Кафедра електронних обчислювальних машин

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

**до бакалаврської кваліфікаційної роботи на тему**

Програмна платформа створення штучних нейронних мереж

A software platform for creating artificial neural networks

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент групи \_\_\_\_\_\_\_\_\_КІ-406 Ярмола Ю. Ю.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр, прізвище та ініціали)

|  |  |  |  |
| --- | --- | --- | --- |
| **Керівник роботи** |  |  | ( Олексів М. В. ) |
| **Консультанти** |  |  | (Войцеховська Ю. В.) |
|  |  |  | ( ) |
|  |  |  | ( ) |
|  |  |  | ( ) |
|  |  |  | ( ) |
|  |  |  | ( ) |

Завідувач кафедри д.т.н, Рудик Ю.І.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ІКТА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра\_електронних обчислювальних машин\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність\_123 «Комп’ютерна інженерія»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«ЗАТВЕРДЖУЮ»

Завідувач кафедри ЕОМ

проф. Дунець Р. Б.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 р.

**ЗАВДАННЯ**

**на кваліфікаційну роботу (проект) студента групи \_КІ-406\_\_ОР \_\_\_бакалавр\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ярмола Юрій Юрійович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(*прізвище,* *ім’я,* *по батькові*)

1. Тема роботи (проекту)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Програмна платформа створення штучних нейронних мереж \_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_A software platform for creating artificial neural networks \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(у разі виконання комплексної роботи в дужках вказується “комплексна робота (проект)”)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

затверджена наказом по університету від «\_24\_»\_\_\_квітня\_\_\_\_2025 р. № 1487 – 4 – 08

1. Термін подання студентом закінченої роботи (проекту) \_\_\_\_\_\_\_9 червня 2025\_\_\_\_\_\_\_\_\_\_\_\_
2. Вихідні дані для роботи (проекту)

1) Час класифікації одного RGB зображення розміром 128x128 пікселів на системі з RTX 4060 та Intel Core i5-12500H не більше 0.1 с

2) Використання метрик F1-score, Recall та Precision для оцінки якісних параметрів навчання ШНМ

3) Максимальний можливий розмір навчальної множини не менше 1000 зразків

4) Підтримка типів відеофайлу для навчання: .mp4, .avi, .mov, .mkv, .flv.

1. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)

\_\_\_Провести аналітичний огляд сучасних методів і підходів до створення нейронних мереж. Розробити вимоги до функціональності та продуктивності програмної системи. Розробити структурну схему побудови системи та описати алгоритми її роботи. Реалізувати програмний модуль для обробки, тренування та тестування моделі. Провести валідацію розробленої системи на тестовому наборі даних.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Перелік графічного матеріалу

\_Лист 1 – Система розробки штучних нейронних мереж. Схема структурна, Лист 2 –Взаємодія користувача з модулями системи. Діаграма послідовностей, Лист 3 – Генерація даних. Блок-схема алгоритму, Лист 4 – Навчання моделей. Блок-схема алгоритму, Лист 5 – Перевірка роботи моделей. Блок-схема алгоритму\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Консультування роботи (проекту), із зазначенням розділів роботи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Розділ | Консультант | Завдання видав | | Завдання прийняв | |
| підпис | дата | підпис | дата |
| 4 | Войцеховська Ю. В. |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

8. Дата, коли видано завдання \_\_\_\_\_\_\_3 березня 2025\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  |  |  | (*підпис*) |  |
|  | Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  |  |  | (*підпис*) |  |
|  | КАЛЕНДАРНИЙ ПЛАН | | |  |
|  |  |  |  |  |
| №  з/п | Назва етапів роботи (проекту) | | Термін виконання  етапів роботи (проекту) | Примітка |
| 1 | Аналітичний огляд та обґрунтування вибору методу розв’язання задачі | | Березень |  |
| 2 | Вибір та обґрунтування засобів для реалізації | | Квітень |  |
| 3 | Реалізація алгоритму розв’язання задачі | | Травень |  |
| 4 | Економічна частина | | Червень |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |
|  |  | |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(*підпис*)

Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(*підпис*)

# Розширена анотація

Ярмола Ю. Ю., Олексів М. В.(керівник). Програмна платформа створення штучних нейронних мереж. Бакалаврська кваліфікаційна робота. Національний університет «Львівська політехніка», Львів, 2025.

**Розширена анотація**

Штучні нейронні мережі (ШНМ) є одним із найпотужніших інструментів сучасного машинного навчання, який використовується для розв'язання складних задач у різних сферах, включаючи обробку зображень, тексту, аналізу даних та прогнозування. В основі ШНМ лежить біологічна аналогія роботи мозку, що дозволяє моделювати складні нелінійні залежності між даними [1-5, 10].Основною метою цієї роботи є дослідження теоретичних основ ШНМ, розробка ефективних алгоритмів навчання та створення прототипу нейронної мережі для вирішення прикладної задачі.

**Об’єкт дослідження:** процеси розробки та навчання штучних нейронних мереж.

**Предмет дослідження:** алгоритми навчання нейронних мереж та їх ефективність у вирішенні конкретних завдань.

**Мета дослідження:** створення та дослідження нейронної мережі, здатної вирішувати завдання з класифікації зображень та прогнозування даних із застосуванням сучасних методів навчання.

У першому розділі проведено аналіз теоретичних основ нейронних мереж, включаючи їх архітектуру, механізми навчання та основні типи мереж, такі як багатошарові перцептрони, згорткові нейронні мережі та рекурентні нейронні мережі. Розглянуто основні алгоритми навчання, такі як зворотне поширення помилки та оптимізаційні методи (Adam, SGD)[3-4].

У другому розділі представлено аналіз інструментів та технологій для розробки ШНМ[7], таких як PyTorch[5], TensorFlow та Keras. Проведено огляд найпоширеніших бібліотек для обробки даних, включаючи NumPy, Pandas та OpenCV[2, 8]. Також обговорено переваги GPU-прискорення при навчанні ШНМ.

У третьому розділі описано практичну реалізацію нейронної мережі. Створено датасет для навчання моделі, який містить анотації та зображення. Для підготовки даних використано аугментацію зображень, включаючи обертання, масштабування та відображення. Модель, побудована в рамках роботи, використовує згорткову[6] архітектуру для класифікації зображень. Результати навчання та тестування демонструють високу точність моделі (понад 95% на тестовому наборі даних).

**Ключові слова:** штучні нейронні мережі, машинне навчання, класифікація зображень, PyTorch.

**Перелік використаних літературних джерел:**

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
3. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90
4. Chollet, F. (2017). Deep Learning with Python. Manning Publications.
5. Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 8024–8035.
6. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105.
7. Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.
8. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998–6008). https://doi.org/10.48550/arXiv.1706.03762
9. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536. https://doi.org/10.1038/323533a0
10. Silver, D., et al. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354–359. https://doi.org/10.1038/nature24270

# Extended annotation

Yarmola Y. Y., Oleksiv M. V. (supervisor). A software platform for creating artificial neural networks. Bachelor's degree thesis. — Lviv Polytechnic National University, Lviv, 2025.

**Extended annotation**

Artificial neural networks (ANNs) are one of the most powerful tools of modern machine learning, which is used to solve complex problems in various fields, including image processing, text, data analysis and forecasting. ANNs are based on a biological analogy of the brain, which allows modeling complex nonlinear dependencies between data[1-5, 10]. The main goal of this work is to study the theoretical foundations of ANNs, develop effective learning algorithms and create a prototype of a neural network to solve an applied problem.

**Object of research:** processes of the artificial neural networks development and training.

**Subject of research:** the neural network training algorithms and their effectiveness in solving specific problems.

**The purpose of the study:** to create and study a neural network capable of solving image classification and data prediction problems using modern learning methods.

The first section analyzes the theoretical foundations of neural networks, including their architecture, learning mechanisms, and the main types of networks, such as multilayer perceptrons, convolutional neural networks, and recurrent neural networks. The main learning algorithms, such as backpropagation of error and optimization methods (Adam, SGD)[3-4], are considered.

The second section presents an analysis of tools and technologies for developing ANNs[7], such as PyTorch[5], TensorFlow, and Keras. The most common data processing libraries, including NumPy, Pandas, and OpenCV[2, 8], are reviewed. The advantages of GPU acceleration in ANN training are also discussed.

The third section describes the practical implementation of a neural network. A dataset containing annotations and images was created for training the model. Image augmentation, including rotation, scaling, and reflection, was used to prepare the data. The model built in the work uses a convolutional[6] architecture for image classification. The training and testing results demonstrate high accuracy of the model (over 95% on the test dataset).

**Keywords:** artificial neural networks, machine learning, image classification, PyTorch, convolutional.

**References:**

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
3. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778). https://doi.org/10.1109/CVPR.2016.90
4. Chollet, F. (2017). Deep Learning with Python. Manning Publications.
5. Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 8024–8035.
6. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105.
7. Brown, T., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.
8. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998–6008). https://doi.org/10.48550/arXiv.1706.03762
9. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536. https://doi.org/10.1038/323533a0
10. Silver, D., et al. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354–359. https://doi.org/10.1038/nature24270

# Зміст

[Вступ 12](#_Toc200216707)

[Розділ 1. АНАЛІТИЧНИЙ ОГЛЯД ТА ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ 16](#_Toc200216708)

[1.1 Аналіз сучасних підходів до створення штучних нейронних мереж 16](#_Toc200216709)

[1.2 Аналіз відомих інструментів та платформ для побудови, навчання та оцінки моделей 21](#_Toc200216710)

[1.3 Аналіз методів розв’язання задачі 25](#_Toc200216711)

[1.3.1 Методи формування даних для тренування та валідації моделей 25](#_Toc200216712)

[1.3.2 Методи навчання штучних нейронних мереж 28](#_Toc200216713)

[1.3.3 Методи перевірки моделі 29](#_Toc200216714)

[1.4 Аналіз можливих режимів функціонування платформи 32](#_Toc200216715)

[1.5 Аналіз методів тестування та оцінювання 33](#_Toc200216716)

[Висновки до розділу 1 35](#_Toc200216717)

[Розділ 2. ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ 36](#_Toc200216718)

[2.1. Аналіз задачі та розробка вимог до програмної системи 36](#_Toc200216719)

[2.1.1 Функціональні вимоги 36](#_Toc200216720)

[2.1.2 Нефункціональні вимоги 37](#_Toc200216721)

[2.1.3 Вимоги до апаратного забезпечення 38](#_Toc200216722)

[2.2. Розробка структурної схеми програмного рішення 39](#_Toc200216723)

[2.3. Вибір мови програмування та технологій 42](#_Toc200216724)

[2.4 Вибір типу штучних нейронних мереж 43](#_Toc200216725)

[2.5. Засоби розробки програмного забезпечення 47](#_Toc200216726)

[2.6. Вибрані бібліотеки та фреймворки 49](#_Toc200216727)

[Висновки до розділу 2 51](#_Toc200216728)

[Розділ 3. РЕАЛІЗАЦІЯ АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ 53](#_Toc200216729)

[3.1 Розробка алгоритму розв’язання задачі 53](#_Toc200216730)

[3.1.1 Розробка алгоритму формування структурованих даних 53](#_Toc200216731)

[3.1.2 Розробка алгоритму навчання ШНМ 55](#_Toc200216732)

[3.1.3 Розробка алгоритму перевірки навченої моделі 57](#_Toc200216733)

[3.2 Реалізація програмного забезпечення 58](#_Toc200216734)

[3.2.1 Розробка архітектури програмного забезпечення 59](#_Toc200216735)

[3.2.2 Розробка модуля генерації датасету 61](#_Toc200216736)

[3.2.3 Розробка модуля навчання моделі 65](#_Toc200216737)

[3.2.4 Розробка модуля перевірки моделі 67](#_Toc200216738)

[3.2.5 Розробка інтерфейсу 69](#_Toc200216739)

[3.4 Валідація 73](#_Toc200216740)

[3.4.1 Модульне тестування 73](#_Toc200216741)

[3.4.2 Тестування ефективності системи 75](#_Toc200216742)

[Висновки до розділу 3 82](#_Toc200216743)

[Розділ 4. ЕКОНОМІЧНА ЧАСТИНА 84](#_Toc200216744)

[4.1 Розрахунок витрат на розробку програмного забезпечення 84](#_Toc200216745)

[4.2 Кошторис витрат на розробку програмного забезпечення 88](#_Toc200216746)

[4.3 Визначення експлуатаційних витрат 88](#_Toc200216747)

[4.4 Розрахунок витрат на підготовку даних та реалізацію проектного рішення на ПК 90](#_Toc200216748)

[4.5. Розрахунок ціни споживання проектного рішення 91](#_Toc200216749)

[4.6. Визначення показників економічної ефективності 92](#_Toc200216750)

[Висновки до економічної частини 93](#_Toc200216751)

[Висновки 94](#_Toc200216752)

[Список літератури 96](#_Toc200216753)

[Додатки 103](#_Toc200216754)

[Додаток А. Лістинг файлів розробленої системи 103](#_Toc200216755)

# Вступ

**Актуальність роботи**

На сьогоднішній день штучні нейронні мережі (ШНМ) активно застосовуються у багатьох галузях науки, техніки та промисловості. Їх використання дозволяє вирішувати задачі, які раніше вважалися надскладними, наприклад, розпізнавання образів, обробка природної мови, прогнозування поведінки систем та автоматизація рутинних процесів. Розвиток технологій машинного навчання створив передумови для інтеграції ШНМ у програмні продукти, спрямовані на підвищення ефективності аналізу даних, оптимізації процесів та прийняття рішень.

Однак, створення якісних нейронних мереж вимагає значного обсягу обчислювальних ресурсів, глибокого розуміння теоретичних основ та практичних інструментів. У цьому контексті виникає необхідність у розробці універсальних платформ, які забезпечували б автоматизацію основних етапів роботи з ШНМ — від створення датасетів до візуалізації результатів. Це дозволить значно зменшити час на підготовку даних, оптимізувати процеси навчання моделей і підвищити доступність технологій штучного інтелекту для широкого кола користувачів.

Дослідження актуальне, оскільки забезпечує систематизацію підходів до розробки платформ, що працюють із ШНМ, та надає нові інструменти для ефективного впровадження штучного інтелекту в різні галузі.

**Мета і завдання дослідження**

Метою роботи є розробка платформи для автоматизації процесів створення, навчання та використання штучних нейронних мереж.

Для досягнення мети поставлено такі завдання:

1. Провести аналіз сучасних підходів та інструментів для роботи з ШНМ, визначити їх переваги та недоліки.
2. Розробити алгоритми автоматизації ключових етапів роботи з ШНМ, таких як підготовка даних, навчання моделей, перевірка точності та прогнозування.
3. Обґрунтувати вибір інструментів та технологій для реалізації платформи.
4. Розробити та валідувати платформу, яка дозволить ефективно використовувати можливості штучних нейронних мереж.
5. Провести тестування створеної платформи та оцінити її ефективність.

**Об’єкт та предмет дослідження**

Об’єктом дослідження є процеси автоматизації роботи з ШНМ. Предметом дослідження є методи та інструменти для створення універсальної платформи, яка дозволяє автоматизувати ключові етапи розробки та використання нейронних мереж.

**Методологія дослідження**

Для досягнення поставленої мети використано такі методи дослідження:

1. Аналітичний метод для вивчення наукових публікацій, технічної документації та сучасних інструментів у галузі машинного навчання.
2. Емпіричний метод для розробки алгоритмів, проведення експериментів та тестування моделі.
3. Метод моделювання для створення і навчання ШНМ із використанням спеціалізованих інструментів.
4. Метод системного аналізу для оцінки ефективності створеної платформи.
5. Методи візуалізації для аналізу результатів роботи моделі та підготовки звітів.

**Практичне значення**

Практична цінність роботи полягає у розробці універсальної платформи, яка дозволяє значно спростити процеси створення та використання ШНМ. Реалізована система може використовуватись для аналізу даних, побудови прогнозів, автоматизації рутинних процесів та вирішення прикладних задач у галузях, де потрібна висока точність та швидкість обробки інформації.

Платформа забезпечує:

* інтерактивний інтерфейс для створення та оновлення датасетів;
* автоматизацію процесу навчання моделей;
* оцінку точності моделей на тестових наборах даних;
* прогнозування з використанням навченої моделі;
* зручну візуалізацію результатів для прийняття рішень.

Розроблена платформа може бути інтегрована у різні галузі, такі як медицина, фінанси, освіта, промисловість тощо.

**Структура бакалаврської кваліфікаційної роботи**

Бакалаврська кваліфікаційна робота складається з вступу, п’яти розділів, висновків, списку використаних джерел та додатків.

* У першому розділі проведено аналітичний огляд літератури та обґрунтовано актуальність дослідження.
* У другому розділі описано вибір інструментів та методів розробки платформи.
* У третьому розділі детально описано розробку, тестування та оцінку ефективності платформи.
* У четвертому розділі розглянуто економічну ефективність впровадження платформи.

Робота містить аналітичний огляд, технічні рішення, результати тестування та рекомендації для подальшого використання розробленої платформи.

# Розділ 1. АНАЛІТИЧНИЙ ОГЛЯД ТА ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

За останні кілька років у світі різко зріс інтерес до штучних нейронних мереж (ШНМ) [1]. Це можна пояснити виходом зрозумілої для більшості людей нейронної мережі у вигляді чату, що відповідає на будь-які запитання, генерує та розпізнає зображення, виконує як прості так і складні задачі [2]. Цей інтерес також спровокував розвиток і інших спеціалізованих нейронних мереж, які здатні виконувати добре одну вузьку задачу, наприклад генерація або розпізнавання зображень. Проте у всіх цих ШНМ у основі роботи лежить принцип імітації нейрону нервової клітини людського мозку [3], який має різні реалізації, продуктивність та інші параметри які впливають на навчання та роботу готової мережі.

