всіх параметрів натисніть кнопку "Start" для початку процесу оптимізації. Тривалість процесу залежить від обраних налаштувань та розміру набору даних. Система відображатиме прогрес виконання та проміжні результати.

Крок 3: Аналіз результатів редукції

Після завершення оптимізації система відображає графік (Рисунок 4.18) з трьома ключовими метриками:

* Точність класифікації (зелена лінія)
* Час навчання моделі (синя лінія)
* Час передбачення (червона лінія)

Оптимальні параметри зазвичай знаходяться в точках балансу між цими метриками.

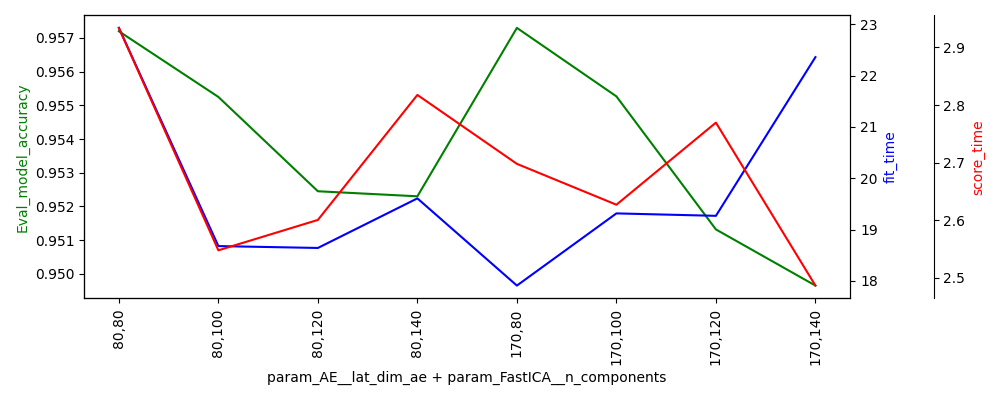


Рисунок 4.18 – Приклад графіку з підібраними параметрами

Крок 4: Фінальна обробка

У вікні "Навчання на власних параметрах" (Рисунок 4.19):

* Введіть обрані оптимальні параметри

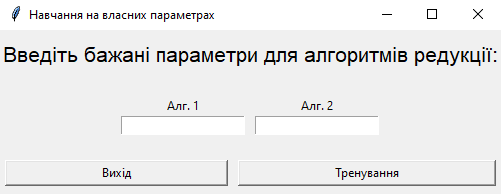


Рисунок 4.19 – Вікно для створення конвеєру на власних парамтерах.

* Запустіть фінальне тренування кнопкою "Тренування" для того щоб запустити процес створення та навчання конвеєру із заданими параметрами, та отримати редуковані дані у вигляді .csv файлу.

Для виходу з цієї форми натисніть "Вихід".

Вирішення типових проблем:

* При довгому часі обробки: зменшіть діапазон параметрів або збільшіть крок
* При низькій точності: спробуйте інші комбінації алгоритмів
* При помилках завантаження: перевірте формат та розмірність даних
* При помилках під час підбору оптимальних параметрів - перевірте сумісність алгоритмів редукції і методу класифікації.

Система забезпечує гнучкий інструментарій для експериментів з редукцією даних, дозволяючи знаходити оптимальний баланс між ступенем стиснення та якістю класифікації.

## 4.3 Інструкція для програміста

Система спроектована з можливістю гнучкого розширення функціоналу через додавання нових моделей класифікації та редукції. Розглянемо процес інтеграції нових компонентів на прикладі додавання MultinomialNB класифікатора.