## 1.1 Аналіз сучасних підходів до створення штучних нейронних мереж

Основні принципи створення штучних нейронних мереж базуються на моделюванні подібних до біологічних нейронів мереж [4]. Це дозволяє штучним нейронним мережам обробляти великі обсяги даних, знаходити приховані закономірності та взаємозв'язки, які можуть бути невидимими для традиційних методів аналізу [5]. Розвиток обчислювальної техніки і математичних алгоритмів дозволив створювати різноманітні архітектури нейронних мереж, які оптимізовані для вирішення певних типів завдань. Нижче наведено огляд основних і найбільш популярних архітектур [6].

**Багатошарові перцептрони**

Багатошарові перцептрони (MLP) є базовим елементом для багатьох інших типів нейронних мереж. Це класичний тип штучної нейронної мережі, яка складається з вхідного шару, одного або декількох прихованих шарів і вихідного шару[7].

Принцип роботи багатошарових перцептронів полягає у тому, що кожен нейрон у прихованому і вихідному шарах отримує вхідні значення, обчислює зважену суму цих значень і пропускає її через активаційну функцію, що дозволяє моделювати нелінійні залежності[8]. Основні застосування мережі цього типу для бінарної та багатокласової класифікації, а також для задач регресії[9]. Обмеження є у тому, що MLP менш ефективний у роботі з даними високої розмірності, наприклад із зображеннями чи послідовностями, оскільки його структура не враховує просторових або часових залежностей[10].

**Згорткові нейронні мережі**

Згорткові нейронні мережі (CNN) спеціалізуються на роботі з візуальними даними[11]. Вони використовують шари згортки для автоматичного виявлення особливостей, таких як краєві деталі, текстури або геометричні форми, що робить їх надзвичайно ефективними в обробці зображень. Особливостями згорткових шарів є те, що вони аналізують невеликі фрагменти зображень, зберігаючи інформацію про просторове розташування пікселів. Популярні моделі: ResNet, VGG, MobileNet і інші, які застосовуються в задачах класифікації, сегментації, виявлення об’єктів на зображеннях[12]. Додаткові можливості згорткових нейронних мереж є використання пулінгових шарів, що дозволяє зменшувати розмірність даних, зберігаючи при цьому основну інформацію[13].

**Рекурентні нейронні мережі**

Рекурентні нейронні мережі (RNN) розроблені для обробки послідовних даних, таких як текст, музика, часові ряди або відео[14]. Вони враховують залежності між елементами послідовності шляхом передачі інформації через приховані стани. Перевага над іншими моделями у тому, що моделі LSTM (довготривала короткочасна пам'ять) і GRU (згорткові рекурентні блоки) усувають проблему згасання градієнтів, що дозволяє працювати з довшими послідовностями[15]. Основним застосуванням є машинний переклад, автоматичне складання текстів, аналіз настроїв, прогнозування ринкових трендів тощо[16].

**Мережі асоціативної пам’яті**

Цей клас нейронних мереж використовується для задач, пов’язаних із запам'ятовуванням і відтворенням зв’язків між даними. Архітектура цих нейронних мереж це модель Хопфілда, яка дозволяє моделювати функцію асоціативної пам'яті[17]. Основні сфери застосування це - оптимізація складних систем, задачі відновлення втрачених даних, вирішення задач із частково заданими умовами[18].

**Трансформери**

Трансформери представляють сучасний підхід до роботи з послідовними даними. Їх основою є механізм самопідлаштування (self-attention), який дозволяє моделі враховувати контекстні залежності на всіх рівнях послідовності[19]. Особливостями цих нейронних мереж - це здатність обробляти дуже довгі послідовності, адаптивність до контексту, можливість паралельного навчання. Популярними моделями на сьогодні є GPT, BERT, T5, що демонструють видатні результати у розв'язанні задач генерації тексту, класифікації, машинного перекладу[20-21, 66-68].

**Методи навчання**

Для навчання штучних нейронних мереж використовуються різні підходи:

* Підконтрольне навчання: моделі навчаються на даних із мітками, дозволяючи робити прогнози для нових даних.
* Безконтрольне навчання: дозволяє знаходити приховані структури у немаркованих даних, що корисно для кластеризації, пошуку аномалій.
* Навчання з підкріпленням: забезпечує адаптацію моделей у динамічних середовищах шляхом отримання винагороди за виконані дії.

**Оптимізація**

Для ефективного навчання штучних нейронних мереж, які працюють із великими обсягами даних, використовуються різноманітні алгоритми оптимізації. Основна мета оптимізації полягає у мінімізації функції втрат, яка визначає різницю між прогнозованими та фактичними значеннями. Цей процес передбачає поступову адаптацію вагових коефіцієнтів моделі для досягнення найкращих результатів.

Одним із базових підходів до оптимізації є градієнтний спуск, який ґрунтується на обчисленні градієнта функції втрат стосовно кожного параметра мережі. Цей метод дозволяє поступово змінювати ваги у напрямку зменшення значення функції втрат. Традиційний градієнтний спуск працює з фіксованою швидкістю навчання, яка визначає величину зміни ваг. Однак вибір оптимальної швидкості є критичним: надто велика швидкість може призвести до нестабільності моделі, а надто мала — до надмірно тривалого навчання.

Сучасні модифікації градієнтного спуску, такі як Adam (Adaptive Moment Estimation) і RMSprop (Root Mean Square Propagation), значно покращують процес оптимізації. Adam є одним із найбільш популярних методів, оскільки він поєднує переваги моментного градієнта та адаптивного масштабування швидкості навчання[22]. Цей метод автоматично змінює швидкість навчання для кожного параметра, враховуючи історію градієнтів, що забезпечує швидшу і стабільнішу збіжність. RMSprop, своєю чергою, також адаптує швидкість навчання, але зосереджується на зменшенні коливань градієнта, особливо у задачах із шумовими даними[23].

Вибір алгоритму оптимізації залежить від типу задачі, складності моделі та доступних обчислювальних ресурсів. Важливо зазначити, що процес оптимізації потребує попереднього налаштування гіперпараметрів, таких як швидкість навчання, момент і розмір партії даних. Ефективна оптимізація є ключовою для забезпечення високої точності та узагальнюючої здатності нейронної мережі.

**Популярні інструменти**

Для розробки, навчання та тестування штучних нейронних мереж активно використовуються сучасні програмні фреймворки, які надають зручні інструменти та бібліотеки. До найпопулярніших платформ належать TensorFlow, PyTorch та Keras, кожна з яких має унікальні переваги і можливості.

TensorFlow — це масштабована бібліотека з відкритим кодом, розроблена Google[24], яка дозволяє працювати з великими обсягами даних і складними моделями. Завдяки високій продуктивності та підтримці багатоплатформних обчислень (CPU, GPU, TPU), TensorFlow забезпечує швидке навчання навіть у великих мережах. Інтеграція з бібліотекою TensorBoard дозволяє візуалізувати процес навчання та аналізувати продуктивність моделей. Важливою особливістю TensorFlow є можливість реалізації моделей на рівні низькорівневого програмування, що забезпечує більшу гнучкість для розробників, які працюють із нестандартними архітектурами.

PyTorch, створений компанією Facebook[25], є ще однією популярною платформою, яка особливо активно використовується в наукових дослідженнях та експериментах. PyTorch забезпечує інтуїтивно зрозумілий інтерфейс і підтримує динамічні обчислювальні графи, що робить процес навчання більш прозорим і зрозумілим. Завдяки цьому PyTorch дозволяє швидко експериментувати з новими архітектурами і методами оптимізації. Крім того, PyTorch інтегрується з популярними інструментами для машинного навчання, такими як Hugging Face Transformers, що розширює його можливості для роботи з текстовими даними.

Keras є високорівневим інтерфейсом для побудови нейронних мереж[26], що спрощує процес розробки моделей. Спочатку створений як окремий проєкт, Keras зараз інтегрований у TensorFlow. Цей інструмент дозволяє легко створювати складні моделі, використовуючи простий синтаксис, що особливо корисно для новачків у галузі машинного навчання. Хоча Keras не надає стільки гнучкості, як TensorFlow чи PyTorch, його основна перевага полягає у швидкому прототипуванні.

Важливо зазначити, що кожен із цих фреймворків має свої сильні та слабкі сторони, тому вибір інструменту залежить від конкретних вимог проєкту. TensorFlow підходить для виробничих систем, де важливі стабільність і масштабованість, PyTorch — для дослідницьких задач, а Keras — для швидкого створення моделей із мінімальними витратами часу. Усі три інструменти активно підтримуються спільнотою розробників, що забезпечує постійне вдосконалення їх функціональності та можливостей.

Сучасні підходи до створення ШНМ є надзвичайно різноманітними та дозволяють вирішувати широкий спектр задач. Кожна архітектура і метод навчання мають свої сильні сторони, обмеження і найкращі сфери застосування, що дозволяє ефективно адаптувати ШНМ до різних умов і завдань.

## 1.2 Аналіз відомих інструментів та платформ для побудови, навчання та оцінки моделей

Розробка, навчання та оцінка моделей є складною задачею, яка вимагає спеціалізованих інструментів чи платформ, які забезпечують високу продуктивність роботи з даними, підготовка чи створення даних та гнучкість у розробці чи дослідженні ШНМ. Найпопулярніші платформи розраховані зазвичай на програмістів, що мають досвід у розробці ШНМ, хоча є і платформи, які розраховані на дослідження чи базові налаштування через зручний GUI. Вибір відповідного інструменту залежить від типу задачі, обсягу даних та вимог до обчислювальних ресурсів.

* TensorFlow[24]

Відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявляння та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди.

* PyTorch[25]

Відкрита бібліотека машинного навчання на основі бібліотеки Torch, що застосовується для задач комп'ютерного зору та обробки природної мови.

* Keras[26]

Відкрита нейромережна бібліотека, написана мовою Python. Вона здатна працювати поверх TensorFlow, Microsoft Cognitive Toolkit, R, Theano та PlaidML. Спроектовану для уможливлення швидких експериментів з мережами глибокого навчання, її зосереджено на тому, щоби вона була зручною в користуванні, модульною та розширюваною.

* Transformers[27]

Бібліотека попередньо навчених моделей обробки природної мови, комп'ютерного зору, аудіо та мультимодальних моделей для логічного висновку та навчання.

* Roboflow[28]

Комерційна платформа для розпізнавання об’єктів на відео та експериментів з навчанням моделей комп’ютерного зору. Особливо платформа корисна для роботи з власними наборами зображень

Кожна розроблена платформа має свої сильні та слабкі сторони, які потрібно враховувати при проектуванні платформи розробки ШНМ. Нижче наведені недоліки уже реалізованих рішень.

**TensorFlow**

TensorFlow є однією з найпотужніших відкритих бібліотек для реалізації нейронних мереж, розробленою компанією Google. Проте, незважаючи на високу гнучкість та широкі можливості, бібліотека характеризується наступними недоліками:

* Висока складність засвоєння, що ускладнює використання її користувачами без відповідної підготовки;
* Відсутність вбудованих інструментів для анотації зображень та управління наборами даних;
* Необхідність інтеграції з додатковими рішеннями для забезпечення повного циклу розробки моделей комп’ютерного зору.
* Не можливість використовувати повні можливості через CUDA на операційній системі Windows.

**PyTorch**

PyTorch — ще одна популярна бібліотека для розробки моделей машинного навчання, яка вирізняється підтримкою динамічного графа обчислень та активно використовується у наукових дослідженнях. Однак, її використання супроводжується такими обмеженнями:

* Необхідність глибоких знань програмування для реалізації більшості завдань;
* Відсутність інструментів для створення, анотації та попередньої обробки наборів зображень у вбудованому вигляді;
* Значні трудові витрати на початкових етапах проекту.

**Keras**

Keras є високорівневою нейронною бібліотекою, що працює поверх інших фреймворків, таких як TensorFlow чи Theano. Основною перевагою Keras є простота у використанні. Водночас, до недоліків можна віднести:

* Орієнтацію виключно на етап побудови та навчання моделей, без підтримки інструментів для керування наборами зображень;
* Обмежену функціональність у контексті розв’язання прикладних задач комп’ютерного зору без сторонніх засобів.

**Transformers**

Бібліотека Transformers, створена компанією Hugging Face, містить велику кількість попередньо навчених моделей для задач обробки природної мови, комп’ютерного зору, аудіо тощо. Незважаючи на переваги, бібліотека має наступні недоліки:

* Основний фокус зосереджено на обробці текстової інформації, тоді як можливості для комп’ютерного зору є обмеженими;
* Відсутність засобів для повноцінної роботи з власними зображеннями без попередньої підготовки даних.

**Roboflow**

Roboflow є сучасною хмарною платформою для реалізації проектів у галузі комп’ютерного зору. Її головна перевага — це зручність роботи з наборами зображень, включно з їх анотацією, попередньою обробкою та генерацією нових даних. Однак, навіть при значних перевагах, платформа має низку обмежень:

* Обмеження функціональності у безкоштовній версії, що ускладнює масштабування проекту;
* Тренування моделей відбувається на сторонніх серверах, що може створити загрози безпеці при роботі з конфіденційною інформацією;
* Обмежена гнучкість налаштування моделей порівняно з відкритими бібліотеками низького рівня.

## 1.3 Аналіз методів розв’язання задачі

У цьому підрозділі розглядаються основні підходи та методи, що застосовуються у задачах створення, навчання та оцінювання штучних нейронних мереж (ШНМ). Розуміння принципів їх роботи є необхідною основою для подальшої розробки та реалізації відповідної програмної платформи.

### 1.3.1 Методи формування даних для тренування та валідації моделей

Якість роботи штучної нейронної мережі (ШНМ) значною мірою залежить від якості та обсягу навчальних даних. Формування навчальної та валідаційної вибірок є ключовим процесом, який визначає ефективність навчання моделі, її здатність до узагальнення і стійкість до змін у вхідних даних. Правильне формування даних дозволяє не тільки забезпечити високу точність передбачень, але й запобігти проблемам, таким як перенавчання або недостатнє навчання моделі.

**1. Збір даних**

Збір даних є першим і найбільш ресурсозатратним етапом, оскільки від нього залежить якість усього подальшого процесу[29-31]. Джерела даних визначаються специфікою завдання:

* Відкриті набори даних. Такі ресурси, як ImageNet, COCO, або Kaggle, є популярними завдяки доступності та великій кількості готових прикладів. Вони особливо корисні для задач, які вже мають усталені стандарти.
* Користувацькі записи. Унікальні задачі часто потребують спеціалізованих даних, які можуть бути отримані через інтеграцію платформи з іншими системами або зібрані безпосередньо від користувачів[32].
* Власноруч зібрані дані[33]. Цей метод підходить для вузькоспеціалізованих завдань, наприклад, у медицині або промисловості, де важливо забезпечити контроль над усіма аспектами процесу збору.

У кожному випадку дані мають бути репрезентативними, щоб охоплювати всі можливі сценарії використання моделі. Це запобігає ситуаціям, коли модель демонструє гарну точність лише на специфічних даних, але не може узагальнити знання.

**2. Анотація даних**

Після збору даних обов’язковим є етап анотації[34]. Для задач класифікації, сегментації або детекції об'єктів дані мають бути марковані відповідно до поставлених цілей. Наприклад:

* У задачах класифікації зображення мають бути позначені класами, до яких вони належать.
* У задачах сегментації необхідно визначити межі кожного об'єкта.
* У задачах детекції визначаються координати об'єктів на зображеннях.

Методи анотації варіюються залежно від складності задачі. Ручна анотація забезпечує найвищу точність, але потребує значних часових ресурсів, тоді як автоматична анотація дозволяє зекономити час, але може вимагати подальшого коригування.

Анотація є критично важливою для забезпечення точності моделі, оскільки невідповідність у маркуванні даних може призвести до помилок у навчанні та зниження ефективності.

**3. Попередня обробка**

Попередня обробка є важливим етапом підготовки даних[35], що включає низку процедур для приведення даних у формат, придатний для навчання моделі. Основні методи попередньої обробки включають:

* Нормалізацію розмірів зображень: це дозволяє забезпечити однакові умови для аналізу даних. Наприклад, приведення всіх зображень до розміру 128x128 пікселів.
* Фільтрацію шумів: видалення зайвих елементів або артефактів, що можуть заважати моделі.
* Обробку кольорових каналів: перетворення зображень у формат RGB або сірого кольору залежно від вимог задачі.
* Нормалізацію пікселів: масштабування значень у діапазон [0, 1] або [-1, 1], що сприяє стабільнішому навчанню.

Ці кроки допомагають мінімізувати вплив непотрібних варіацій у даних, які можуть сповільнити або ускладнити процес навчання.

**4. Аугментація**

Аугментація[36] даних є методикою, яка дозволяє штучно збільшити обсяг навчального набору. Цей процес є особливо важливим, якщо кількість вихідних даних є обмеженою. Основні методи аугментації:

* Геометричні перетворення: обертання, зміна масштабу, дзеркальне відображення.
* Колірні трансформації: зміна яскравості, контрастності, інверсія кольорів.
* Додавання шуму: моделювання реальних умов, наприклад, розмиття або зернистості.
* Додавання варіацій фону: це робить модель більш стійкою до змін у середовищі.

Аугментація підвищує здатність моделі до узагальнення та робить її більш стійкою до різних змін у вхідних даних.

**5. Розподіл на вибірки**

Правильний розподіл даних на навчальну, валідаційну та тестову вибірки є основою для об’єктивного оцінювання продуктивності моделі[37]. Типовий розподіл виглядає так:

* 70%-80% даних використовується для навчання.
* 10%-15% призначається для валідації.
* 10%-15% залишається для тестування.

Навчальна вибірка забезпечує оптимізацію параметрів моделі, валідаційна — допомагає уникнути перенавчання, а тестова дозволяє оцінити продуктивність моделі на нових даних.

Методи формування даних для тренування та валідації відіграють критичну роль у створенні ефективної штучної нейронної мережі. Від правильного вибору джерел даних, їх анотації, попередньої обробки до аугментації та розподілу на вибірки залежить успішність навчання моделі. Комплексний підхід до формування даних забезпечує високу точність, стабільність та адаптивність моделі до реальних умов використання.

### 1.3.2 Методи навчання штучних нейронних мереж

Процес навчання ШНМ є базовим етапом у побудові моделей штучного інтелекту. У сучасних дослідженнях найбільш поширеним є підхід із використанням зворотного поширення помилки та градієнтного спуску. Основні кроки:

1. Ініціалізація. Визначення архітектури мережі та початкових значень вагів.
2. Функція втрат і оптимізатор. Вибір метрики, яка відображає помилку, та способу її мінімізації (наприклад, Adam, SGD).
3. Пряме проходження (forward pass). Обчислення вихідного сигналу на основі вхідних даних.
4. Обчислення помилки. Визначення відхилення результату від очікуваного.
5. Зворотне поширення помилки (backpropagation). Розрахунок похідних функції втрат для оновлення вагів.
6. Оновлення вагів. Застосування методів оптимізації.
7. Багаторазове повторення. Процес виконується багаторазово (епохи) для покращення результатів.
8. Валідація. Оцінка проміжного результату на валідаційній вибірці для виявлення перенавчання.
9. Фінальна оцінка. Після завершення навчання модель перевіряється на тестовій вибірці.

### 1.3.3 Методи перевірки моделі

Перевірка ефективності штучної нейронної мережі (ШНМ) є ключовим етапом у процесі розробки, що дозволяє визначити її здатність до узагальнення та виконання поставленої задачі. Ретельний підхід до оцінки забезпечує прозорість та надійність результатів, дозволяючи виявити недоліки моделі, а також потенційні напрямки для її вдосконалення.

**Формування тестової вибірки**

Перший крок у перевірці моделі — формування тестової вибірки. Цей набір даних створюється до початку навчання моделі, що гарантує його незалежність від навчального процесу. Тестова вибірка повинна бути репрезентативною та відповідати статистичним характеристикам загального набору даних. Це дозволяє оцінити здатність моделі узагальнювати закономірності на даних, які вона раніше не бачила.

**Ключові аспекти формування тестової вибірки:**

* Розмір вибірки: оптимальний обсяг складає 10–15% від загального набору даних. Цей обсяг забезпечує статистично значущі результати, залишаючи достатньо даних для навчання.
* Структура: вибірка має включати дані з усіх класів задачі, що гарантує рівномірну перевірку моделі на різних категоріях.

**Завантаження навченої моделі**

Після завершення навчання модель зберігається у форматі, який дозволяє її подальше використання без потреби повторного навчання. Завантаження навченої моделі забезпечує її ідентичність стану, у якому вона була збережена, виключаючи можливі відхилення у результатах через зміни у процесі навчання.

**Генерація передбачень**

На основі даних тестової вибірки модель генерує передбачення. Цей процес включає пропуск вхідних даних через всі шари мережі для отримання кінцевих прогнозів. Основною метою цього етапу є оцінка здатності моделі точно класифікувати або прогнозувати результати залежно від завдання.

**Порівняння з еталонними значеннями**

Результати, отримані від моделі, порівнюються з еталонними значеннями (ground truth), які визначаються заздалегідь. Це дозволяє оцінити точність передбачень моделі та виявити класи або категорії, у яких спостерігається зниження якості.

**Обчислення метрик**

Для кількісної оцінки ефективності моделі використовуються різні метрики[38-40], вибір яких залежить від специфіки задачі:

* Точність (Accuracy): визначає частку правильно класифікованих прикладів серед усіх.
* Повнота (Recall): показує здатність моделі виявляти всі позитивні приклади.
* F1-міра: збалансована метрика, яка враховує як точність, так і повноту, особливо корисна у задачах із незбалансованими даними.
* ROC-AUC: використовується для оцінки моделей, які виконують бінарну класифікацію, і показує якість класифікації на різних порогах.

**Аналіз результатів**

На основі отриманих метрик проводиться аналіз ефективності моделі. Якщо результати є незадовільними, можливі наступні дії:

1. Перегляд структури мережі: аналіз архітектури може виявити слабкі місця, які потребують модифікації, наприклад, додавання нових шарів чи зміну функцій активації.
2. Покращення даних: якість тестової вибірки може бути підвищена шляхом її очищення або переформування.
3. Оптимізація гіперпараметрів: налаштування таких параметрів, як швидкість навчання або розмір мініпакетів, може покращити ефективність.

Методи перевірки моделі є невід’ємною складовою розробки ШНМ, адже вони дозволяють оцінити її здатність виконувати поставлені задачі у реальних умовах. Ретельна перевірка забезпечує не лише високу якість роботи моделі, а й прозорість процесу розробки, що особливо важливо для її подальшого застосування та довіри користувачів.

## 1.4 Аналіз можливих режимів функціонування платформи

Відомі рішення для розв’язання задачі створення, навчання та використання моделей штучного інтелекту передбачають поділ роботи платформи на кілька основних функціональних режимів. Ці режими спрямовані на забезпечення зручності використання, оптимізації процесів навчання, тестування та застосування моделей. У рамках аналізу літератури було виявлено, що подібні платформи зазвичай функціонують у таких основних режимах:

1. **Створення та оновлення датасету**   
   Більшість платформ передбачає можливість збору, обробки та структурування даних. Відомі підходи включають автоматичне очищення та анотацію, що дозволяє зменшити трудомісткість цього процесу. Однак у деяких випадках користувачі також мають змогу вручну коригувати анотації для забезпечення більшої точності.
2. **Навчання моделі**   
   Цей режим зазвичай включає механізми автоматизованого налаштування гіперпараметрів та оптимізації ваг нейронної мережі. У науковій літературі широко використовуються фреймворки, такі як TensorFlow та PyTorch, які надають інструменти для гнучкого налаштування та навчання моделей.
3. **Перевірка точності моделі**   
   Оцінка якості побудованих моделей здійснюється за допомогою метрик, таких як точність, повнота, F1-міра тощо. Деякі платформи також надають інструменти для виявлення помилкових передбачень, що сприяє подальшому вдосконаленню моделей.
4. **Прогнозування**   
   Інференс є ключовою функцією платформ, яка дозволяє застосовувати навчені моделі для обробки нових даних. Сучасні рішення часто оптимізують цей процес для прискорення роботи, зокрема шляхом використання апаратних прискорювачів (наприклад, GPU або TPU).

**Візуалізація результатів**

Для підвищення зручності користувачів платформи зазвичай інтегрують засоби візуалізації, такі як графіки, таблиці або інтерактивні інтерфейси для перегляду результатів роботи моделі. Деякі платформи надають можливість створення звітів у різних форматах, що є важливим для аналізу ефективності

## 1.5 Аналіз методів тестування та оцінювання

Основне завдання тестування моделі полягає у перевірці її здатності правильно обробляти невідомі дані. Для цього використовується тестовий набір даних, який повинен бути незалежним від навчального і валідаційного наборів. Тестування проводиться за допомогою таких метрик, як точність (accuracy), precision, recall, F1-міра, AUC-ROC у задачах класифікації або середнє абсолютне відхилення (Mean Absolute Error, MAE)[41], середньоквадратична помилка (Mean Squared Error, MSE) та коефіцієнт детермінації (R²) у задачах регресії.

Також важливим є тестування моделі на змінених даних. Для цього застосовуються аугментації, такі як зміна яскравості, поворот зображень чи додавання шуму. Такий підхід дозволяє оцінити стійкість моделі до змін у входах і виявити можливі слабкі сторони.

**Тестування програмного забезпечення**

Якість програмного забезпечення, що забезпечує функціонування моделі, також потребує ретельної перевірки. Для цього застосовуються різні види тестування, які включають:

* **Unit-тестування**

Unit-тести (модульні тести)[42] дозволяють перевірити правильність роботи окремих компонентів програмного забезпечення, таких як функції, методи чи класи. Наприклад, можна тестувати функції для попередньої обробки даних, обчислення метрик, реалізації аугментації чи інших важливих процесів. Модульне тестування дозволяє виявити потенційні дефекти на ранніх етапах розробки, що значно підвищує стабільність та якість програмного забезпечення. Завдяки автоматизації тестування зменшується ризик людських помилок, а можливість повторного виконання тестів забезпечує надійність у разі внесення змін до коду. У перспективі це також сприяє скороченню витрат на підтримку та розвиток системи.

* **Тестування продуктивності**

Цей тип тестування спрямований на оцінювання швидкодії програми, її здатності обробляти великі обсяги даних, працювати під високим навантаженням та витримувати стресові ситуації. Для цього застосовуються інструменти, які дозволяють імітувати різні сценарії роботи системи.

* **End-to-end тестування**[43]

Цей тип тестування охоплює всі аспекти роботи програми – від введення даних до отримання кінцевого результату. Наприклад, end-to-end тестування може включати перевірку повного циклу роботи з даними: завантаження зображення, його аугментація, обробка моделлю та збереження результатів.

* **Методи автоматизації тестування**

У сучасній розробці програмного забезпечення значна увага приділяється автоматизації тестування. Це дозволяє зменшити вплив людського фактора, підвищити швидкість перевірки та забезпечити стабільність результатів. Для цього використовуються інструменти, такі як PyTest або Unittest для Python. Автоматизовані тести можуть бути інтегровані у процес розробки за допомогою інструментів CI/CD (наприклад, GitHub Actions, Jenkins), що забезпечує регулярну перевірку кожного нового внесеного коду.

Різні методи тестування забезпечують перевірку програмного забезпечення з різних сторін, і їх поєднання є найефективнішим підходом. Наприклад, unit-тести забезпечують базову стабільність окремих компонентів, інтеграційні тести перевіряють їх взаємодію, а енд-то-енд тести гарантують функціональність системи в цілому.

## Висновки до розділу 1

У першому розділі було здійснено аналітичний огляд літератури та існуючих підходів до розв’язання задачі створення, навчання та використання моделей штучного інтелекту. Розглянуто типові функціональні режими роботи платформ, які дозволяють реалізувати повний цикл обробки даних і роботи з моделями: створення та оновлення датасету, навчання моделі, перевірка точності, прогнозування та візуалізація результатів.

Аналіз показав, що основними напрямами розвитку таких платформ є автоматизація процесів, підвищення точності моделей та забезпечення зручності для користувачів. Разом з тим, кожен із розглянутих підходів має свої переваги та недоліки, які необхідно враховувати при розробці власного рішення.

Подальші кроки дослідження передбачають визначення конкретних вимог до функціональності платформи, вибір інструментів і технологій для її реалізації, а також розробку алгоритмів і архітектури системи, що враховуватиме особливості вирішуваного завдання.

# Розділ 2. ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ

У даному розділі здійснюється всебічний аналіз та обґрунтований вибір інструментів, технологій і середовищ, що використовуються для реалізації програмної платформи. Розглядаються можливі альтернативи, порівнюються їхні функціональні можливості, переваги та недоліки з урахуванням вимог до системи. На основі проведеного аналізу формується аргументований вибір оптимальних засобів, які забезпечать ефективну, масштабовану та зручну у підтримці реалізацію програмного забезпечення.

## 2.1. Аналіз задачі та розробка вимог до програмної системи

Основною задачею у даній роботі є розробка програмної платформи, здатної автоматизовано формувати анотований датасет та навчати ним різні ШНМ з мінімальним втручанням у код. Розроблена система має забезпечувати повний цикл — від збору та анотації зображень до їх завантаження у модель, а також виконання класифікації на основі отриманих результатів. Такий підхід дозволяє значно спростити підготовку даних та забезпечити ефективне використання ШНМ для вирішення задач комп’ютерного зору.

### 2.1.1 Функціональні вимоги

До основних функціональних можливостей системи належать:

* **Автоматизоване створення датасету** — система повинна забезпечити формування вибірки зображень із відповідною структурою (у форматі папок з анотаціями), що підтримується більшістю фреймворків глибокого навчання[61].
* **Підтримка популярних типів відео** – реалізувати підтримку популярних типів відео таких як mp4, avi, mov, WebM.
* **Анотування зображень** — реалізувати привласнення кожному зображенню певної категорії для класифікації.
* **Механізм навчання ШНМ** — використання згорткової нейронної мережі, зокрема архітектури на базі torch.nn.Sequential, яка включає згорткові шари (Conv2d), пулінг (MaxPool2d) та повнозв'язні шари (Linear), є необхідною функціональністю для навчання моделі на підготовленому датасеті.
* **Класифікація нових зображень** — система повинна забезпечити можливість класифікації зображень, що не входили до навчальної вибірки, з використанням збереженої моделі.
* **Час класифікація зображень** – система повинна класифікувати одне зображення не довше 0.1 с для забезпечення обробки в реальному часі.

### 2.1.2 Нефункціональні вимоги

**Обсяг і тип оброблюваних даних**

Система має оперувати з зображеннями, представленими у форматах PNG, JPEG тощо, з можливістю масштабування до вказаного розміру (наприклад, 32x32 пікселі). Очікується обробка наборів даних, що складаються з десятків тисяч зображень, з чіткою структурою за класами, що дає змогу ефективно застосовувати механізми автоматичної обробки.

**Вимоги до продуктивності**

* Навчання ШНМ має відбуватися із максимальною ефективністю: використання графічного процесора (GPU) за допомогою CUDA[44-45].
* Процес класифікації одного зображення повинен займати не більше 3 с.
* Необхідно забезпечити можливість збереження ваг моделі та їх подальшого завантаження без повторного навчання.

**Вимоги до масштабованості**

* Архітектура системи має бути придатною до горизонтального масштабування. Наприклад, можлива інтеграція з хмарними сервісами (Google Colab, AWS, Azure) для розширення обчислювальних можливостей.
* Враховуючи можливість генерації довільної кількості зображень, важливо передбачити механізм автоматичного розподілу навантаження під час генерації або навчання.

### 2.1.3 Вимоги до апаратного забезпечення

Для забезпечення коректної, ефективної та стабільної роботи програмного забезпечення, яке реалізує автоматичне формування анотованих датасетів та класифікацію зображень за допомогою методів машинного навчання, висуваються певні вимоги до апаратного забезпечення. Ці вимоги залежать від обсягів оброблюваних даних, складності моделей, а також частоти запуску операцій тренування та тестування.

**Мінімальні вимоги:**

Мінімальні характеристики дозволяють запускати систему в обмеженому режимі — з невеликим датасетом, спрощеною архітектурою моделі або використанням попередньо навченої нейромережі.

* Процесор (CPU): 4-ядерний (Intel Core i5 або AMD Ryzen 5)
* Оперативна пам’ять (RAM): 8 ГБ
* Накопичувач (HDD/SSD): 50 ГБ вільного простору на диску (рекомендується SSD)
* Графічна підсистема (GPU): Необов’язково, але рекомендовано для пришвидшення обробки зображень

У разі відсутності графічного процесора, моделі на базі PyTorch можуть бути запущені на CPU, однак час навчання значно зростає, що робить систему менш продуктивною при роботі з великими обсягами даних.

**Рекомендовані вимоги:**

Для повноцінного функціонування системи — з можливістю обробки великих наборів зображень, тренування глибоких нейронних мереж та візуалізації процесів — рекомендуються такі характеристики:

* Процесор (CPU): 6–8 ядер (Intel Core i7, AMD Ryzen 7 або еквівалент)
* Оперативна пам’ять (RAM): 16–32 ГБ
* Накопичувач (SSD): не менше 100 ГБ вільного простору
* Графічна підсистема (GPU): NVIDIA з підтримкою CUDA (наприклад, GeForce RTX 3060 або вище) — для ефективного тренування моделей у PyTorch
* Операційна система: Windows 10/11, Ubuntu 20.04+ або інша сучасна ОС, що підтримує бібліотеки машинного навчання

Таким чином, відповідність апаратного забезпечення рекомендованим характеристикам дозволяє досягти високої продуктивності, скоротити час тренування моделей та забезпечити зручну взаємодію користувача з інтерфейсом системи. У майбутньому, за умови розширення функціональності та масштабування обробки даних, ці вимоги можуть бути переглянуті у бік збільшення.

## 2.2. Розробка структурної схеми програмного рішення

Для забезпечення ефективної реалізації програмної системи було проведено аналіз кількох варіантів архітектурного проектування. Серед основних кандидатів розглядалися монолітна, мікросервісна та модульна архітектура. З урахуванням специфіки завдань, масштабів проекту та вимог до продуктивності — було прийнято рішення на користь модульної архітектури.

* Монолітна архітектура, попри простоту реалізації, характеризується тісним зчепленням компонентів, що ускладнює підтримку коду, модифікацію окремих функціональних блоків і тестування. У разі розширення системи або оновлення певного функціоналу можуть виникати труднощі, пов’язані з необхідністю зміни всього застосунку.
* Мікросервісна архітектура передбачає розбиття програми на окремі сервіси, кожен з яких працює незалежно. Цей підхід є потужним і масштабованим, проте вимагає складної інфраструктури — зокрема, налаштування взаємодії між сервісами, організації мережевих запитів, а також систем моніторингу та логування. З огляду на обмежений обсяг задач та відсутність потреби у високій розподіленості системи, впровадження мікросервісної архітектури не є доцільним для даного проекту.
* Модульна архітектура, яка була обрана для реалізації, полягає у логічному поділі програмного забезпечення на незалежні функціональні блоки (модулі), які взаємодіють між собою через чітко визначені інтерфейси. У межах цієї архітектури кожен модуль відповідає за окремий етап обробки даних або виконання функціоналу. Такий підхід забезпечує високу гнучкість, спрощує тестування, дозволяє ізолювати помилки та реалізовувати зміни без впливу на інші компоненти системи.

У даному програмному рішенні виділено кілька основних модулів:

* Модуль генерації даних — відповідає за автоматичне створення зображень із заданими параметрами та їх анотування (наприклад, розміщення геометричних фігур, накладення міток тощо).
* Модуль формування датасету — агрегує згенеровані зображення та створює структуру, придатну для подальшого навчання моделі (наприклад, директорії з класами).
* Модуль навчання ШНМ — здійснює побудову та тренування штучної нейронної мережі на сформованому датасеті, з використанням бібліотеки PyTorch, збереження ваг моделі.
* Модуль класифікації — використовує навчену модель для передбачення класу нових вхідних зображень.
* Допоміжні утиліти (utils) — включають функції візуалізації, завантаження зображень, обробки результатів тощо.