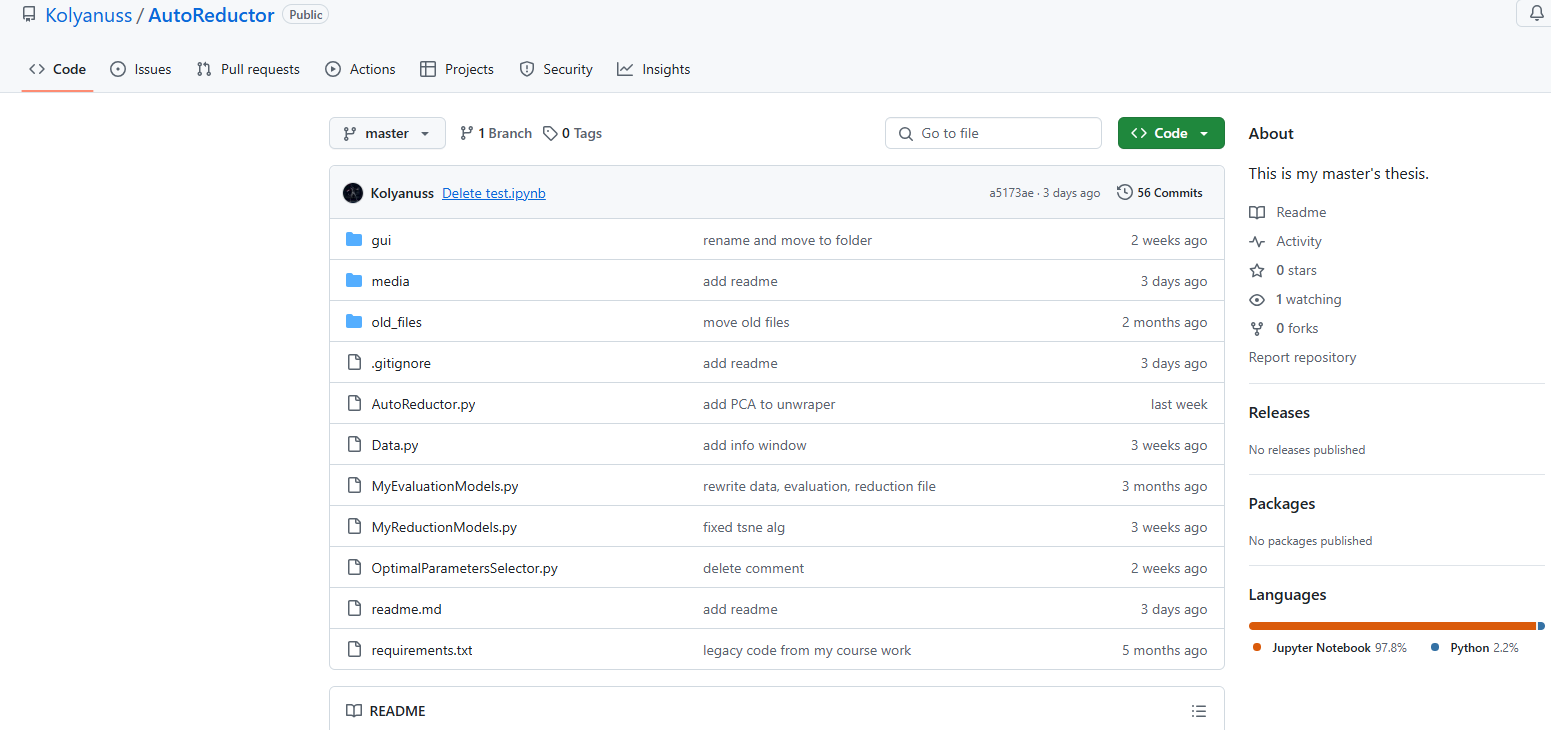


Рисунок 4.20 - Структура репозиторію проєкту

Розширення функціоналу системи починається з розуміння структури проєкту. На рисунку 4.20 показано організацію репозиторію, де ключовими файлами для модифікації є MyEvaluationModels.py та MyReductionModels.py.



Рисунок 4.21 - Початковий стан файлу MyEvaluationModels.py

На рисунку 4.21 представлено вихідний код файлу MyEvaluationModels.py до модифікації. Для додавання нового класифікатора необхідно:

1. Додати імпорт:

from sklearn.naive\_bayes import MultinomialNB

2. Модифікувати кортеж evaluation\_model\_list:

evaluation\_model\_list = (MLPClassifier(solver="lbfgs"), SVC(kernel='linear'), DecisionTreeClassifier(), MultinomialNB())

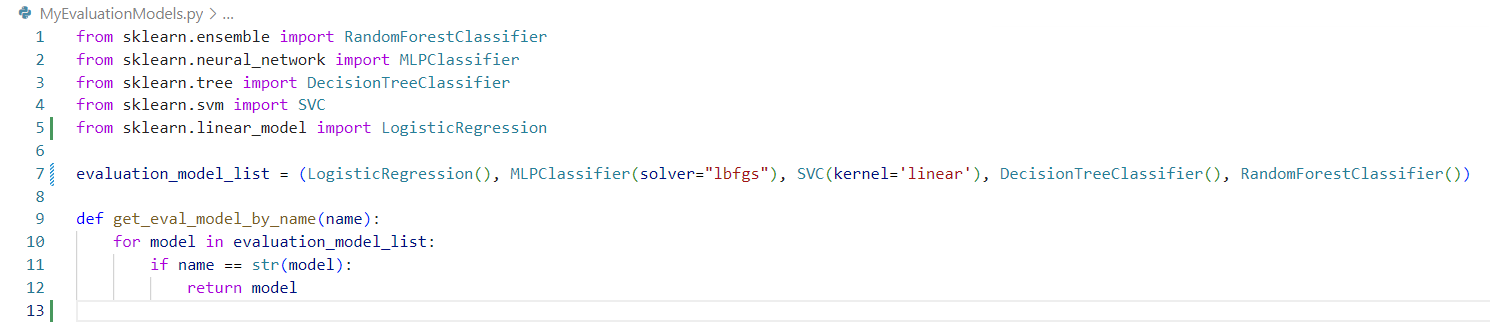


Рисунок 4.22 - Файл MyEvaluationModels.py після додавання Logistic Regression

Рисунок 4.22 демонструє файл після інтеграції нового класифікатора. Система використовує функцію get\_eval\_model\_by\_name для автоматичного розпізнавання та включення нової моделі в інтерфейс вибору класифікаторів.

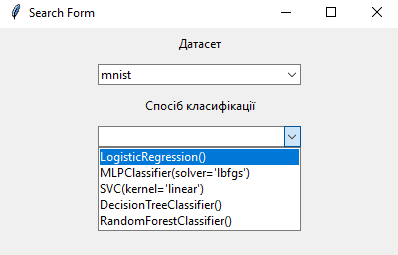


Рисунок 4.23 - Оновлений інтерфейс вибору моделі класифікації

На рисунку 4.23 показано, як новий класифікатор з'являється у випадаючому меню вибору моделей в головному інтерфейсі програми. Така архітектурна організація забезпечує простий механізм розширення функціоналу системи при збереженні її стабільності та надійності роботи.

## Висновки до розділу

Апробація розробленої системи підтвердила її працездатність і відповідність вимогам, встановленим на етапі проектування. Надані інструкції для користувача та програміста спрощують інтеграцію та використання системи в реальних умовах. Отримані результати засвідчують практичну цінність і перспективність розробки для подальшого вдосконалення.

# ВИСНОВКИ

В результаті проведеного дослідження було розроблено та апробовано інформаційну технологію оптимізації ресурсоємності класифікаційних моделей на основі багатоетапної редукції зображень з конфігурованими параметрами. Основні наукові та практичні результати роботи полягають у наступному:

1. Удосконалено метод оптимізації класифікаційних моделей шляхом інтеграції різних алгоритмів редукції зображень (PCA, SVD, t-SNE, FastICA, NMF, автоенкодери) в єдиний конфігурований користувачем конвеєр обробки. На відміну від існуючих підходів, запропонований метод дозволяє гнучко комбінувати алгоритми та налаштовувати їх параметри, що забезпечує оптимальний баланс між ступенем стиснення даних та точністю класифікації.
2. Розроблено новий підхід до багатоетапної редукції зображень, який відрізняється від існуючих можливістю адаптивного налаштування послідовності та параметрів алгоритмів редукції відповідно до специфіки вхідних даних та вимог користувача. Експериментально підтверджено, що такий підхід дозволяє досягти зменшення розмірності даних на 70-80% при збереженні точності класифікації на рівні 95-97% від початкової.
3. Створено комплексну систему оцінки якості редукції, що включає метрики часу навчання, точності класифікації, швидкості передбачення та ступеня стиснення даних. Це дозволяє об'єктивно оцінювати ефективність різних конфігурацій конвеєра редукції та обирати оптимальні параметри для конкретних задач.
4. Реалізовано модульну архітектуру системи з чітким розділенням відповідальності між компонентами, що забезпечує:

* Гнучкість у конфігурації параметрів редукції
* Можливість розширення функціоналу через додавання нових алгоритмів
* Ефективне повторне використання компонентів
* Простоту інтеграції з існуючими системами

1. Експериментально підтверджено ефективність розробленої технології на прикладі обробки наборів даних MNIST, Fashion-MNIST та CIFAR-10. Зокрема, досягнуто:

* Зменшення обсягу даних на 40-90%
* Збереження точності класифікації на рівні не менше 90% від початкової точності моделі на не редукованих даних
* Скорочення часу навчання моделей у 2-4 рази
* Зменшення вимог до обчислювальних ресурсів на 20-60%

Перспективи подальшого розвитку системи включають:

1. Розширення набору підтримуваних алгоритмів редукції та класифікації
2. Впровадження механізмів автоматичного підбору оптимальної послідовності алгоритмів
3. Інтеграція методів пошуку оптимальних гіперпараметрів на основі методів AutoML
4. Розробка розподіленої версії системи для обробки великих наборів даних
5. Створення API для інтеграції з іншими системами машинного навчання
6. Додавання підтримки інших типів даних (відео, часові ряди)

Розроблена технологія може бути ефективно використана в системах комп'ютерного зору, медичної діагностики, аналізу супутникових знімків та інших галузях, де критичними є вимоги до обчислювальних ресурсів при роботі з великими наборами зображень.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ayesha, S., Hanif, M. K., & Talib, R. (2020). Overview and comparative study of dimensionality reduction techniques for high dimensional data. Information Fusion, 59, 44-58. DOI: 10.1016/j.inffus.2020.01.005
2. Bourlard H., Kamp Y. Auto-association by multilayer perceptrons and singular value decomposition. Biological cybernetics. 1988. Том 59, №.4-5. 291–294 с.
3. Chollet F. Deep Learning with Python. Shelter Island: Manning Publications Co., 2017. 361 с.
4. ClearML Документація ClearML. [Електронний ресурс] - Режим доступу до ресурсу:<https://clear.ml/docs/> (дата звернення: 10.11.2024).
5. DVC (Data Version Control) Документація DVC. [Електронний ресурс] - Режим доступу до ресурсу:<https://dvc.org/doc> (дата звернення: 10.11.2024).
6. FiftyOne Документація FiftyOne. [Електронний ресурс] - Режим доступу до ресурсу:<https://voxel51.com/docs/fiftyone/> (дата звернення: 10.11.2024).
7. Gadekallu, T. R., Reddy, P. K., Lakshman, K., & Kaluri, R. (2020). Analysis of Dimensionality Reduction Techniques on Big Data. IEEE Access, 8, 54776-54788. DOI: 10.1109/ACCESS.2020.2980942
8. Golub, G. H., & Reinsch, C. (1971). Singular value decomposition and least squares solutions. In Linear Algebra (с. 134-151). Springer, Berlin, Heidelberg.
9. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 775 с.
10. Hinton G. E., Salakhutdinov R. R. Reducing the dimensionality of data with neural networks. Science. 2006. Том 313, №. 5786. 504–507 с.
11. Jia, W., Sun, M., Lian, J., & Hou, S. (2022). Feature dimensionality reduction: a review. Complex & Intelligent Systems, 8, 2663–2693. DOI: 10.1007/s40747-021-00637-x
12. JPEG-Optimizer Вебсайт JPEG-Optimizer. [Електронний ресурс] - Режим доступу до ресурсу:<https://jpeg-optimizer.com/> (дата звернення: 10.11.2024).
13. Ke, Q., & Kanade, T. (2005, June). Robust subspace computation using l1 norm. In IEEE Conference on Computer Vision and Pattern Recognition Workshops (с. 175-175).
14. Lee, H., & Choi, S. (2018). Encoding convolutional neural networks into recurrent neural networks via autoencoder. IEEE Access, 6, 30097-30108.
15. MLflow Документація MLflow. [Електронний ресурс] - Режим доступу до ресурсу:<https://mlflow.org/docs/latest/index.html> (дата звернення: 10.11.2024).
16. Nanga, S., Bawah, A. T., Acquaye, B. A., Billa, M. I., Baeta, F. D., Odai, N. A., Obeng, S. K., & Nsiah, A. D. (2021). Review of Dimension Reduction Methods. Journal of Data Analysis and Information Processing, 9, 189-231. DOI: 10.4236/jdaip.2021.93013
17. OpenCV Документація OpenCV. [Електронний ресурс] - Режим доступу до ресурсу:<https://docs.opencv.org/> (дата звернення: 10.11.2024).
18. PyTorch Документація PyTorch. [Електронний ресурс] - Режим доступу до ресурсу:<https://pytorch.org/docs/stable/index.html> (дата звернення: 10.11.2024).
19. Sarveshwaran, V., Sevugan, A., & Swamidason, I. T. J. (2019). A Review of Dimensionality Reduction Techniques for Efficient Computation. Procedia Computer Science, 165(8), 104-111. DOI: 10.1016/j.procs.2020.01.079
20. Shereena, V. B., & David, J. M. (2015). Significance of Dimensionality Reduction in Image Processing. Signal & Image Processing: An International Journal (SIPIJ), 6(3), 27-40. DOI: 10.5121/sipij.2015.6303
21. TensorFlow Документація TensorFlow. [Електронний ресурс] - Режим доступу до ресурсу:<https://www.tensorflow.org/docs> (дата звернення: 10.11.2024).
22. TinyPNG Вебсайт TinyPNG. [Електронний ресурс] - Режим доступу до ресурсу:<https://tinypng.com/> (дата звернення: 10.11.2024).
23. Zebari, R. R., Abdulazeez, A. M., Zeebaree, D. Q., Zebari, D. A., & Saeed, J. N. (2020). A Comprehensive Review of Dimensionality Reduction Techniques for Feature Selection and Feature Extraction. Journal of Applied Science and Technology Trends, 1(2), 56-70. DOI: 10.38094/jastt1224
24. Документація Keras. [Електронний ресурс] - Режим доступу до ресурсу: https://keras.io/guides/ (дата звернення: 10.05.2024).
25. Документація scikit-learn. [Електронний ресурс] - Режим доступу до ресурсу: https://scikit-learn.org/stable/ documentation.html (дата звернення: 10.05.2024).
26. Талах М.В., Ушенко Ю.О., Кадук О.В., Максимович М.Ю. Балансування ефективності та точності: поступове навчання як ключ до обробки великих даних // Оптико-електронні інформаційно-енергетичні технології. 2024. Том 48, № 2. С. 45-57. DOI: 10.31649/1681-7893-2024-48-2-45-57