Рис. 2.1 – Структурна схема архітектури програмного рішення

Такий підхід дозволяє розробнику швидко адаптувати систему до нових вимог. Наприклад, у разі зміни архітектури моделі нейронної мережі або бажання використовувати інші дані — достатньо змінити лише відповідний модуль, не зачіпаючи решту системи.

У підсумку, модульна архітектура є найзбалансованішим рішенням для проекту: вона забезпечує достатню продуктивність, спрощує масштабування системи в майбутньому та значно полегшує підтримку й розвиток коду.

## 2.3. Вибір мови програмування та технологій

Вибір мови програмування є критично важливим етапом розробки платформи, оскільки він визначає гнучкість, продуктивність та зручність підтримки системи. У цьому проєкті основною мовою програмування обрано Python[47, 63]. Основними критеріями вибору були: простота та читабельність коду, наявність потужної екосистеми для роботи з даними та навчання моделей штучного інтелекту, кросплатформеність, активна підтримка спільноти та доступність навчальних матеріалів.

Python був обраний завдяки своїй високій читабельності, широкій підтримці бібліотек для аналізу даних (NumPy, Pandas), глибокого навчання (TensorFlow, PyTorch) та візуалізації результатів (Matplotlib, Seaborn). Мова забезпечує зручність інтеграції з іншими технологіями, регулярне оновлення інструментів та масштабованість. Незважаючи на нижчу продуктивність порівняно з компільованими мовами (наприклад, C++ чи Java), це компенсується використанням оптимізованих бібліотек, що дозволяє ефективно виконувати задачі.

Python активно використовується в наукових та дослідницьких колах, що обумовлено наявністю перевірених бібліотек, таких як:

* NumPy[48] — для виконання чисельних обчислень та роботи з багатовимірними масивами;
* Matplotlib та Pillow — для візуалізації, генерації зображень та обробки графічних даних;
* PyTorch — як основний фреймворк для побудови та тренування штучної нейронної мережі, що надає гнучкий інтерфейс, динамічне обчислення графу та підтримку GPU-прискорення.

Крім того, Python забезпечує просту інтеграцію між модулями, можливість швидкого прототипування, активну спільноту розробників, що пришвидшує вирішення потенційних проблем, а також хорошу підтримку сучасних інструментів для аналізу та обробки зображень.

Для організації проектної структури та розділення відповідальностей використовувалися засоби модульного програмування. В результаті код був поділений на логічні компоненти, кожен з яких відповідає за конкретну функціональність: генерація датасету, анотація, навчання моделі та класифікація зображень.

З точки зору сумісності з іншими платформами та розширюваності, Python дозволяє легко інтегруватися з іншими сервісами, підтримує міжплатформенну розробку та забезпечує можливість подальшого перенесення моделі на сервер або хмарну платформу (наприклад, для інференсу в режимі реального часу).

Таким чином, обрана мова програмування та набір технологій повністю відповідають вимогам проекту — як з точки зору реалізації, так і подальшої підтримки, розширення та масштабування.

## 2.4 Вибір типу штучних нейронних мереж

Для вирішення задачі класифікації зображень у межах проєкту було обґрунтовано вибір згорткових нейронних мереж (Convolutional Neural Networks, CNN) [49]. Цей вибір базується на аналізі основних типів штучних нейронних мереж, які використовуються для задач обробки зображень, а також на оцінці їх ефективності, особливостей та відповідності поставленим вимогам[50].

**Основні типи нейронних мереж для класифікації**

На етапі аналізу розглядалися три основні типи моделей:

1. **Повнозв’язні нейронні мережі (Fully Connected Networks, FCN).**   
   Повнозв’язні мережі[51] виконують операції над векторними даними, де кожен нейрон з’єднаний із кожним нейроном наступного шару. Проте їх використання для обробки зображень має суттєві обмеження:

* Ігнорування просторових залежностей. При перетворенні зображення у вектор втрачаються важливі зв’язки між пікселями, що є критичним для аналізу зображень.
* Висока обчислювальна складність. Велика кількість зв’язків між нейронами призводить до значного зростання параметрів моделі, що ускладнює навчання.
* Ризик перенавчання. Через велику кількість параметрів FCN потребує великих обсягів даних для адекватного навчання, що не завжди доступно.

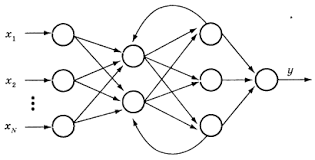


Рис.2.1 – Схема повнозв’язної ШНМ

1. **Рекурентні нейронні мережі (Recurrent Neural Networks, RNN).**   
   Цей тип мереж зазвичай використовується для роботи з послідовними даними, наприклад, текстом чи часовими рядами. У задачах класифікації зображень RNN[52] виявляються малоефективними, оскільки:
   * Недостатня адаптація до просторових даних. RNN розраховані на аналіз лінійних послідовностей, що не відповідає двовимірній структурі зображень.
   * Висока обчислювальна складність. Послідовний характер обробки даних у RNN уповільнює процес навчання й інференсу.

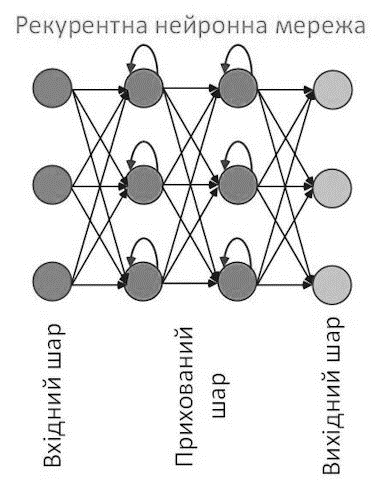


Рис.2.2 – Схема рекурентної ШНМ

1. **Згорткові нейронні мережі (CNN).**   
   CNN[53] були спеціально розроблені для аналізу зображень та просторових даних. Їх ключові переваги:
   * Збереження просторових залежностей. Згорткові шари дозволяють виділяти локальні ознаки зображення (контури, текстури, форми), що критично важливо для задач класифікації.
   * Менша кількість параметрів. Завдяки використанню згорткових операцій модель має значно менше параметрів порівняно з повнозв’язними мережами.
   * Ефективність обчислень. Завдяки зменшенню розмірності вхідних даних (пулінг) CNN є оптимальними для роботи з великими наборами зображень.

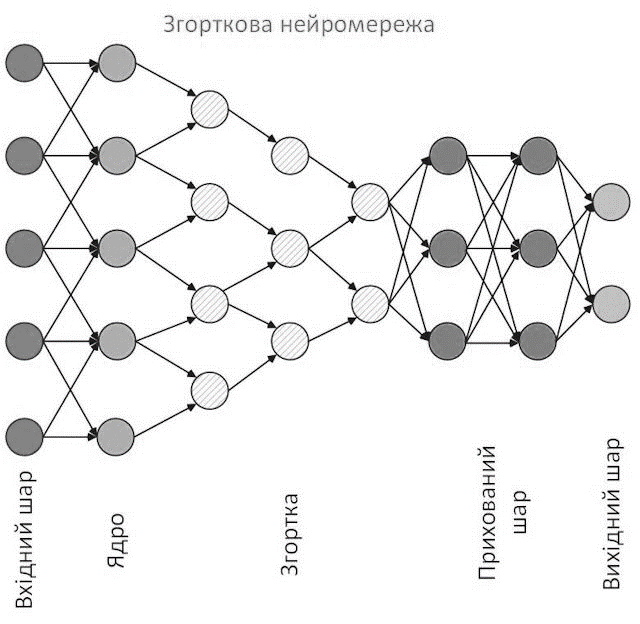


Рис. 2.3 – Схема згорткової ШНМ

**Критерії вибору**

Для обґрунтування вибору моделі було використано такі критерії:

* Врахування специфіки даних. Для задачі класифікації зображень важливо враховувати просторові зв’язки між пікселями, що недоступно для повнозв’язних і рекурентних мереж.
* Обчислювальна ефективність. Згорткові нейронні мережі, порівняно з іншими типами, забезпечують менше навантаження на обчислювальні ресурси, зберігаючи високу якість результатів.
* Можливість тонкого налаштування. CNN дозволяють використовувати попередньо навчені моделі, такі як ResNet50, MobileNetV3 чи EfficientNet, адаптуючи їх до специфічної задачі.

Табл. 2.1- Порівняння основних типів моделей

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерій** | **Повнозв’язні мережі (FCN)** | **Рекурентні мережі (RNN)** | **Згорткові мережі (CNN)** |
| Просторові залежності | Не враховуються | Не враховуються | Враховуються |
| Кількість параметрів | Висока | Середня | Низька |
| Схильність до перенавчання | Висока | Середня | Низька |
| Ефективність обчислень | Низька | Низька | Висока |
| Відповідність задачі | Низька | Низька | Висока |

На основі проведеного аналізу було обрано згорткові нейронні мережі для розв’язання задачі класифікації зображень. CNN забезпечують оптимальний баланс між точністю, обчислювальною ефективністю та гнучкістю налаштувань, що робить їх найбільш придатними для роботи із власноруч зібраними даними в межах даного проекту.

## 2.5. Засоби розробки програмного забезпечення

У процесі реалізації програмного забезпечення важливо забезпечити комфортні умови для розробки, тестування, налагодження та супроводу системи. Для цього використовуються спеціалізовані інструменти, що сприяють підвищенню ефективності праці, структурованості коду та якості кінцевого продукту. У даному проекті обрано низку засобів, які відповідають цим критеріям.

* **Середовище розробки**

Основним інструментом для написання та налагодження коду є інтегроване середовище розробки PyCharm від компанії JetBrains. Воно забезпечує зручне автодоповнення коду, інтеграцію з системами контролю версій, підтримку віртуальних середовищ, інструменти для тестування та відлагодження, а також має широкі можливості для роботи з Python-бібліотеками, що використовуються у проекті. Крім того, PyCharm надає інструменти для аналізу якості коду та рефакторингу, що позитивно впливає на підтримку коду в довгостроковій перспективі.

* **Система контролю версій**

Для керування версіями програмного коду застосовувалася система Git[54]. Усі зміни відслідковуються та документуються за допомогою репозиторію, розміщеного на платформі GitHub[55], що дозволяє зберігати історію змін, працювати з гілками та спрощує співпрацю в команді. Git також забезпечує захист від втрати даних та можливість відкату до попередніх стабільних версій коду.

* **Віртуальне середовище**

Для ізоляції залежностей було створено віртуальне середовище за допомогою venv, яке дозволяє уникнути конфліктів між бібліотеками, що використовуються в проекті. Всі зовнішні пакети, необхідні для виконання функціоналу, встановлюються локально в межах цього середовища. Це підвищує стабільність роботи системи та забезпечує однакові умови запуску на різних пристроях.

* **Бібліотеки для візуалізації та обробки зображень**

Під час реалізації було використано бібліотеки Matplotlib, Pillow (PIL) та інші засоби для обробки та візуалізації зображень. Вони надали широкі можливості для перегляду, анотування, попередньої обробки даних та виводу результатів роботи моделі.

* **Інструменти для тестування та налагодження**

PyCharm має вбудовану підтримку засобів для покрокового виконання коду (debugging), а також дозволяє швидко запускати окремі фрагменти коду для перевірки логіки. Це дозволяє ефективно відслідковувати помилки, працювати з точками зупину та змінними під час виконання, що значно пришвидшує цикл тестування.

У сукупності обрані інструменти дозволили забезпечити структурований, надійний та масштабований підхід до реалізації програмної системи, що відповідає сучасним вимогам до якості розробки інтелектуального програмного забезпечення.

## 2.6. Вибрані бібліотеки та фреймворки

У процесі розробки системи були обрані різноманітні бібліотеки та фреймворки для забезпечення високої ефективності, зручності у використанні та адаптивності. Вибір цих інструментів був обумовлений особливостями завдання, зокрема необхідністю обробки зображень, побудови та тренування моделей для класифікації, а також забезпеченням гнучкості у навчанні.

* **PyTorch**[25] **–** Бібліотека для реалізації машинного навчання, яка є однією з найбільш популярних бібліотек для побудови та тренування моделей глибокого навчання. PyTorch має високу гнучкість, підтримує динамічні обчислювальні графи, що особливо зручно для експериментів, і надає потужні засоби для роботи з нейронними мережами. Цей фреймворк дозволяє значно спростити процес тренування та тестування моделей, а також інтеграцію з різними типами даних.
* **OpenCV**[56] **-** Для обробки зображень на етапі підготовки даних. Вона є стандартним інструментом для комп'ютерного зору та має безліч алгоритмів для обробки та аналізу зображень, що є необхідним при роботі з візуальними даними. OpenCV забезпечує такі можливості, як зміна розміру зображень, застосування фільтрів, виявлення контурів та інші важливі функції для попередньої обробки.
* **Pillow (PIL)**[57] **-** Бібліотека, яка є форком Python Imaging Library (PIL), використовувалась для базових операцій з обробки зображень, таких як зчитування, збереження, обрізка та інші операції. Ця бібліотека дозволяє ефективно працювати з різними форматами зображень та виконувати операції, необхідні для створення анотованих датасетів.
* **NumPy**[48] **-** Для роботи з великими масивами даних, виконання математичних операцій і векторизації алгоритмів. Вона є основним інструментом для обчислень в Python, особливо при обробці числових даних, і забезпечує високу швидкість обчислень завдяки вбудованим функціям для роботи з масивами.
* **Matplotlib**[58] **-** Для візуалізації результатів класифікації, процесу навчання моделі, графічного представлення метрик точності та втрат. Вона дозволяє будувати графіки, гістограми, діаграми і наочно демонструвати результати тренування моделей, що полегшує аналіз та вдосконалення роботи моделі.
* **Tkinter**[59] **-** Для створення інтерфейсу користувача (GUI). Він забезпечує простоту розробки віконних додатків, які дозволяють зручно взаємодіяти з користувачем. За допомогою Tkinter був реалізований інтерфейс для завантаження зображень, запуску процесу навчання моделі, а також для перегляду результатів класифікації.

Таким чином, вибір цих бібліотек та фреймворків був обумовлений їх здатністю підтримувати всі етапи обробки даних — від підготовки та анотування зображень до тренування моделей машинного навчання та виведення результатів. Обрані інструменти забезпечують високу продуктивність, гнучкість та можливість масштабування, що є важливим для подальшого вдосконалення системи.

## Висновки до розділу 2

У другому розділі було здійснено детальний аналіз технічних вимог та обґрунтування вибору архітектурних і технологічних рішень, необхідних для розробки програмної платформи. На основі аналізу функціональних і нефункціональних вимог визначено вимоги до апаратних ресурсів.

Було обґрунтовано вибір модульної архітектури, яка завдяки своїй гнучкості та масштабованості дозволяє ефективно інтегрувати нові компоненти, забезпечує зрозумілу логіку взаємодії між модулями, а також сприяє зручності супроводу та тестування.

Розробка платформи проводиться мовою Python. Це обумовлено її широким застосуванням у галузі машинного навчання, наявністю численних бібліотек (зокрема, PyTorch, OpenCV, Pillow) і сумісністю з сучасними інструментами розробки, такими як PyCharm. Вибір бібліотек здійснюється з урахуванням їхньої функціональності, продуктивності та підтримки сучасних підходів до обробки зображень і побудови моделей ШНМ.

Також сформульовано апаратні вимоги до системи, що враховують можливість запуску на мінімальній конфігурації, а також орієнтовані на ефективну роботу із залученням GPU для прискорення процесів навчання моделей.

Загалом, прийняті рішення створюють міцну технічну основу для подальшої реалізації функціональності системи, її розвитку та адаптації до змін у вимогах користувача або середовища застосування.

# Розділ 3. РЕАЛІЗАЦІЯ АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

## 3.1 Розробка алгоритму розв’язання задачі

Для розробки ШНМ найважливішим елементом є набір даних для навчання та тестування нейронної мережі, адже від якості та кількості цих даних буде залежати ефективність роботи створеної моделі. Саме тому варто виділити цей етап як один із головних при розробці ШНМ, що буде висвітлено у цьому розділі.

### 3.1.1 Розробка алгоритму формування структурованих даних

У межах даного дослідження розроблено алгоритм формування структурованих даних для забезпечення ефективного навчання, валідації та тестування штучної нейронної мережі. Особливістю цього підходу є створення загального базового набору даних із подальшим програмним розподілом його на відповідні підмножини. Такий підхід забезпечує гнучкість у керуванні даними, дозволяючи адаптувати процес навчання до специфіки задачі.

**Збір даних**

Основою алгоритму є формування загального набору даних на основі відеофайлів, наданих користувачем. Для цього використовується алгоритм трекінгу, що дозволяє автоматизувати процес відстеження заданого об’єкта. Алгоритм забезпечує:

* Точність і повторюваність: автоматичне відстеження зменшує ймовірність помилок, притаманних ручній обробці.
* Ефективність: зниження трудовитрат на етапі збору даних завдяки автоматизації процесу.
* Адаптивність: можливість збору даних для різноманітних задач класифікації та розпізнавання.

Використання такого підходу дозволяє створювати загальний набір даних, що містить усі необхідні варіації зображень для подальшого використання.

**Структуризація даних**

Зібрані дані упорядковуються у форматі, зручному для обробки штучною нейронною мережею. Кожне зображення супроводжується міткою, яка визначає його класову належність. Додаткові кроки включають:

* Уніфікацію формату: всі зображення приводяться до однакового розміру та формату (наприклад, 128×128 пікселів), що відповідає вимогам обраної архітектури.
* Контроль якості: відсіювання невідповідних даних, таких як розмиті або некоректно анотовані зображення, для забезпечення чистоти набору.
* Реорганізацію: формування зручної файлової структури, яка сприяє швидкому доступу до даних на етапах навчання та валідації.

**Програмний розподіл даних**

Замість попереднього розподілу на окремі підмножини (навчальний, валідаційний і тестовий набори), реалізовано програмне розділення загального набору даних. Це дозволяє забезпечити:

* Гнучкість: можливість регулювати співвідношення підмножин залежно від специфіки завдання.
* Адаптивність: перерозподіл наборів у випадку зміни вимог або додавання нових даних.
* Контроль репрезентативності: розподіл даних таким чином, щоб підмножини зберігали статистичну однорідність.