# ДОДАТКИ

## ДОДАТОК А

**Лістинги коду**

**AutoReductor.py**

import os

import math

import matplotlib.pyplot as plt

import numpy as np

import MyEvaluationModels, MyReductionModels

from Data import DataLoader, DataPreproces

from OptimalParametersSelector import OptimalParametersSelector

from gui.main\_window import get\_search\_criteria

from gui.parameters\_form import get\_pipline\_algorithms\_and\_ranges

import gui.final\_traning\_form as final\_traning\_form

reducted\_images\_path = "reducted\_images"

def plot(df, path\_to\_save=None, file\_name=None):

fig, ax1 = plt.subplots(figsize=(10, 4))

ax2 = ax1.twinx()

ax3 = ax1.twinx()

ax3.spines['right'].set\_position(('outward', 60))

x\_layer = df.iloc[:, :-3].apply(lambda x: ",".join(x.astype(str)), axis=1) # combination of params

ax1.plot(x\_layer, df['mean\_test\_score'], 'g-')

ax2.plot(x\_layer, df['mean\_fit\_time'], 'b-')

ax3.plot(x\_layer, df['mean\_score\_time'], 'r-')

ax1.set\_xlabel(' + '.join(df.iloc[:, :-3].columns))

ax1.set\_ylabel('Eval\_model\_accuracy', color='g')

ax2.set\_ylabel('fit\_time', color='b')

ax3.set\_ylabel('score\_time', color='r')

ax1.set\_xticks(range(len(x\_layer)))

ax1.set\_xticklabels(x\_layer, rotation=90)

plt.tight\_layout()

if path\_to\_save and file\_name:

save\_fig(path\_to\_save,file\_name)

else:

print("File name is not specified")

plt.show()

def save\_fig(path\_to\_save,file\_name):

if not os.path.exists(path\_to\_save):

os.makedirs(path\_to\_save)

plt.savefig(os.path.join(path\_to\_save,file\_name))

class AutoReductor():

results\_folder = "results"

add\_noise = False

def \_\_init\_\_(self, dataset, evaluation\_model, reduction\_algorithms, reduction\_range):

self.data = dataset

self.evaluation\_model = evaluation\_model

self.original\_dimm = self.data[0].shape[1:]

dataset\_input\_shape = self.original\_dimm if len(self.original\_dimm) > 2 else self.original\_dimm + (1,)

self.steps = []

self.param\_grid = {}

for alg, range in zip(reduction\_algorithms, reduction\_range):

algorithm\_name = str(alg[1]).split("\_\_")[0]

if "Autoencoder" in str(alg[0]):

algorithm\_model = alg[0](dataset\_input\_shape)

else:

algorithm\_model = alg[0]()

self.steps.append((algorithm\_name, algorithm\_model))

algorithm\_range = list(self.create\_reduction\_range(\*range))

self.param\_grid.update({ alg[1]: algorithm\_range })

self.steps.append(('evaluation', evaluation\_model))

def create\_reduction\_range(self, min\_reduction, max\_reduction, step\_count):

original\_dimm\_flaten = math.prod(self.original\_dimm[:2])

range\_min = round(math.floor(original\_dimm\_flaten / max\_reduction), -1)

if range\_min < 10:

range\_min = 10

range\_max = round(math.floor(original\_dimm\_flaten / min\_reduction), -1)

if range\_max > original\_dimm\_flaten:

range\_max = original\_dimm\_flaten

range\_step = round(math.floor((range\_max-range\_min) / step\_count), 0)

return range(range\_min, range\_max, range\_step)

def start(self):

data = DataPreproces.normalize\_x(self.data)

name\_of\_first\_object = str(self.steps[0][1])

if any(substring in name\_of\_first\_object for substring in ("TruncatedSVD", "PCA", "FastICA", "TSNETransformer", "NMF")):

data = DataPreproces.unwrapper(data)

noised = ""

if self.add\_noise:

data = DataPreproces.add\_gaussian\_noise(data)

noised = "\_noised"

print(data[0].shape)

ops = OptimalParametersSelector(data, self.steps, self.param\_grid)

ops.find\_optimal\_param()

ops.plot\_accuracy\_matrix()

result = ops.get\_result()

save\_path = os.path.join(self.results\_folder, str(self.evaluation\_model))

result\_save\_name = '\_'.join([step[0] for step in self.steps]) + noised + ".png"

data\_save\_name = '\_'.join([step[0] for step in self.steps]) + noised + ".csv"

plot(result, save\_path, result\_save\_name)

final\_traning\_form.create\_form(len(self.steps)-1)

usersParam = final\_traning\_form.get\_input\_values()

if len(usersParam) == 0:

return

reducted\_data = ops.get\_reducted\_data(usersParam)

X = np.concatenate((self.data[0], self.data[2]))

Y = np.concatenate((self.data[1], self.data[3]))

DataPreproces.save\_reducted\_data(X, Y, reducted\_data, save\_path, data\_save\_name)

if \_\_name\_\_ == "\_\_main\_\_":

# show form1

selected\_data = get\_search\_criteria(DataLoader().get\_datasets\_names(), MyEvaluationModels.evaluation\_model\_list)

print(selected\_data)

chosen\_dataset = DataLoader().get\_data\_by\_name(selected\_data["dataset"])

chosen\_evaluation\_method = MyEvaluationModels.get\_eval\_model\_by\_name(selected\_data["evaluation\_method"])

# analyse chosen\_dataset

best\_alg\_list = MyReductionModels.reduction\_model\_list[0] # temp - all

# show form2 with pipline creation

selected\_data2 = get\_pipline\_algorithms\_and\_ranges(best\_alg\_list)

print(selected\_data2)

chosen\_algs = []

reduction\_ranges = []

for item in selected\_data2:

chosen\_algs.append(MyReductionModels.get\_reduction\_model\_by\_name(item["chosen\_algorithm"]))

reduction\_ranges.append(item["reduction\_range"])

AutoReductor(chosen\_dataset, chosen\_evaluation\_method, chosen\_algs, reduction\_ranges).start()

**Data.py**

import numpy as np

import tensorflow as tf

import csv

import os

import numpy as np

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

import tkinter as tk

from tkinter import filedialog, messagebox

class DataLoader():

    folder\_path = ""

    test\_size = 0.2

    def \_\_init\_\_(self):

        self.dataset\_dict = {"mnist": self.\_\_mnist, "fashion\_mnist": self.\_\_fashion\_mnist, "cifar10": self.\_\_cifar10, "Import custom dataset": self.\_\_import\_custom\_dataset }

    def get\_datasets\_names(self):

        return list(self.dataset\_dict.keys())

    def \_\_mnist(self):

        (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

        return (x\_train, y\_train, x\_test, y\_test)

    def \_\_fashion\_mnist(self):

        (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

        return (x\_train, y\_train, x\_test, y\_test)

    def \_\_cifar10(self):

        (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

        return (x\_train, y\_train.ravel(), x\_test, y\_test.ravel())

    def \_\_import\_custom\_dataset(self):

        x = []

        y = []

        root = tk.Tk()

        root.withdraw()

        self.folder\_path = filedialog.askdirectory(title="Оберіть папку")

        root.destroy()

        if not os.path.exists(self.folder\_path):

            print("Error, folder is'n exist")

            exit(1)

        for class\_name in os.listdir(self.folder\_path):

            class\_folder\_path = os.path.join(self.folder\_path, class\_name)

            if not os.path.isdir(class\_folder\_path):

                continue

            for image\_name in os.listdir(class\_folder\_path):

                image\_path = os.path.join(class\_folder\_path, image\_name)

                img = load\_img(image\_path)

                x.append(img\_to\_array(img))

                y.append(class\_name)  # припускаємо, що назва папки — це клас

        x = np.array(x)

        y = np.array(y)

        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=self.test\_size, random\_state=42, stratify=y)

        return (x\_train, y\_train, x\_test, y\_test)

    def get\_data\_by\_name(self, name):

        try:

            data\_loader\_func = self.dataset\_dict[name]

            return data\_loader\_func()

        except Exception as e:

            print("Error in Data module:")

            print(e)

            exit(1)

class DataPreproces():

    def normalize\_x(data: tuple):

        (x\_train, y\_train, x\_test, y\_test) = data

        x\_train = x\_train.astype('float32') / 255.0

        x\_test = x\_test.astype('float32') / 255.0

        return (x\_train, y\_train, x\_test, y\_test)

    def unwrapper(data: tuple):

        '''

            Return 1d arr for each x item

            Example:

            (60000,10,10,3) -> (60000, 300)

        '''

        (x\_train, y\_train, x\_test, y\_test) = data

        x\_train\_flat = x\_train.reshape((len(x\_train), -1))

        x\_test\_flat = x\_test.reshape((len(x\_test), -1))

        return (x\_train\_flat, y\_train, x\_test\_flat, y\_test)

    def add\_gaussian\_noise(data: tuple, mean=0.0, std=0.1):