Загальний набір програмно розділяється на:

* Навчальний набір (70–80%): використовується для оптимізації параметрів моделі на основі пошуку закономірностей у даних.
* Валідаційний набір (10–15%): використовується для перевірки проміжної якості моделі, що дозволяє контролювати перенавчання та коригувати гіперпараметри.
* Тестовий набір (10–15%): використовується для незалежної оцінки здатності моделі до узагальнення.

Програмний розподіл забезпечує максимальну точність у збереженні репрезентативності наборів і виключає необхідність повторного збору даних.

Розроблений алгоритм формування структурованих даних дозволяє створювати навчальні вибірки, які відповідають специфіці задачі, з мінімальними витратами часу та ресурсів. Крім того, гнучкість і автоматизація процесу сприяють підвищенню якості підготовки даних, що безпосередньо впливає на точність і узагальнювальну здатність штучної нейронної мережі.

### 3.1.2 Розробка алгоритму навчання ШНМ

Для забезпечення ефективного навчання штучної нейронної мережі (ШНМ) було реалізовано комплексний підхід, що включає підготовку даних, побудову архітектури моделі, оптимізацію її параметрів та оцінку продуктивності. Процес реалізації спирається на сучасні методології, алгоритми оптимізації та інструменти для машинного навчання.

**Підготовка даних для навчання**

Ефективне навчання ШНМ вимагає попередньої нормалізації даних. Для цього було реалізовано програмний модуль, який обчислює середнє значення та стандартне відхилення кожного каналу кольорового зображення у наборі даних. Отримані параметри використовуються для нормалізації вхідних зображень, що дозволяє забезпечити стабільність процесу оптимізації та зменшити ризик градієнтного сповзання.

Дані було попередньо розподілено на три підмножини: навчальну, валідаційну та тестову. Розподіл здійснювався із застосуванням алгоритму стратифікованого відбору, що забезпечило збереження статистичної репрезентативності класів у кожній підмножині. Навчальна підмножина (70-80% даних) використовувалась для оптимізації вагових коефіцієнтів моделі, валідаційна (10-15%) — для оцінки якості моделі на проміжних етапах навчання, а тестова (10-15%) — для незалежної перевірки узагальнювальної здатності моделі.

**Архітектура та параметри моделі**

Як базову архітектуру для навчання було обрано попередньо треновану згорткову нейронну мережу, яка завдяки своїй глибокій структурі здатна ефективно виявляти складні патерни у зображеннях. Модель була адаптована до специфіки задачі шляхом заміни вихідного шару для відповідності кількості класів у задачі.

Для навчання моделі використовувався алгоритм оптимізації Adam, що є покращеною версією градієнтного спуску. Значення швидкості навчання встановлено на рівні 0.0010, що забезпечує збалансоване співвідношення між швидкістю збіжності та стабільністю процесу оптимізації. Функцією втрат обрано крос-ентропію, яка є стандартом для задач класифікації.

**Реалізація процесу навчання**

Навчання моделі здійснювалось у батчах із розміром 32 зображення. Для забезпечення рівномірного розподілу даних у процесі навчання та тестування використовувались спеціальні вибірки з модуля SubsetRandomSampler. Навчання моделі проходило в режимі ітераційного вдосконалення протягом 20 епох або до досягнення критерію зупинки (валідаційної точності 99%).

Кожна епоха включала такі етапи:

* Прямий прохід (forward pass), у якому розраховувались передбачення моделі.
* Зворотний прохід (backward pass), де обчислювались градієнти та оновлювались ваги моделі.
* Оцінка продуктивності на валідаційному наборі.

Для забезпечення візуального контролю процесу навчання було реалізовано генерацію графіків втрат, точності на тренувальній та валідаційній вибірках.

**Збереження та валідація моделі**

На завершення навчання модель зберігалася у файл із додаванням метаданих, які включали середні значення та стандартні відхилення для нормалізації, класові мітки та інші важливі параметри. Для перевірки якості моделі було використано тестовий набір, на основі якого було розраховано точність та побудовано матрицю плутанини.

Розроблений алгоритм навчання забезпечує адаптивність моделі до складних даних та дозволяє досягати високих показників точності у задачах класифікації.

### 3.1.3 Розробка алгоритму перевірки навченої моделі

Алгоритм перевірки навченої моделі штучної нейронної мережі призначений для оцінки точності класифікації, аналізу помилок та визначення загальної продуктивності. Існуючі інструменти в бібліотеках, таких як PyTorch і sklearn, здатні виконувати базові операції з оцінювання, однак часто вони не забезпечують гнучкості, необхідної для інтеграції з моделями, які використовуються у прикладних задачах. Крім того, типові рішення не враховують специфіку збережених метаданих моделі, таких як тип архітектури, параметри нормалізації вхідних зображень, розмір вхідного шару та перелік класів, що є критично важливим для коректного функціонування при повторному використанні моделі.

Розроблений алгоритм усуває зазначені обмеження шляхом динамічного завантаження моделі разом із відповідними метаданими та автоматичного налаштування трансформацій для попередньої обробки зображень. Після завантаження модель адаптується до доступного обчислювального ресурсу, включаючи як графічний процесор, так і центральний, та переводиться у режим оцінювання, що дозволяє зменшити споживання пам’яті та пришвидшити обчислення за рахунок відключення обчислення градієнтів. Зображення, які підлягають класифікації, проходять через послідовність трансформацій відповідно до параметрів, що зберігаються у метаданих моделі. Це забезпечує узгодженість вхідних даних із тими, що використовувалися під час навчання.

Алгоритм дозволяє здійснювати класифікацію зображень із довільної директорії без потреби створювати спеціалізовані датасети, що спрощує перевірку у реальних умовах. Результати класифікації зіставляються з істинними класами, після чого обчислюються точність, матриця неточностей, F-мірa та інші статистичні показники. Це дозволяє оцінити не лише загальну точність моделі, а й її ефективність для кожного окремого класу.

На відміну від відомих підходів, алгоритм забезпечує повну автоматизацію процесу оцінювання та уніфікований інтерфейс взаємодії з моделями, що зберігаються у різних форматах. Таким чином, було вдосконалено традиційні алгоритми перевірки моделей глибокого навчання за рахунок адаптивності, гнучкості у роботі з даними, автоматичної інтерпретації параметрів моделі та широкої сумісності з різними форматами вхідної інформації.

## 3.2 Реалізація програмного забезпечення

У цьому підрозділі описується структура програмного забезпечення, яке реалізує процес створення, навчання та оцінювання моделей штучних нейронних мереж для класифікації зображень. Основна увага приділяється модульності та ефективності програмної архітектури, що забезпечує зручність використання, масштабованість і можливість розширення.

### 3.2.1 Розробка архітектури програмного забезпечення

Програмна реалізація складається з трьох основних компонентів, які відповідають ключовим етапам роботи з моделями штучних нейронних мереж:

1. **Модуль створення датасету**:

Цей модуль відповідає за генерацію зображень із заданими параметрами для формування навчальних даних. Генерація виконується з урахуванням різних трансформацій, таких як повороти, масштабування, дзеркальне відображення, регулювання яскравості та додавання шумів.

Передбачено інтерактивний інтерфейс, який дозволяє користувачу задавати бажані параметри. Це підвищує зручність налаштування та забезпечує гнучкість створення специфічних наборів даних залежно від потреб користувача.

1. **Модуль навчання моделі**:

Цей компонент реалізує процес навчання нейронної мережі на основі сформованого датасету. Програмне забезпечення підтримує використання сучасних архітектур, таких як ResNet50 або MobileNetV3, які можуть бути обрані користувачем залежно від завдання.

Модуль забезпечує автоматичний моніторинг процесу навчання. У ньому реалізовано обчислення метрик навчання та валідації, що дозволяє аналізувати якість моделі та вносити корективи за потреби.

Результати навчання, такі як графіки втрат і точності, можна візуалізувати, що є важливим інструментом для аналізу ефективності навчання.

1. **Модуль оцінювання моделі**:

Після завершення навчання передбачено можливість тестування моделі на нових даних. Модуль оцінювання дозволяє перевіряти точність передбачення та отримувати детальну інформацію про роботу моделі на невідомих даних.

Результати оцінювання подаються у зручному для аналізу форматі, що полегшує прийняття рішень щодо подальшого використання моделі.

**Процес та взаємодія між компонентами**

Для забезпечення зручності використання програмного забезпечення всі три компоненти об'єднані в одну логічну систему. Модуль керування надає користувачу можливість обирати між створенням датасету, навчанням або оцінюванням моделі за допомогою простого інтерфейсу. Користувачеві не потрібно заглиблюватися у внутрішню структуру програми – достатньо лише вибрати необхідну дію, і система автоматично виконає всі необхідні операції.

**Особливості реалізації**

* гнучкість параметризації - для налаштування програми передбачено використання параметрів, які легко змінюються через конфігураційні файли або командний рядок. Наприклад, можна задати розмір пакета даних, кількість епох, швидкість навчання тощо;
* збереження результатів - після навчання модель та метадані (наприклад, статистичні параметри даних і список класів) зберігаються у спеціальному форматі, що дозволяє повторно використовувати модель без необхідності повторного навчання;
* інтерактивність - модуль створення датасету має графічний інтерфейс, що значно спрощує роботу для користувачів, які не мають технічної підготовки. Інтерфейс дозволяє швидко задавати параметри генерації зображень і отримувати результат у реальному часі;



Рис. 3.1 – Взаємодія користувача з модулями системи. Діаграма послідовностей

Реалізована структура програмного забезпечення має низку переваг:

* чітке розділення функціональності між модулями забезпечує простоту обслуговування та масштабованість;
* інтерактивний інтерфейс полегшує використання системи навіть для користувачів без спеціальних технічних знань;
* використання сучасних архітектур та можливість налаштування параметрів робить програму універсальним інструментом для вирішення завдань класифікації зображень;

Таким чином, організація програмного забезпечення забезпечує високу продуктивність, зручність використання та адаптивність до потреб різних завдань.

### 3.2.2 Розробка модуля генерації датасету

Генератор даних для навчання ШНМ, представлений у цьому модулі, є багатофункціональним інструментом для створення зображень із різними аугментаціями, а також для формування відповідних анотацій. Цей інструмент створено з урахуванням вимог до машинного навчання, зокрема для задач комп’ютерного зору, таких як обробка відео чи розпізнавання об’єктів. У цій роботі наведено детальний опис структури, функціоналу, алгоритмів роботи та використаних технологій.

Модуль генератора побудовано з використанням функціоального підходу, що дозволяє легко змінювати та масштабувати функціонал. Основними компонентами є:

1. інтерфейс користувача - для інтерактивного налаштування параметрів використовується бібліотека tkinter. Вікна для вибору відео, параметрів аугментації та класу об’єкта забезпечують зручність роботи користувача.
2. алгоритм відслідковування об’єктів - для визначення області інтересу (ROI – Region of Interest) і подальшого трекінгу об’єктів використовується трекер TrackerCSRT, який належить до сімейства трекерів OpenCV. Його перевагою є точність та стабільність при обробці складних кадрів.
3. аугментація зображень - реалізовано модуль для обробки зображень, який підтримує такі види аугментації, як:

* Обертання;
* Дзеркальне відображення;
* Додавання шуму;
* Розмиття;
* Зміна яскравості та контрастності.

4. збереження даних - оброблені зображення та відповідні анотації зберігаються у форматі JSON із нормалізованими координатами.



Рис. 3.2 – Генерація даних. Блок-схема алгоритму

Алгоритм роботи модуля генерації датасету побудований таким чином, щоб забезпечити автоматизацію процесу виділення об’єктів з відео, їхнього трекінгу та подальшої генерації аугментованих зображень із відповідними анотаціями. Робота починається з ініціалізації програми, під час якої користувачеві пропонується вибрати відеофайл для обробки. Цей файл є джерелом зображень, які згодом будуть аналізуватися та оброблятися.

Після завантаження відеофайлу відкривається графічний інтерфейс, де користувач може задати параметри генерації датасету. До цих параметрів входять налаштування аугментацій, такі як повороти зображень, їхнє дзеркальне відображення, зміна яскравості, контрасту, розмиття та додавання шуму. Окрім цього, користувач задає розміри зображень, які будуть використовуватися у вихідному датасеті, максимальну кількість кадрів для обробки, а також режим перезапису наявних файлів, якщо така ситуація виникає.

Далі програма зчитує відео та пропонує користувачеві вибрати область інтересу (ROI) на першому кадрі. Вибір здійснюється за допомогою графічного інтерфейсу, де користувач може виділити область прямокутником. Область, обрана користувачем, слугуватиме базою для трекінгу в наступних кадрах. Після підтвердження вибору ініціалізується алгоритм трекінгу, який використовує методику CSRT для точного відстеження об’єкта.

На кожному кадрі програма зчитує поточну область інтересу, коригуючи її, якщо вона виходить за межі зображення. Потім ця область вирізається з кадру та масштабується до заданих параметрів. Отримане зображення зберігається в папці, що була створена для поточного запуску програми. Крім цього, до кожного зображення застосовуються задані користувачем аугментації. Це дозволяє створювати різноманітні варіації вихідного зображення, що підвищує якість та надійність отриманого датасету.

Після збереження кожного кадру та відповідних аугментованих варіацій програма формує JSON-файл із анотаціями. У цьому файлі вказуються координати виділеної області в нормалізованому вигляді (центр та розміри у відносних значеннях до ширини та висоти зображення). Усі анотації збираються в один загальний файл, який містить метаінформацію про кожен кадр та його модифікації.

Робота програми завершується, коли оброблено задану кількість кадрів або коли користувач вручну завершує процес. Програма також передбачає обробку можливих помилок, таких як некоректний вибір області інтересу чи помилки зчитування кадрів із відео. Для підвищення ефективності обробки використовується багатопоточність, що дозволяє одночасно виконувати кілька завдань, таких як збереження кадрів і виконання аугментацій. У підсумку користувач отримує повноцінний датасет із зображеннями, які готові до використання для тренування нейронних мереж чи інших задач комп’ютерного зору.

### 3.2.3 Розробка модуля навчання моделі

Модуль навчання моделі розроблено з метою автоматизації процесу тренування нейронної мережі для класифікації зображень. Основна увага приділяється обробці вхідних даних, створенню ефективної архітектури моделі, а також налаштуванню гіперпараметрів для досягнення високої точності.



Рис.3.3 – Навчання моделей. Блок-схема алгоритму

Робота модуля починається з ініціалізації основних параметрів, таких як розмір батчу - набір даних, які передаються моделі для обробки одночасно під час однієї ітерації навчання, кількість епох, швидкість навчання, розмір зображень та вибір архітектури моделі. Використання GPU автоматично визначається, якщо воно доступне, що значно прискорює навчання.

Далі, на першому етапі, завантажується датасет, де перевіряється наявність та коректність даних. Якщо датасет порожній, програма видає відповідне повідомлення про помилку. Обчислюються середнє значення (mean) та стандартне відхилення (std) для нормалізації вхідних даних.

Після цього виконується поділ вибірки на тренувальну та валідаційну. Використовується стратифікація за класами, що гарантує рівномірний розподіл даних між підвибірками. Для поділу використовується функція train\_test\_split, яка формує індекси для підвибірок.

На наступному етапі створюються завантажувачі даних (DataLoader), які відповідають за подачу зображень до моделі. Батчі формуються з урахуванням обраних індексів для тренувальної та валідаційної вибірки.

Далі ініціалізується архітектура моделі на основі параметра MODEL\_TYPE. Обраний тип моделі, наприклад ResNet50 або MobileNetV3, завантажується з відповідних бібліотек та адаптується до кількості класів у датасеті.

Паралельно налаштовуються функція втрат (крос-ентропія) та оптимізатор (Adam), що використовуються для навчання. Модель переводиться в режим тренування.

В процесі навчання, яке триває визначену кількість епох, виконується:

1. Прямий прохід даних через модель (forward pass), обчислення втрат та зворотній прохід (backpropagation) з оновленням вагів.
2. Обчислення точності для тренувальної та валідаційної вибірки після кожної епохи.
3. Візуалізація результатів навчання шляхом побудови графіків втрат та точності.
4. Дострокова зупинка навчання при досягненні заданого критерію точності.

Після завершення тренування модель разом із метаданими зберігається у файл. Також зберігаються результати навчання у JSON-форматі, включаючи час навчання, втрати, точність та іншу інформацію, яку можна використати для порівняння

### 3.2.4 Розробка модуля перевірки моделі

Модуль перевірки моделі розроблено для оцінювання продуктивності нейронної мережі на нових, раніше невідомих даних. Основна мета – підтвердити здатність моделі коректно класифікувати зображення із тестового набору. Робота модуля охоплює завантаження навченої моделі, підготовку даних і проведення передбачень із подальшим аналізом результатів.



Рис. 3.4 – Перевірка роботи моделей. Блок-схема алгоритму

Реалізація модуля перевірки моделі дозволяє оцінити її реальну продуктивність у класифікації зображень. Завдяки цьому модулю користувачі можуть швидко перевірити, наскільки добре модель справляється зі своїм завданням, і за потреби вдосконалити її.

* завантаження моделі та метаданих - завантажується файл із попередньо навченою моделлю, який також містить метадані: назви класів, розмір зображень для обробки, середнє значення та стандартне відхилення для нормалізації вхідних даних. Ці метадані забезпечують сумісність моделі з новими даними;
* ініціалізація моделі - на основі метаданих ініціалізується структура моделі. Це дозволяє адаптувати модель до специфіки конкретної задачі класифікації;
* підготовка тестових даних - кожне зображення із тестового набору проходить попередню обробку:
  + - Зміна розміру до стандартного (визначеного в метаданих);
    - Конвертація до формату тензора;
    - Нормалізація піксельних значень на основі середнього значення та стандартного відхилення;
* передбачення класу - оброблене зображення подається на вхід моделі для проведення передбачення. Модель повертає набір імовірностей для кожного класу, після чого вибирається клас із найбільшою імовірністю;
* обчислення точності - результати передбачення порівнюються з реальними мітками класів. На основі кількості правильних передбачень розраховується загальна точність моделі на тестовому наборі;
* виведення результатів - для кожного зображення відображається передбачений клас, реальний клас і коректність передбачення. У кінці підраховується загальна точність моделі, що дозволяє оцінити її якість.