        (x\_train, y\_train, x\_test, y\_test) = data

        noise = np.random.normal(mean, std, x\_train.shape)

        x\_train\_noise = x\_train + noise

        noise = np.random.normal(mean, std, x\_test.shape)

        x\_test\_noise = x\_test + noise

        return (x\_train\_noise, y\_train, x\_test\_noise, y\_test)

    def save\_reducted\_data(X, Y, X\_reduced, save\_path, data\_save\_name):

        if not os.path.exists(save\_path):

            os.makedirs(save\_path)

        if not data\_save\_name.endswith('.csv'):

            data\_save\_name += '.csv'

        file\_path = os.path.join(save\_path, data\_save\_name)

        with open(file\_path, mode='w', newline='') as file:

            writer = csv.writer(file, delimiter=';')

            # Записуємо заголовки

            writer.writerow(['Image\_id', 'class', 'original\_width', 'original\_height', 'original\_channel', 'reducted\_data'])

            # Проходимо через всі зображення та їхні редуковані вектори

            for i, (image, img\_class, reduced\_data) in enumerate(zip(X, Y, X\_reduced)):

                if len(image.shape) == 2:

                    original\_height, original\_width = image.shape

                    original\_channel = 1

                elif len(image.shape) == 3:

                    original\_height, original\_width, original\_channel = image.shape

                reducted\_data\_str = ', '.join(map(str, reduced\_data))  # Форматуємо вектор редукованих даних в рядок

                # Записуємо рядок в файл

                writer.writerow([i + 1, img\_class, original\_width, original\_height, original\_channel, reducted\_data\_str])

        print(f'Data successfully saved to {file\_path}')

        messagebox.showinfo("Info", f'Data successfully saved to {file\_path}')

**MyEvaluationModels.py**

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

evaluation\_model\_list = (LogisticRegression(), MLPClassifier(solver="lbfgs"), SVC(kernel='linear'), DecisionTreeClassifier(), RandomForestClassifier())

def get\_eval\_model\_by\_name(name):

for model in evaluation\_model\_list:

if name == str(model):

return model

**MyReductionModels.py**

import math

from sklearn.base import BaseEstimator, TransformerMixin

from tensorflow.keras.layers import Input, Dense, Flatten, Reshape, Conv2D, MaxPooling2D, UpSampling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import MeanSquaredError

from sklearn.decomposition import PCA, TruncatedSVD, FastICA, NMF

from sklearn.manifold import TSNE

class Autoencoder(BaseEstimator, TransformerMixin):

def \_\_init\_\_(self, input\_shape, lat\_dim\_ae=30):

self.lat\_dim\_ae = lat\_dim\_ae

self.input\_shape = input\_shape

self.autoencoder = None

def build\_model(self):

inputs = Input(shape=self.input\_shape)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = MaxPooling2D((2, 2), padding='valid')(x)

x = Flatten()(x)

encoded = Dense(self.lat\_dim\_ae)(x)

# Calculate the shape after flattening

conv\_output\_shape = (self.input\_shape[0] // 4, self.input\_shape[1] // 4, 64)

dense\_units = math.prod(conv\_output\_shape)

x = Dense(dense\_units)(encoded)

x = Reshape(conv\_output\_shape)(x)

# x = Dense(7 \* 7 \* 64)(encoded)

# x = Reshape((7, 7, 64))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

x = UpSampling2D((2, 2))(x)

decoded = Conv2D(self.input\_shape[2], (3, 3), activation='sigmoid', padding='same')(x)

self.autoencoder = Model(inputs, decoded)

self.autoencoder.compile(optimizer=Adam(), loss=MeanSquaredError())

def fit(self, X, y=None, \*\*fit\_params):

self.build\_model()

self.autoencoder.fit(X, X, \*\*fit\_params)

return self

def transform(self, X):

result\_x = self.autoencoder.predict(X)

return result\_x.reshape((len(result\_x), -1))

class TSNETransformer(BaseEstimator, TransformerMixin):

def \_\_init\_\_(self, n\_components=2):

self.n\_components = n\_components

self.tsne = TSNE(n\_components=n\_components)

def fit(self, X, y=None):

self.X\_transformed = self.tsne.fit\_transform(X)

return self

def transform(self, X, y=None):

return self.X\_transformed

reduction\_model\_list = [

[Autoencoder, TruncatedSVD, PCA, FastICA, TSNETransformer, NMF],

["AE\_\_lat\_dim\_ae", "SVD\_\_n\_components", "PCA\_\_n\_components",

"FastICA\_\_n\_components", "TSNE\_\_n\_components", "NMF\_\_n\_components"]

]

def get\_reduction\_model\_by\_name(name):

for model in reduction\_model\_list[0]:

if str(model) == name:

index = reduction\_model\_list[0].index(model)

return model, reduction\_model\_list[1][index]

**OptimalParametersSelector.py**

import pandas as pd

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import Pipeline

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

import numpy as np

class OptimalParametersSelector():

def \_\_init\_\_(self, data, steps, param\_grid) -> None:

'''