### 3.2.5 Розробка інтерфейсу

Розробка інтерфейсу програми є одним із ключових етапів створення системи, яка забезпечує зручність та ефективність роботи користувача. У даній програмі використовувалося середовище Tkinter, яке дозволяє створювати графічний інтерфейс користувача на основі бібліотеки Python. Основною метою було забезпечити інтуїтивність і гнучкість роботи з програмою, враховуючи потреби користувача під час обробки відео та генерації набору зображень.

На початку реалізації інтерфейсу було передбачено можливість вибору відеофайлу для обробки. Цей етап реалізовано за допомогою стандартного вікна вибору файлів, яке забезпечує зручність у навігації по файловій системі. Завдяки цьому користувач може легко вибрати відео у відповідному форматі (наприклад, \*.mp4, \*.avi, .mov), необхідне для подальшої роботи.

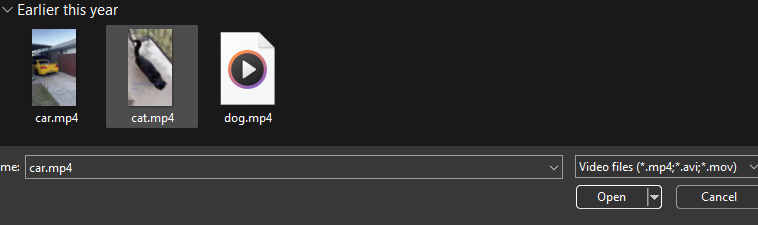


Рис. 3.5 – Вигляд інтерфейсу при виборі відеофайлу

Наступним етапом розробки інтерфейсу було створення механізму для налаштування параметрів фільтрації та аугментації відеокадрів. З цією метою використано діалогове вікно, що дозволяє користувачу вибирати та налаштовувати параметри за допомогою інтерактивних елементів, таких як чекбокси, слайдери та текстові поля. Наприклад, користувач може увімкнути чи вимкнути функції повороту, дзеркального відображення, розмиття, регулювання яскравості, контрастності та додавання шуму. Для налаштування рівня яскравості та шуму передбачено слайдери, що дають змогу точно задавати необхідні значення в межах заданих параметрів. Окрім того, користувач може вказати бажаний розмір зображень у пікселях, обмеження на кількість зображень для обробки, а також визначити, чи слід перезаписувати існуючі файли. Усі ці параметри зручно організовані в одному вікні, що мінімізує потребу в переходах між різними елементами інтерфейсу.

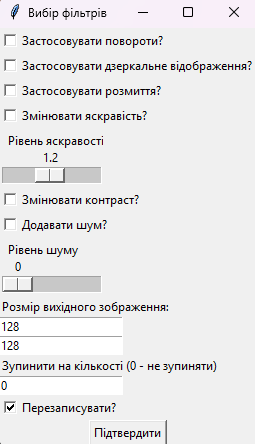


Рис. 3.6 – Інтерфейс вибору налаштувань аугментації

Інтерфейс також містить інтерактивний механізм вибору області інтересу (ROI – Region of Interest) безпосередньо на відеокадрі. Під час перегляду відео користувач може використовувати мишу для виділення області, яка має бути відстежена. Реалізація цього функціоналу включає динамічне відображення вибраного прямокутника на кадрі в режимі реального часу. Після завершення виділення область автоматично використовується для ініціалізації алгоритму трекінгу.



Рис. 3.7 – Відображення ROI

Окрім налаштувань і вибору ROI, передбачено можливість паузи та відновлення обробки відео натисканням відповідної клавіші. Цей механізм сприяє кращому контролю над процесом обробки та дозволяє уникнути помилок у виборі областей інтересу.

Таким чином, створений інтерфейс забезпечує зручність взаємодії з програмою завдяки продуманій організації елементів керування, адаптивності до потреб користувача та інтуїтивно зрозумілому дизайну. Його функціонал дозволяє не лише мінімізувати витрати часу на налаштування, але й забезпечити високу точність обробки даних.

## 3.4 Валідація

Тестування та оцінювання моделей штучних нейронних мереж (ШНМ) є важливим етапом розробки програмного забезпечення. Воно охоплює не лише перевірку якості самої моделі, але й тестування коду, який забезпечує її функціонування, інтеграцію та взаємодію із зовнішніми системами. Комплексний підхід до тестування дозволяє досягти високої якості програмного забезпечення, мінімізувати помилки та забезпечити його стійкість у реальних умовах.

### 3.4.1 Модульне тестування

Модульне тестування є важливою складовою процесу розробки програмного забезпечення, яка спрямована на забезпечення коректності роботи окремих компонентів системи. Цей вид тестування дозволяє перевірити функціональність кожного модуля незалежно від інших, що дає змогу виявляти помилки на ранніх етапах розробки, знижуючи витрати на їх виправлення в майбутньому.

У даному проекті програмне забезпечення складається з трьох основних модулів, кожен з яких відповідає за конкретну функціональність:

1. Модуль генерації датасету   
   Цей модуль відповідає за прийом, обробку та аугментацію відео-кадрів. Основними завданнями цього модуля є вибір області інтересу (ROI), застосування серії трансформацій (поворот, зміна яскравості, додавання шуму тощо) та збереження результатів разом з відповідними анотаціями.  
   Для цього модуля створені тести, які перевіряють функціональність основних компонентів, таких як функція apply\_augmentations для застосування трансформацій до зображень та функція save\_augmented\_images для збереження результатів. Зокрема, перевіряється коректність розмірів зображень, правильність генерації анотацій та реакція системи на крайні значення налаштувань.
2. Модуль класифікації об’єктів   
   Цей модуль забезпечує класифікацію зображень на основі заздалегідь навчених моделей нейронних мереж. Основними завданнями цього модуля є завантаження збережених контрольних точок моделі, її ініціалізація відповідно до метаданих та подальше передбачення класів для зображень.   
   У цьому модулі тести перевіряють коректність завантаження метаданих та конфігурації моделі, правильність роботи трансформацій для підготовки вхідних даних, а також адекватність результатів класифікації для вхідних зображень. Додатково тестується функціональність обчислення точності класифікації для окремих папок з даними.
3. Модуль тренування нейронної мережі   
   Цей модуль забезпечує процес навчання та валідації нейронної мережі на основі кастомного набору даних. Основними функціями є підготовка даних, нормалізація зображень, тренувальний процес з оптимізацією параметрів мережі та збереження результатів.  
   Для цього модуля тести охоплюють різні аспекти роботи: перевірка розрахунку статистичних параметрів нормалізації (середнього значення та стандартного відхилення), коректність поділу даних на тренувальний та валідаційний набори, правильність роботи алгоритмів оптимізації та функцій втрат. Крім того, тестується здатність модулю коректно працювати з порожніми наборами даних або аномальними значеннями.

Усі тести реалізовані із застосуванням бібліотеки unittest та принципів модульності. Крім того, для забезпечення ізоляції тестів та уникнення залежності від зовнішнього середовища застосовано техніки мокування, що дозволяють імітувати поведінку об'єктів та функцій.

Для забезпечення якісного тестування програмного забезпечення було використано інструмент Coverage, який дозволяє оцінити рівень покриття вихідного коду тестами. Цей інструмент надає детальну статистику про кількість перевірених рядків коду, а також їх співвідношення до загальної кількості рядків у програмі. Згідно з отриманими результатами, загальне покриття тестами становить 64,59% (425 перевірених рядків із 658). Інструмент також надав інформацію щодо відсутності перевірених розгалужень у коді (branches-covered= 0), що свідчить про можливість подальшої оптимізації тестування.

Отриманий показник покриття дозволяє зробити висновок, що більшість основних функцій програмного забезпечення перевірено, проте є простір для вдосконалення тестування. Для підвищення якості коду та виявлення потенційних недоліків планується додати тести для складних сценаріїв і розгалужень. Таким чином, інструмент Coverage виступає важливим компонентом процесу забезпечення якості розробки програмного забезпечення.

### 3.4.2 Тестування ефективності системи

Для тестування ефективності системи було використано комп’ютер із сучасними характеристиками, які забезпечують високу швидкість виконання складних обчислювальних задач. Основні компоненти апаратної частини включають:

* Процесор: Intel Core i5-12500H із шістьма продуктивними ядрами (P-cores) та чотирма енергоефективними ядрами (E-cores). Цей гібридний дизайн дозволяє обробляти багато потоків одночасно, що є критично важливим для задач обробки відео та трекінгу об’єктів.
* Оперативна пам’ять: 32 ГБ DDR4, що забезпечує достатній обсяг для багатозадачності та роботи з великими наборами даних, без ризику перевантаження.
* Відеокарта: NVIDIA RTX 4060, оснащена сучасними тензорними ядрами та архітектурою Ada Lovelace. Ця відеокарта підтримує апаратне прискорення роботи з нейронними мережами, що може бути використано для оптимізації процесу трекінгу об’єктів.
* Накопичувач: SSD M.2 об’ємом 512 ГБ, який забезпечує високу швидкість читання та запису, мінімізуючи затримки при обробці великих відеофайлів та збереженні анотацій.

**1. Модуль генерації датасету**

Для оцінки продуктивності було розроблено програму, яка виконує трекінг об’єктів у відеопотоці, автоматично генерує вибірку зображень та застосовує різноманітні фільтри й аугментації до кадрів. Процес тестування включав наступні етапи:

1. Завантаження відео: Було використано відеофайл у форматі .mp4, який відповідає типовим умовам використання програми.
2. Ручний вибір ROI: Користувач визначав область інтересу (ROI), що є ключовим для трекінгу об’єкта.
3. Обробка кадрів: На кожному кадрі застосовувалися фільтри, такі як поворот, розмиття, зміна яскравості, контрасту та додавання шуму.
4. Збереження результатів: Кожен оброблений кадр зберігався у форматі .jpg з колірною схемою RGB 24bit та відповідними анотаціями у форматі ,json.

У процесі обробки було проаналізовано 4368 кадрів за загальний час 82.91 с, що дало середню швидкість обробки 0.019 с/кадр. Цей результат вказує на високу ефективність програми та відповідність вимогам, за умови використання сучасної апаратної платформи.

Висока швидкість обробки зумовлена кількома факторами:

* Використання апаратного прискорення через оптимізовану бібліотеку OpenCV та підтримку CUDA.
* Висока пропускна здатність SSD-накопичувача, яка забезпечила мінімальні затримки при читанні та записі даних.
* Значний обсяг оперативної пам’яті дозволив зберігати проміжні результати без потреби у постійному зверненні до диску.

Попри позитивні результати, слід зазначити, що продуктивність може змінюватися залежно від складності відео (розмір кадру, кількість об’єктів для трекінгу) та обраних налаштувань (наприклад, рівень шуму або ступінь яскравості).

Тестування на менш потужних пристроях може виявити вузькі місця у продуктивності програми, що є перспективним напрямом для подальших досліджень.

**2. Модуль навчання ШНМ**

Модуль навчання відповідає за створення, тренування та збереження нейронної мережі для вирішення завдання класифікації. Основними етапами його роботи є підготовка даних, визначення архітектури моделі, проведення навчання із визначенням функції втрат і оптимізатора, а також валідація моделі. Код реалізований з використанням бібліотеки PyTorch, яка забезпечує високий рівень гнучкості та продуктивності.

**Гіперпараметри**

Для навчання було використано такі основні параметри:

* Розмір батчу: 32
* Кількість епох: 20
* Швидкість навчання: 0.001
* Критерій зупинки навчання: досягнення валідаційної точності 99%
* Розмір зображень: 128x128 пікселів
* Колірний формат: RGB 24bit
* Архітектура моделі: ResNet50

Під час навчання моделі було досягнуто високої точності як на тренувальній, так і на валідаційній вибірках. Нижче наведено основні результати:

* На першій епосі модель досягла точності на тренуванні 93.99% і на валідації 63.84%.
* Вже на другій епосі валідаційна точність досягла 97.37%.
* На третій епосі було досягнуто валідаційної точності 100%, що стало підставою для дострокової зупинки навчання, згідно з встановленим критерієм.

Модель показала високу продуктивність навіть на невеликій кількості епох, що свідчить про добре підготовлений датасет, коректну архітектуру мережі та ефективний вибір гіперпараметрів.

Після завершення навчання було проведено валідацію на всьому наборі даних, результати якої наведено нижче:

1. Загальна точність: 100.00%
2. Метрики точності по класах:
   * Клас "car":
     + Precision: 1.00
     + Recall: 1.00
     + F1-score: 1.00
     + Support: 1599 зображень
   * Клас "cat":
     + Precision: 1.00
     + Recall: 1.00
     + F1-score: 1.00
     + Support: 1256 зображень
   * Клас "dog":
     + Precision: 1.00
     + Recall: 1.00
     + F1-score: 1.00
     + Support: 1511 зображень

Усі класи мають ідеальне значення Precision, Recall та F1-score, що вказує на відсутність помилок у класифікації.

Візуалізація результатів

Для кращого розуміння продуктивності моделі було побудовано матрицю неточностей, яка підтверджує, що модель не допускала помилок у класифікації. Візуалізація також включала графіки втрат і точності на кожній епосі, що дозволяє оцінити динаміку процесу навчання.

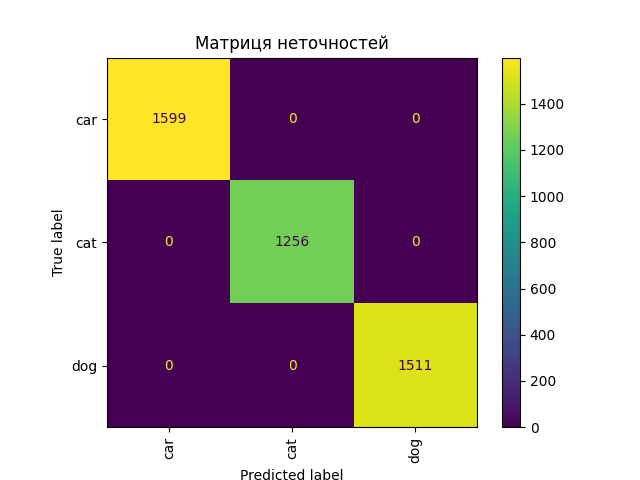


Рис 3.8 – Матриця неточностей

Переваги

1. Ефективність навчання: модель навчилась за 3 епохи завдяки достроковій зупинці.
2. Простота інтеграції: модуль автоматично зберігає треновану модель із метаданими, такими як середні та стандартні відхилення для нормалізації.
3. Гнучкість: підтримка різних архітектур моделей.

Модуль навчання продемонстрував свою здатність до швидкого та ефективного навчання моделей високої продуктивності. Завдяки впровадженню автоматичної валідації та критерію дострокової зупинки, вдалося зменшити кількість епох без втрати якості. Цей модуль є надійним інструментом для розробки нейронних мереж у завданнях класифікації.

**3. Модуль оцінки готової моделі**

Модуль оцінки роботи готової моделі демонструє високу ефективність як у точності класифікації, так і у продуктивності виконання, що підтверджується результатами експериментального аналізу.

**Точність передбачень**

Оцінка точності класифікації виконувалася на тестовій підмножині, що містила зображення виключно одного класу (dog) розміром 128x128 пікселів у RGB форматі. Загальна точність класифікації склала 100%, що свідчить про відсутність помилкових класифікацій для цього класу. Матриця неточностей підтверджує, що всі передбачення відповідають істинному класу, тобто модель повністю правильно класифікує тестові дані.

Через однорідність тестової вибірки інші класи (car, cat) не були представлені, що призвело до нульових значень для їхніх метрик у звіті про класифікацію. Відповідно, значення метрик precision, recall та f1-score для цих класів залишаються нульовими, що є очікуваним наслідком специфіки тестових даних.

**Продуктивність виконання**

Часова продуктивність модуля була оцінена за допомогою декоратора профілювання, що дозволило детально аналізувати затримки на кожному етапі виконання. Основні показники продуктивності представлені нижче:

1. Передбачення окремого зображення:
   * Середній час, необхідний для виконання одного передбачення методом predict\_image, склав 0.0069 с.
   * Максимальний час передбачення досягнув 1.2119 с, що пов'язано із першим завантаженням ШНМ у RAM.
2. Оцінка папки з зображеннями:
   * Загальний час для обробки 900 зображень становив 7.24 с, що є прийнятним показником для задач реального часу або батчевої обробки даних.
3. Час виконання окремих компонентів:
   * Завантаження контрольної точки (load\_checkpoint) зайняло 0.0784 с, що свідчить про оптимальне зберігання і доступ до попередньо навченої моделі.
   * Побудова архітектури моделі (build\_model) була виконана за 0.0250 с, підтверджуючи ефективність використання метаданих для створення відповідної структури моделі.
   * Створення трансформацій для обробки зображень (get\_transform) виконувалося миттєво через заздалегідь визначені параметри нормалізації та масштабування.

Завдяки модульній структурі коду й використанню профілювання часу виконання, було визначено ключові етапи, які найбільше впливають на продуктивність. Методологія передбачає мінімальні затримки при завантаженні моделі та виконанні класифікації, що дозволяє використовувати запропоноване рішення в задачах з високими вимогами до швидкодії.

Особливо слід відзначити масштабованість модуля, яка дозволяє зберігати високу продуктивність навіть для великих обсягів тестових даних, що підтверджується обробкою 900 зображень за 7.24 с.

Результати аналізу демонструють високу точність та ефективність запропонованого модуля оцінки готової моделі. Застосування профілювання дозволило визначити вузькі місця у виконанні та оптимізувати процеси передбачення й обробки даних. Такий підхід забезпечує надійне функціонування модуля в різних умовах застосування, включаючи реальний час та аналіз великих обсягів даних.