Example:

steps = [

('SVD', TruncatedSVD()),

('MLP', MLPClassifier(solver="lbfgs"))]

pipeline = Pipeline(steps)

param\_grid = {

'SVD\_\_n\_components': list(range(10,160,10)),

}

'''

self.data = data

self.steps = steps

self.param\_grid = param\_grid

self.pipeline = Pipeline(steps)

print(steps)

print(self.param\_grid)

def find\_optimal\_param(self): # todo

(x\_train, y\_train, x\_test, y\_test) = self.data

grid\_search = GridSearchCV(self.pipeline, param\_grid=self.param\_grid, cv=2, scoring='accuracy', return\_train\_score=True, verbose=2)

grid\_search.fit(x\_train, y\_train)

params = ['param\_' + key for key in self.param\_grid.keys() ]

self.results = pd.DataFrame(grid\_search.cv\_results\_)[[\*params, 'mean\_test\_score', 'mean\_fit\_time', 'mean\_score\_time']]

def get\_result(self):

return self.results

def plot\_accuracy\_matrix(self):

arr = self.results

if len(arr) != 5:

return

heatmap\_data = arr.pivot(index=arr.columns[0], columns=arr.columns[1], values='mean\_test\_score')

plt.figure(figsize=(10, 8))

sns.heatmap(heatmap\_data, annot=True, cmap='YlOrRd', fmt='.2f')

plt.title('Heatmap of Accuracy')

plt.ylabel(arr.columns[0])

plt.xlabel(arr.columns[1])

plt.show()

def get\_reducted\_data(self, param\_list):

steps = self.steps[:-1]

for step\_id, param in zip(range(len(steps)), param\_list):

if steps[step\_id][0] == "AE":

dataset\_input\_shape = steps[step\_id][1].input\_shape

alg = type(steps[step\_id][1])(dataset\_input\_shape, param)

else:

alg = type(steps[step\_id][1])(param)

steps[step\_id] = (steps[step\_id][0], alg)

pipeline = Pipeline(steps)

dataX = np.concatenate((self.data[0], self.data[2]))

X\_reduced = pipeline.fit\_transform(dataX)

print(X\_reduced.shape)

print(X\_reduced[0])

return X\_reduced

**gui/final\_traning\_form.py**

import tkinter as tk

from tkinter import messagebox

entries = []

def train(window):

values = [entry.get() for entry in entries]

try:

values = [int(value) for value in values]

except ValueError:

messagebox.showerror("Помилка", "Всі значення повинні бути числовими")

return

print("Train function called with values:", values)

global input\_values

input\_values = values

messagebox.showinfo("Info", "Редукція даних почалась.")

window.quit()

def create\_form(num\_el):

global entries

entries = []

global input\_values

input\_values = []

window = tk.Tk()

window.title("Навчання на власних параметрах")

# Напис вгорі вікна

label = tk.Label(window, text="Введіть бажані параметри для алгоритмів редукції:", font=("Arial", 16))

label.pack(pady=10)

input\_frame = tk.Frame(window)

input\_frame.pack(pady=10)

# Поля для вводу

for i in range(num\_el):

field\_frame = tk.Frame(input\_frame)

field\_frame.pack(side=tk.LEFT, pady=5) # Використовуємо TOP замість LEFT для вертикального розташування

label = tk.Label(field\_frame, text=f"Алг. {i + 1}")

# label.pack(side=tk.LEFT, padx=5)

label.grid(row=0, column=0, padx=5)

entry = tk.Entry(field\_frame)

# entry.pack(side=tk.LEFT, padx=5)

entry.grid(row=1, column=0, padx=5)

entries.append(entry)

# Frame для кнопок

button\_frame = tk.Frame(window)

button\_frame.pack(side=tk.BOTTOM, pady=10, fill=tk.X)

# Кнопка Вихід

exit\_button = tk.Button(button\_frame, text="Вихід", command=window.quit)

exit\_button.pack(side=tk.LEFT, padx=5, expand=True, fill=tk.X)

# Кнопка Тренування

train\_button = tk.Button(button\_frame, text="Тренування", command=lambda: train(window))

train\_button.pack(side=tk.RIGHT, padx=5, expand=True, fill=tk.X)

window.mainloop()

def get\_input\_values():

return input\_values

**gui/main\_window.py**

import tkinter as tk

from tkinter import ttk

class SearchForm(tk.Tk):

def \_\_init\_\_(self, dataset\_list, evaluation\_model\_list):

super().\_\_init\_\_()

self.geometry("400x300")

self.title("Search Form")

# Перше випадаюче меню "Датасет"

self.dataset\_label = tk.Label(self, text="Датасет")

self.dataset\_label.pack(pady=5)

self.dataset\_var = tk.StringVar()

self.dataset\_menu = ttk.Combobox(self, textvariable=self.dataset\_var, state="readonly", width=30)

self.dataset\_menu['values'] = dataset\_list

self.dataset\_menu.pack(pady=5)

self.dataset\_menu.bind("<<ComboboxSelected>>", self.check\_selection)

# Друге випадаюче меню "Спосіб класифікації"

self.classification\_label = tk.Label(self, text="Спосіб класифікації")

self.classification\_label.pack(pady=5)

self.classification\_var = tk.StringVar()

self.classification\_menu = ttk.Combobox(self, textvariable=self.classification\_var, state="readonly", width=30)

self.classification\_menu['values'] = evaluation\_model\_list

self.classification\_menu.pack(pady=5)

self.classification\_menu.bind("<<ComboboxSelected>>", self.check\_selection)

# Кнопка "Пошук"

self.search\_button = tk.Button(self, text="Пошук", state=tk.DISABLED, command=self.search)

self.search\_button.pack(pady=20)

def check\_selection(self, event=None):

if self.dataset\_var.get() and self.classification\_var.get():

self.search\_button.config(state=tk.NORMAL)

else:

self.search\_button.config(state=tk.DISABLED)

def search(self):

self.result\_data = {

"dataset": self.dataset\_var.get(),

"evaluation\_method": self.classification\_var.get()