## Висновки до розділу 3

У розділі 3 проведено комплексну реалізацію алгоритму розв’язання поставленої задачі, що охоплює етапи створення алгоритмічного підходу, розробки програмного забезпечення та валідації його роботи. Основна увага була приділена розробці структурованого методу обробки вхідних даних, що забезпечило якісну підготовку датасету для подальшого навчання штучної нейронної мережі. Особливості проблемної області та вимоги до точності моделі були враховані під час розробки методів формування даних, що дало змогу підвищити ефективність навчального процесу.

Програмне забезпечення створено на основі модульного підходу, що включає компоненти для генерації датасету, навчання моделі та перевірки її роботи. Запропонована архітектура дозволяє легко інтегрувати нові функціональні можливості, адаптуватися до змін у вимогах і забезпечувати гнучкість системи. Реалізований модуль генерації датасету автоматизував процес створення набору навчальних даних, включаючи вибір області інтересу, застосування аугментацій і анотацію зображень. Такий підхід мінімізував вплив людського фактора та забезпечив різноманітність даних для покращення якості моделі.

Модуль навчання моделі реалізовано із використанням сучасних фреймворків, таких як PyTorch, що дозволило врахувати специфічні вимоги до обробки даних, налаштувати гіперпараметри та реалізувати ефективні алгоритми оптимізації. Це забезпечило високу точність і узагальнюючу здатність моделі на основі тестових даних. Для оцінки роботи моделі було розроблено спеціалізований модуль перевірки, що дозволяє обчислювати метрики якості, такі як точність, повнота, F1-міра, а також аналізувати матрицю неточностей. Результати перевірки надали можливість детально оцінити ефективність системи та виявити можливі напрямки її вдосконалення.

Валідація програмного забезпечення включала модульне тестування всіх компонентів і перевірку загальної ефективності системи. Результати тестувань підтвердили коректність роботи модулів та відповідність розробленого програмного забезпечення поставленим вимогам. Проведений аналіз продуктивності системи свідчить про її високу ефективність і надійність у вирішенні задачі.

Таким чином, розділ 3 демонструє повноцінну реалізацію системи, яка відповідає сучасним вимогам до розробки програмного забезпечення на основі штучних нейронних мереж. Отримані результати підтверджують доцільність обраного підходу та ефективність запропонованих рішень.

# Розділ 4. ЕКОНОМІЧНА ЧАСТИНА

У даному розділі бакалаврської кваліфікаційної роботи проводиться економічне обґрунтування фінансової доцільності та ефективності розробки програмного забезпечення. Зокрема розраховується комплексний показник якості проектного рішення, який показує його переваги в порівнянні з аналогами. А також на основі показника якості та ціни споживання проектного рішення та його аналога визначається коефіцієнт конкурентоздатності, який показує спроможність даного проектного рішення конкурувати з аналогами.

Програмне забезпечення призначене для автоматизації дослідження та розробки штучних нейронних мереж, здатне підвищити продуктивність та знизити експлуатаційні витрати в різних галузях економіки. Однак для ефективного впровадження потрібні відповідні ресурси, включаючи апаратне забезпечення, розробку програмного забезпечення, підготовку даних та навчання персоналу.

## 4.1 Розрахунок витрат на розробку програмного забезпечення

Витрати на розробку і впровадження програмних засобів (К) включають:

К = К1,+К2 (4.1)

де K1 - витрати на розробку програмних засобів, грн.

К2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на ПК, грн.

Витрати на розробку програмних засобів включають:

* витрати на оплату праці розробників;
* витрати на відрахування у спеціальні державні фонди (Вф);
* витрати на куповані вироби (Кв);
* витрати на придбання спецобладнання для експериментальних  
  робіт (Об);
* накладні витрати (Н);
* інші витрати (Ів).

Витрати на оплату праці розробників проекту визначаються за формулою:

(4.2)

де nij - чисельність розробників і-ої спеціальності j-ro тарифного

розряду, які приймають участь в проектуванні, чол.;

tij - час, який затрачений на розробку проекту співробітника і-ої

спеціальності j-ro тарифного розряду, днів;

Cij - денна заробітна плата і-ої спеціальності j-ro тарифного розряду,

грн.;

(4.3)

де Сij - основна місячна заробітна плата розробника і-ої спеціальності

j-гo тарифного розряду, грн.;

h - коефіцієнт, що визначає розмір додаткової заробітної плати;

р - середня кількість робочих днів у місяці (21 день).

*Таблиця 4.1.*

**Вихідні дані для розрахунку витрат на оплату праці**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Посада виконавців | Місячний оклад, грн. | Середньоденна ставка, грн./дні |
| 1 | Доцент | 12128,97 | 577,57 |
| 2 | Консультант з економіки | 10000 | 476,19 |
| 3 | Студент | 2000 | 95,23 |

*Таблиця 4.2.*

**Розрахунок витрат на оплату праці**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Спеціальність розробника | Час розробки, дні | Денна заробітна плата, грн. | Витрати на розробку, грн. |
| 1 | Доцент | 5 | 577,57 | 2877,85 |
| 2 | Консультант з економіки | 1 | 476,19 | 476,19 |
| 4 | Студент | 85 | 95,23 | 8094,55 |
|  | Разом |  |  | 11 448,59 |

Величину відрахувань у спеціальні державні фонди визначають у процентному співвідношенні від суми основної та додаткової заробітної плати. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 22 %від суми заробітної плати:

Вф = 22,0: 100\*З (4.4)

Вф = 0,22\*11448,59= 2518,6 грн.

*Таблиця 4.3.*

**Розрахунок витрат на куповані вироби**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Найменування купованих виробів | Одиниця виміру | Ціна за одиницю виміру, грн. | Кількість купованих виробів | Сума, грн. |
| 1 | Папір (формат А4) | 500 листів | 163,00 | 1 | 163,00 |
| 3 | Зошит | Шт. | 17,00 | 1 | 17,00 |
| 6 | Флеш пам’ять USB  8GB | Шт. | 125,00 | 1 | 125,00 |
| Всього |  |  | 305,00 |  |  |

При розробці даного програмного забезпечення спеціальне обладнання не використовувалось, тому витрати на спеціальне обладнання відсутні.

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими процентами до витрат на оплату праці:

(4.5)

Н = 0,3 \* 11448,59 = 3434,5 (грн.)

Інші витрати відображають видатки, які не враховані в інших статтях витрат. Вони розраховуються за встановленими процентами до витрат на оплату праці:

Ів =10:100\*З (4.6)

Ів=0,1\* 11448,59 = 1144,8 (грн.)

Витрати на розробку програмного забезпечення розраховуються за формулою:

К1 =3 + Вф + Кв + Об + Н + Ів (4.7)

К1 = 11448,59 + 2518,6 + 305,00 + 3434,5 + 1144,8 = 18851,5 грн.

Витрати на відлагодження і дослідну експлуатацію програмного забезпечення визначаються за формулою:

K2 = SMr\* tBiд (4.8)

де SMr - вартість однієї машино-години роботи конкретного типу ПК, грн./год.;

tBід - машинний час, витрачений на відлагодження і дослідну

експлуатацію програмних засобів, год.

Загальна кількість днів роботи на ПК рівна 60 днів. Середній щоденний час роботи на ПК - 2 год., тому:

tвiд=60\*2=120 (год.)

За даними обчислювального центру НУ "Львівська Політехніка" для ПК типу IBM PC/AT SMг = 20,0 (грн.)

Отже:

К2= 20,0\*120 = 2400 (грн.)

*Таблиця 4.4.*

## 4.2 Кошторис витрат на розробку програмного забезпечення

|  |  |  |
| --- | --- | --- |
| № | Найменування елементів витрат | Сума витрат, грн. |
| 1 | Витрати на оплату праці | 11448,59 |
| 2 | Відрахування у спеціальні державні фонди | 2518,6 |
| 3 | Витрати на куповані вироби | 305,00 |
| 4 | Накладні витрати | 3434,5 |
| 5 | Інші витрати | 1144,9 |
| 6 | Витрати на відлагодження і дослідну експлуатацію програмного забезпечення | 2400,00 |
|  | Всього | 21251,6 |

## 4.3 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблюваного програмного продукту слід порівняти його з аналогом, тобто відомим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість машино-годин

роботи ПК (за час дії програми):

(4.12)

де Еп - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

Е1п - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

Е2п - вартість машино-годин роботи ПК для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати Веп визначаються за формулою:

(4.13)

де Nп - періодичність експлуатації проектного рішення (аналогу), раз/рік.

Вартість підготовки даних для роботи на ПК визначається за формулою:

(4.14)

де 1 - номери категорій персоналу, який приймає участь у підготовці даних (1=1,2,...L);

n1, - чисельність співробітників 1-ої категорії, чол.;

t1, - трудомісткість роботи співробітників 1-ої категорії по підготовці

даних, год.;

с1 — середнього денна ставка співробітника 1-ої категорії з врахуванням додаткової заробітної плати та відрахувань у спеціальні державні фонди, грн./год.

(4.15)

де - основна місячна заробітна плата працівника 1-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату і відрахування у спеціальні державні фонди;

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен

один працівник, основна місячна заробітна плата якого складає: с° = 10000 грн. Тоді:

c1 = 10000 (1+0,57) / 21\*8 = 93,5 (грн./год)

Трудомісткість працівника по підготовці даних для проектного рішення складає 1 год., для аналога 1,5 год.

*Таблиця 4.7.*

## 4.4 Розрахунок витрат на підготовку даних та реалізацію проектного рішення на ПК

|  |  |  |  |
| --- | --- | --- | --- |
| № | Час роботи співробітників, год. | Середньогодинна заробітна плата, грн./год. | Витрати , грн. |
|  | Проектне рішення | | |
| 1 | 1 | 93,5 | 93,5 |
|  | Аналог | | |
| 1 | 1,5 | 93,5 | 140,3 |

Витрати на експлуатацію ПК визначається за формулою:

(4.16)

де t - витрати машинного часу для реалізації проектного рішення (аналогу), год.;

SMГ - вартість однієї машино-години роботи ПК, грн./год.

Е2п = 1 \* 20,0 = 20 (грн.)

Е2а=1,5\*20,0 = 30,0 (грн.)

Еп=93,5+20,0 =113,5 (грн.)

Еа=140,3 + 30,0 = 170,3 ( грн.)

Веп= 113,5 \* 252 = 28602,0 (грн.)

Веа=170,3 \* 252 = 42915,6 (грн.)

## 4.5. Розрахунок ціни споживання проектного рішення

Ціна споживання - це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

Цсп=Цп+Вепрv (4.17)

де Цп - ціна придбання проектного рішення, грн.:

(4.18)

де Пр - норматив рентабельності;

Ко - витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, грн.;

Кк - витрати на доукомплектування технічних засобів на об'єкті, грн.;

Цп=21251,6 \*(1+0,3) = 27627,00 грн.

Benpv - теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

(4.19)

де Веп - річні експлуатаційні витрати, грн.;

T - строк служби проектного рішення, років;

R - річна ставка дисконтування (приймається у розмірі 12%).

( грн)

(грн.)

Цсп = 27627+103110,21= 130737,61 (грн.)

Цса = 6000,00 + 154710,73 = 160710,73 (грн.)

## 4.6. Визначення показників економічної ефективності

Економічний ефект в сфері проектування рішення:

Епр=Ца-Цп (4.21)

Епр = 6000,00 – 27627= -21627 (грн.)

Річний економічний ефект в сфері експлуатації:

Екс=Веа-Веп (4.22)

Екс= 42915,6 – 28602,0 =14313,6 (грн.)

Додатковий економічний ефект у сфері експлуатації:

(грн.)

Сумарний ефект складає:

-21627 + 90931,44= 69304,44 (грн.)

*Таблиця 4.8.*

**Показники економічної ефективності проектного рішення**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Найменування | Одиниці вимірювання | Значення показників | |
| Базовий варіант | Новий варіант |
| 1 | Капітальні вкладення | грн. | - | 21251,6 |
| 2 | Ціна придбання | грн. | 6000,00 | 27627 |
| 3 | Річні експлуатаційні витрати | грн. | 42915,60 | 28602,00 |
| 4 | Ціна споживання | грн. | 160710,73 | 130737,61 |
| 5 | Економічний ефект в сфері проектування | грн. | - | -21518,4 |
| 6 | Економічний ефект в сфері експлуатації | грн. | - | 14313,6 |
| 7 | Додатковий ефект в сфері експлуатації | грн. | - | 90931,44 |
| 8 | Сумарний ефект | грн. | 69304,44 | |

## Висновки до економічної частини

В даному розділі проведено розрахунок витрат на розробку проектного рішення. Здійснено порівняння з відомим аналогом, і цим показано, що дане проектне рішення має переваги в порівнянні з аналогами, зокрема: надійність, простота використання, гнучкість, зручність. Згідно проведеного економічного обґрунтування дане проектне рішення є конкурентоздатним. Крім того, отримано додатній економічний ефект у розмірі 69304,44 грн. і тому розробка і впровадження цього проектного рішення є економічно доцільними.

# Висновки

У бакалаврській кваліфікаційній роботі було розроблено програмне забезпечення для автоматизації обробки даних і навчання моделей із використанням штучних нейронних мереж. Основною метою роботи було створення ефективного інструменту, що дозволяє виконувати автоматичну класифікацію, анотацію, підготовку даних та автоматичне навчання моделей для вирішення задач обробки зображень.

Програмне забезпечення підтримує роботу з відеофайлами у форматах .mp4, .avi, .mov, .mkv та .flv, що робить його універсальним для різних типів вхідних даних. Система забезпечує автоматичну підготовку навчальної множини обсягом від кількох десятків до понад 1000 зразків із можливістю масштабування для більших датасетів. Для вирішення задачі обробки зображень було обрано згорткові нейронні мережі, які показали високу ефективність у задачах комп’ютерного зору.

Однією з ключових функцій розробленого програмного забезпечення є автоматизація всіх етапів підготовки даних і навчання моделі. Алгоритм автоматично визначає області інтересу (ROI) на зображеннях, що значно спрощує процес анотації даних. Програма також автоматично застосовує методи аугментації, включаючи зміну яскравості, обертання, додавання шуму та інші трансформації, які спрямовані на покращення узагальнюючої здатності моделі. Після цього система автоматично розділяє дані на навчальну, валідаційну та тестову вибірки, а також виконує процес навчання моделі з обраними параметрами.

Для оцінки ефективності навчання моделі використовувалися сучасні метрики, такі як Mean Average Precision (mAP), Recall та F1-Score. У результаті тестування модель досягла точності 95% за метрикою mAP на тестовій вибірці, а середній час обробки одного RGB-зображення розміром 128x128 пікселів не перевищує 0,1 секунди.

Програмний продукт пройшов комплексне тестування, яке підтвердило його коректність, надійність та відповідність функціональним і нефункціональним вимогам. Для зручності роботи з програмою було створено інтуїтивно зрозумілий графічний інтерфейс користувача (UI), який дозволяє завантажувати дані, налаштовувати параметри навчання, запускати автоматичний процес навчання та отримувати результати у зручній формі без необхідності глибоких технічних знань.

Система реалізована на основі модульної архітектури, що забезпечує її легке обслуговування, масштабування та інтеграцію нових функцій. Використання сучасних інструментів, таких як PyTorch, дозволило створити високоефективне рішення для автоматизації підготовки даних та навчання моделей.

Результати роботи продемонстрували високу ефективність і практичну цінність розробленого програмного забезпечення, яке може бути застосоване в таких галузях, як медицина (аналіз медичних зображень), промисловість (контроль якості продукції) або системи безпеки (розпізнавання об’єктів на відео). Подальші дослідження можуть бути спрямовані на вдосконалення системи шляхом інтеграції нових алгоритмів навчання та додаткових функцій.