}

self.destroy() # Закрити форму після натискання кнопки

def get\_search\_criteria(dataset\_list, classification\_model\_list):

form = SearchForm(dataset\_list, classification\_model\_list)

form.mainloop()

return form.result\_data

**gui/parameters\_form.py**

import tkinter as tk

from tkinter import ttk

class PiplineForm(tk.Tk):

def \_\_init\_\_(self, best\_alg):

super().\_\_init\_\_()

self.geometry("500x600")

self.title("Pipline Form")

self.best\_alg = best\_alg

# Лічильник кількості блоків

self.block\_count\_label = tk.Label(self, text="Кількість етапів в пайплайні")

self.block\_count\_label.pack(pady=5)

self.block\_count\_var = tk.IntVar(value=1) # За замовчуванням 1

self.block\_count\_spinbox = tk.Spinbox(self, from\_=1, to=3, width=10, textvariable=self.block\_count\_var, command=self.update\_blocks)

self.block\_count\_spinbox.pack(pady=5)

# Контейнер для блоків

self.blocks\_frame = tk.Frame(self)

self.blocks\_frame.pack(pady=10)

# Масив для збереження блоків

self.blocks = []

# Додаємо початкові блоки (1 блок за замовчуванням)

self.update\_blocks()

# Кнопка "Start"

self.search\_button = tk.Button(self, text="Start", state=tk.DISABLED, command=self.start\_button, width=20)

self.search\_button.pack(pady=20)

# Обробка подій для активації кнопки "Start"

self.bind("<KeyRelease>", self.check\_selection)

self.bind("<ButtonRelease-1>", self.check\_selection)

def update\_blocks(self):

"""Оновлює кількість блоків на основі вибраного числа."""

# Видаляємо всі попередні блоки

for block in self.blocks:

block['frame'].destroy() # Знищуємо сам Frame, який містить блок

self.blocks = []

# Створюємо нові блоки

for i in range(self.block\_count\_var.get()):

block = self.create\_block(i + 1)

self.blocks.append(block)

# self.check\_selection() # Оновлення стану кнопки після зміни блоків

def create\_block(self, block\_num):

"""Створює новий блок з випадаючим меню та полями введення."""

block\_frame = tk.Frame(self.blocks\_frame)

block\_frame.pack(pady=10)

# випадаюче меню

alg\_label = tk.Label(block\_frame, text=f"Пропоновані алгоритми (Етап {block\_num})")

alg\_label.grid(row=0, column=0, columnspan=3, pady=5)

alg\_var = tk.StringVar()

alg\_menu = ttk.Combobox(block\_frame, textvariable=alg\_var, state="readonly", width=60)

alg\_menu['values'] = self.best\_alg

alg\_menu.grid(row=1, column=0, columnspan=3, pady=5)

alg\_menu.bind("<<ComboboxSelected>>", self.check\_selection)

# Поля для введення min\_reduction, max\_reduction, step\_count

min\_reduction\_label = tk.Label(block\_frame, text="Мінімальна ступінь стиснення")

min\_reduction\_label.grid(row=2, column=0, padx=5)

min\_reduction\_spinbox = tk.Spinbox(block\_frame, from\_=1, to=100, width=10)

min\_reduction\_spinbox.grid(row=3, column=0, padx=5)

max\_reduction\_label = tk.Label(block\_frame, text="Максимальна ступінь стиснення")

max\_reduction\_label.grid(row=2, column=1, padx=5)

max\_reduction\_spinbox = tk.Spinbox(block\_frame, from\_=1, to=100, width=10)

max\_reduction\_spinbox.grid(row=3, column=1, padx=5)

step\_count\_label = tk.Label(block\_frame, text="Крок")

step\_count\_label.grid(row=2, column=2, padx=5)

step\_count\_spinbox = tk.Spinbox(block\_frame, from\_=1, to=100, width=10)

step\_count\_spinbox.grid(row=3, column=2, padx=5)

# Зберігаємо всі елементи блоку для подальшого використання

return {

"alg\_var": alg\_var,

"min\_reduction\_spinbox": min\_reduction\_spinbox,

"max\_reduction\_spinbox": max\_reduction\_spinbox,

"step\_count\_spinbox": step\_count\_spinbox,

"frame": block\_frame

}

def check\_selection(self, event=None):

"""Перевіряє всі блоки на коректність введених даних для активації кнопки Start."""

valid = True

for block in self.blocks:

alg = block['alg\_var'].get()

min\_reduction = block['min\_reduction\_spinbox'].get()

max\_reduction = block['max\_reduction\_spinbox'].get()

step\_count = block['step\_count\_spinbox'].get()

# Перевірка, чи всі поля заповнені та коректні

if not (alg and min\_reduction.isdigit() and max\_reduction.isdigit() and step\_count.isdigit() and int(min\_reduction) <= int(max\_reduction)):

valid = False

break

# Активація/деактивація кнопки "Start"

if valid:

self.search\_button.config(state=tk.NORMAL)

else:

self.search\_button.config(state=tk.DISABLED)

def start\_button(self):

"""Збирає результати з усіх блоків і закриває форму."""

self.result\_data = []

for block in self.blocks:

block\_data = {

"chosen\_algorithm": block['alg\_var'].get(),

"reduction\_range": (

int(block['min\_reduction\_spinbox'].get()),

int(block['max\_reduction\_spinbox'].get()),

int(block['step\_count\_spinbox'].get())

)

}

self.result\_data.append(block\_data)

self.destroy() # Закрити форму після натискання кнопки

def get\_pipline\_algorithms\_and\_ranges(best\_alg\_list):

form = PiplineForm(best\_alg\_list)

form.mainloop()

return form.result\_data