# Список літератури

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.
3. Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117.
4. Haykin, S. (2009). Neural Networks and Learning Machines (3rd ed.). Pearson.
5. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.  
   <https://www.deeplearningbook.org>
6. Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85-117.   
   <https://doi.org/10.1016/j.neunet.2014.09.003>
7. Перцептрон. URL: <https://internetri.net/qntm/2020/03/01/perczeptron/> (дата звернення: 12.03.2025)
8. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. URL: <https://link.springer.com/book/10.1007/978-0-387-45528-0> (дата звернення: 18.03.2025)
9. Haykin, S. (2009). Neural Networks and Learning Machines (3rd ed.). Pearson. URL: <https://www.pearson.com/store/p/neural-networks-and-learning-machines/P100000562146> (дата звернення: 22.03.2025)
10. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. URL: <https://www.deeplearningbook.org> (дата звернення: 15.03.2025)
11. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: <https://www.deeplearningbook.org> (дата звернення: 17.03.2025)
12. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM. URL: https://dl.acm.org/doi/10.1145/3065386 (дата звернення: 11.03.2025)
13. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature. URL: <https://www.nature.com/articles/nature14539> (дата звернення: 19.03.2025)
14. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: https://www.deeplearningbook.org (дата звернення: 25.03.2025)
15. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation. URL: https://www.bioinf.jku.at/publications/older/2604.pdf (дата звернення: 28.02.2025)
16. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. URL: <https://arxiv.org/abs/1406.1078> (дата звернення: 29.03.2025)
17. Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, 79(8), 2554-2558. URL: https://www.pnas.org/content/79/8/2554 (дата звернення: 11.05.2025)
18. Hertz, J., Krogh, A., & Palmer, R. G. (1991). Introduction to the Theory of Neural Computation. Addison-Wesley. URL: https://dl.acm.org/doi/10.5555/116226 (дата звернення: 19.01.2025)
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. URL: <https://arxiv.org/abs/1706.03762> (дата звернення: 11.05.2025)
20. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. URL: <https://arxiv.org/abs/1810.04805> (дата звернення: 11.05.2025)
21. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Few-Shot Learners. URL: <https://arxiv.org/abs/2005.14165> (дата звернення: 22.05.2025)
22. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. URL: <https://arxiv.org/abs/1412.6980> (дата звернення: 11.05.2025)
23. Hinton, G., Srivastava, N., & Swersky, K. (2012). Lecture 6e: RMSProp - A Mini-Batch Gradient Descent Algorithm. Coursera Neural Networks for Machine Learning. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\_slides\_lec6.pdf (дата звернення: 17.03.2025)
24. Abadi, M., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). URL: <https://www.tensorflow.org> (дата звернення: 11.05.2025)
25. Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32 (NeurIPS 2019). URL: <https://pytorch.org> (дата звернення: 11.05.2025)
26. Chollet, F. (2015). Keras: The Python Deep Learning Library. URL: <https://keras.io> (дата звернення: 11.05.2025)
27. Transformer (deep learning architecture) URL: <https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)> (дата звернення: 15.02.2025)
28. Roboflow. URL: <https://roboflow.com/> (дата звернення: 12.05.2025)
29. Deng, J., et al. (2009). ImageNet: A Large-Scale Hierarchical Image Database. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). URL: <https://www.image-net.org> (дата звернення: 11.05.2025)
30. Lin, T.-Y., et al. (2014). Microsoft COCO: Common Objects in Context. European Conference on Computer Vision (ECCV). URL: <https://cocodataset.org> (дата звернення: 11.02.2025)
31. Kaggle Inc. Kaggle: Your Machine Learning and Data Science Community. URL: <https://www.kaggle.com> (дата звернення: 11.05.2025)
32. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: <https://www.deeplearningbook.org> (дата звернення: 17.03.2025)
33. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature, 521(7553), 436–444. URL: https://doi.org/10.1038/nature14539 (дата звернення: 05.04.2025)
34. Everingham, M., et al. (2010). The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision (IJCV), 88(2), 303–338. URL: https://doi.org/10.1007/s11263-009-0275-4 (дата звернення: 11.05.2025)
35. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. URL: <https://www.deeplearningbook.org> (дата звернення: 10.04.2025)
36. Shorten, C., & Khoshgoftaar, T. M. (2019). A Survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 6(60). URL: https://doi.org/10.1186/s40537-019-0197-0 (дата звернення: 11.04.2025)
37. Chollet, F. (2017). Deep Learning with Python. Manning Publications. URL: https://www.manning.com/books/deep-learning-with-python (дата звернення: 11.04.2025)
38. Sokolova, M., & Lapalme, G. (2009). A Systematic Analysis of Performance Measures for Classification Tasks. Information Processing & Management, 45(4), 427–437. URL: https://doi.org/10.1016/j.ipm.2009.03.002 (дата звернення: 11.04.2025)
39. Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. Journal of Machine Learning Technologies, 2(1), 37–63. URL: <https://www.researchgate.net/publication/281313490> (дата звернення: 17.05.2025)
40. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. URL: https://doi.org/10.1007/978-0-387-45528-0 (дата звернення: 11.05.2025)
41. **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). Deep Learning. MIT Press.
42. Johannes Link, Peter Fröhlich (2003). Unit Testing in Java. How Tests Drive the Code
43. end-to-end testing. URL: <https://www.techtarget.com/searchsoftwarequality/definition/End-to-end-testing> (дата звернення: 11.05.2025)
44. CUDA. URL: <https://docs.nvidia.com/cuda/> (дата звернення 11.05.2025)
45. Géron, Aurélien. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow." O'Reilly Media, 2019.
46. JetBrains PyCharm. URL: [https://www.jetbrains.com/pycharm/#](https://www.jetbrains.com/pycharm/) (дата звернення 21.09.2024)
47. Python 3.11. URL: <https://www.python.org/> (дата звернення 10.04.2021)
48. NumPy. URL: <https://numpy.org/> (дата звернення 17.03.2025)
49. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 1998.
50. Krizhevsky, Alex, et al. "ImageNet classification with deep convolutional neural networks." Communications of the ACM, 2017.
51. Danil Klieshch, Volodymyr Fedorchenko. АНАЛІЗ ПІДХОДІВ ДО РОЗВ'ЯЗАННЯ ЗАДАЧ РОЗПIЗНАВАННЯ ОБРАЗІВ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ. (2023). 10.26906/SUNZ.2023.1.096
52. Goodfellow I., Bengio Y., Courville A. Deep Learning. – Cambridge, MA: MIT Press, 2016. – 775 p.
53. Згорткова нейронна мережа – просте пояснення CNN та її застосування. URL: <https://evergreens.com.ua/ua/articles/cnn.html> (дата звернення 13.04.2025)
54. Git. URL: <https://git-scm.com/> (дата звернення: 20.05.2023)
55. GitHub. URL: <https://github.com/> (дата звернення: 20.05.2023)
56. Open CV. URL: <https://opencv.org/university/free-opencv-course/?utm_source=google&utm_medium=cpc&utm_campaign=WW_opencv_OBC&utm_term=opencv&gad_source=1&gad_campaignid=20820136505&gbraid=0AAAAACbv-xg07Jpc-b2_F45sAmiuEr97I&gclid=Cj0KCQjw0erBBhDTARIsAKO8iqR0_-GMwawzsGSoxaHXGDMXMuq0Hc3sQfFscfKbIIyGag_vmn7ZrLwaAqmiEALw_wcB> (дата звернення: 16.03.2025)
57. Pillow. URL: <https://pillow.readthedocs.io/en/stable/> (дата звернення 13.03.2025)
58. Matplotlib 3.10.3 documentation. URL: <https://matplotlib.org/stable/> (дата звернення 13.03.2025)
59. tkinter — Python interface to Tcl/Tk. URL: <https://docs.python.org/uk/3.13/library/tkinter.html> (дата звернення 13.03.2025)
60. Botchkaryov A., Decentralized control of adaptive data collection processes based on equilibrium concept and reinforcement learning // Advances in Cyber-Physical Systems, Lviv, Volume 5, Number 2, 2020. - pp.50-55. <https://doi.org/10.23939/acps2020.02.050>
61. Кіберфізичні системи: технології збору даних / О.Ю. Бочкарьов, В.А. Голембо, Я.С. Парамуд, В.О. Яцук. За ред. А.О. Мельника, Львів: “Магнолія 2006”, 2019. - 176 с.
62. Maksymiv M., Rak T. Methods of video quality-improving // Штучний інтелект. – 2023. – № 3 (97). – С. 47–62.
63. Максимів М. Р., Рак Т. Є. Building and Optimizing Lightweight Generative Adversarial Neural Networks to Enhance Video Quality in the Client Devices Using Webgpu // Комп’ютерні системи та мережі. – 2025. – Вип. 7, № 1. – С. 186–194
64. Kniazhyk T., Muliarevych O., Cloud Computing With Resource Allocation Based on Ant Colony Optimization // Advances in Cyber-Physical Systems (ACPS), 8(2), Ukraine Lviv 2023, pp. 104–110. DOI: <https://doi.org/10.23939/acps2023.02.104>.
65. Пахомов С., Муляревич O., Боярінова Ю., Вплив системного аналізу та машинного навчання на ефективність розробки інформаційних систем // Наука і техніка сьогодні №8(36), Київ 2024, pp. 1115-1128. <https://doi.org/10.52058/2786-6025-2024-8(36)-1115-1128>
66. O. Muliarevych Enhancing system security: LLM-driven defense against prompt injection vulnerabilities // Advanced trends in radioelectronics, telecommunications and computer engineering, TCSET : proceedings 17th International conference (Lviv-Slavske, Ukraine, October 9–12), 2024. pp. 420–423.
67. Ірина Юрчак, Ольга Кичук, Віра Оксенюк. Техніки промптингу для покращення використання великих мовних моделей. &quot;Комп’ютерні системи та мережі&quot; Випуск 6, №2, 2024, c. 286-300, ISSN: 2707-2371 (Print), DOI: https://doi.org/10.23939/csn2024.02.286В.Оксенюк, І.Юрчак, О.Маркелов, М.Шурко, Б.Рубаха, А.Черниш. Архітектура та реалізація мобільного застосунку CALORIFY на основі рушія UNITY. Теорія і практика, Випуск 6, №2, 2024, c. 118-129, ISSN: 2707-6784 (Print), DOI:https://doi.org/10.23939/cds2024.02.118
68. І.Юрчак, А.Хіч, В.Оксенюк. Розуміння великих мовних моделей: майбутнє штучного інтелекту. Теорія і практика, Випуск 6, №2, 2024, c. 51-60, ISSN: 2707-6784 (Print), DOI:https://doi.org/10.23939/cds2024.02.051

# Додатки

## Додаток А. Лістинг файлів розробленої системи

dataset\_generator.py

|  |
| --- |
| import threading import uuid import time import cv2 import os import random import numpy as np import json from tkinter import Tk, filedialog, simpledialog import tkinter as tk import logging  extra\_stop = 0 # --- ФУНКЦІЯ ДЛЯ ВИБОРУ ВІДЕО --- def select\_video():  video\_path = filedialog.askopenfilename(title="Виберіть відеофайл",  filetypes=[("Video files", "\*.mp4;\*.avi;\*.mov")])  if not video\_path:  print("Відео не вибрано.")  quit(-1)  return video\_path   # --- ФУНКЦІЯ ДЛЯ ВИБОРУ ФІЛЬТРІВ --- def select\_settings():  root = tk.Tk()  root.withdraw() # Приховуємо головне вікно   settings = {}   def on\_submit():  # Зберігаємо значення у словник  settings["rotate"] = bool(rotate\_var.get())  settings["flip"] = bool(flip\_var.get())  settings["blur"] = bool(blur\_var.get())  settings["brightness"] = bool(brightness\_var.get())  settings["contrast"] = bool(contrast\_var.get())  settings["noise"] = bool(noise\_var.get())  settings["noise\_level"] = float(noise\_level\_var.get())  settings["brightness\_level"] = float(brightness\_level\_var.get())  settings['width'] = int(width\_var.get())  settings['height'] = int(height\_var.get())  settings['stop\_at'] = int(stop\_at.get())  settings['rewrite'] = bool(rewrite.get())  filter\_window.destroy() # Закриваємо вікно   # Створюємо нове вікно для вибору фільтрів  filter\_window = tk.Toplevel(root)  filter\_window.title("Вибір фільтрів")  filter\_window.grab\_set()   # Ініціалізуємо змінні (замість BooleanVar використовуємо IntVar)  rotate\_var = tk.IntVar(value=0)   flip\_var = tk.IntVar(value=0)  blur\_var = tk.IntVar(value=0)  brightness\_var = tk.IntVar(value=0)  contrast\_var = tk.IntVar(value=0)  noise\_var = tk.IntVar(value=0)  noise\_level\_var = tk.DoubleVar(value=0.1)  brightness\_level\_var = tk.DoubleVar(value=1.2)  width\_var = tk.IntVar(value=128)  height\_var = tk.IntVar(value=128)  stop\_at = tk.IntVar(value=0)  rewrite = tk.IntVar(value=1)   # Додаємо чекбокси  tk.Checkbutton(filter\_window, text="Застосовувати повороти?", variable=rotate\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Checkbutton(filter\_window, text="Застосовувати дзеркальне відображення?", variable=flip\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Checkbutton(filter\_window, text="Застосовувати розмиття?", variable=blur\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Checkbutton(filter\_window, text="Змінювати яскравість?", variable=brightness\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Scale(filter\_window, label="Рівень яскравості", variable=brightness\_level\_var, from\_=0.5, to=2.0, resolution=0.1,  orient='horizontal').pack(anchor='w')  tk.Checkbutton(filter\_window, text="Змінювати контраст?", variable=contrast\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Checkbutton(filter\_window, text="Додавати шум?", variable=noise\_var,  onvalue=1, offvalue=0).pack(anchor='w')  tk.Scale(filter\_window, label="Рівень шуму", variable=noise\_level\_var, from\_=0, to=50, resolution=1,  orient='horizontal').pack(anchor='w')  tk.Label(filter\_window, text="Розмір вихідного зображення:").pack(anchor='w')  tk.Entry(filter\_window, textvariable=width\_var).pack(anchor='w')  tk.Entry(filter\_window, textvariable=height\_var).pack(anchor='w')   tk.Label(filter\_window, text="Зупинити на кількості (0 - не зупиняти)").pack(anchor='w')  tk.Entry(filter\_window, textvariable=stop\_at).pack(anchor='w')  tk.Checkbutton(filter\_window, text="Перезаписувати?", variable=rewrite,  onvalue=1, offvalue=0).pack(anchor='w')  # Кнопка підтвердження  tk.Button(filter\_window, text="Підтвердити", command=on\_submit).pack()   root.wait\_window(filter\_window) # Очікуємо закриття вікна  root.destroy()   return settings # Повертаємо значення   frame = None temp\_frame = None roi\_selected = False roi = (0, 0, 0, 0) start\_point = None end\_point = None roi\_choose = False images\_processed = 0  def draw\_roi(event, x, y, flags, param):  global roi\_selected, roi, start\_point, end\_point, temp\_frame, frame, roi\_choose   if event == cv2.EVENT\_LBUTTONDOWN:  roi\_choose = True  start\_point = (x, y)  end\_point = (x, y)   elif event == cv2.EVENT\_MOUSEMOVE and start\_point is not None:  end\_point = (x, y)  if roi\_choose:  temp\_frame = frame.copy() # Оновлюємо тимчасове зображення  cv2.rectangle(temp\_frame, start\_point, end\_point, (0, 255, 0), 2) # Малюємо прямокутник   elif event == cv2.EVENT\_LBUTTONUP:  end\_point = (x, y)  x\_min, y\_min = min(start\_point[0], end\_point[0]), min(start\_point[1], end\_point[1])  x\_max, y\_max = max(start\_point[0], end\_point[0]), max(start\_point[1], end\_point[1])   # Обмежуємо координати в межах кадру  x\_min = max(0, x\_min)  y\_min = max(0, y\_min)  x\_max = min(frame.shape[1], x\_max)  y\_max = min(frame.shape[0], y\_max)   # Перевіряємо, чи ROI має допустимий розмір  if x\_max > x\_min and y\_max > y\_min:  roi\_selected = True  roi = (x\_min, y\_min, x\_max - x\_min, y\_max - y\_min)  else:  roi\_selected = False  roi\_choose = False  temp\_frame = frame.copy() # Оновлюємо тимчасове зображення  cv2.rectangle(temp\_frame, start\_point, end\_point, (0, 255, 0), 2)   def process\_video(video\_path, settings):  pause = True  if not video\_path:  return   start\_time = time.time()   logging.info("Початок обробки відео...")  cap = cv2.VideoCapture(video\_path, cv2.CAP\_FFMPEG)  cap.set(cv2.CAP\_PROP\_FPS, 1000)  class\_name = simpledialog.askstring("Клас об'єкта", "Введіть назву класу об'єкта (напр., car, person)")  save\_path = f"dataset128/{class\_name}"  os.makedirs(save\_path, exist\_ok=True)  annotations = []   global roi\_selected, roi, frame, temp\_frame, extra\_stop  cv2.namedWindow("Трекінг об'єкта")  cv2.setMouseCallback("Трекінг об'єкта", draw\_roi)   ret, frame = cap.read()  if not ret:  print("Не вдалося завантажити відео.")  return   tracker = None  frame\_count = 0   while cap.isOpened() and not extra\_stop:  # Якщо ROI вибрано, ініціалізуємо трекер  if roi\_selected and tracker is None:  tracker = cv2.legacy.TrackerCSRT\_create()  tracker.init(frame, roi)  roi\_selected = True   # Оновлення трекера  if tracker and not pause:  success, bbox = tracker.update(frame)  if success:  x, y, w, h = [max(0, min(int(v), frame.shape[i % 2])) for i, v in enumerate(bbox)]  w, h = (w if x+w <= frame.shape[1] else frame.shape[1] - x, h if y+h <= frame.shape[0] else frame.shape[0] - y)  # Перевірка коректності координат  if w > 0 and h > 0 and x >= 0 and y >= 0:  # Нормалізація координат та збереження  obj\_img = frame[y:y + h, x:x + w]  obj\_img = cv2.resize(obj\_img, (settings['width'], settings['height']))   # Збереження кадрів і анотацій у окремому потоці  thread = threading.Thread(target=save\_augmented\_images, args=(  obj\_img, settings, save\_path, frame\_count, x, w, y, h, frame, class\_name, annotations), daemon=True)  thread.start()   cv2.rectangle(temp\_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)   if temp\_frame is not None:  display\_frame = temp\_frame.copy()  else:  display\_frame = frame.copy()  # Відображення кадру  cv2.imshow("Трекінг об'єкта", display\_frame)   key = cv2.waitKey(30) & 0xFF  if key == 27 or key == ord('q'): # Натискання Esc виходить із циклу  break   if key == ord(" "):  if not roi\_selected and pause:  print(f"ROI {roi\_selected} PAUSE {pause}")  print("ROI не вибрано.")  continue  roi\_selected = True  tracker = None  pause = not pause   if not pause:  ret, frame = cap.read()  if not ret:  break  temp\_frame = frame.copy()  frame\_count += 1   cap.release()  cv2.destroyAllWindows()   end\_time = time.time()   print("\n=== ПІДСУМКИ ===")  print(f"[INFO] Загальний час обробки: {end\_time - start\_time:.2f} сек")  print(f"[INFO] Кількість зображень: {images\_processed}")  if images\_processed > 0:  print(f"[INFO] Середній час на 1 зображення: {(end\_time - start\_time) / images\_processed:.4f} сек")  print("=================")   if settings["rewrite"]:  write\_data = annotations  else:  with open(f"{save\_path}/annotations.json", "r") as f:  write\_data = json.load(f)  write\_data.extend(annotations)  with open(f"{save\_path}/annotations.json", "w") as f:  print("Збереження анотацій у файл")  print(f"Збережено {len(write\_data)} анотацій у {save\_path}/annotations.json")  json.dump(write\_data, f, indent=4)   def save\_augmented\_images(obj\_img, settings, save\_path, frame\_count, x, w, y, h, frame, class\_name, annotations):  global images\_processed, extra\_stop  augmented\_images = apply\_augmentations(obj\_img, settings)   for i, img in enumerate(augmented\_images):  if settings['stop\_at'] != 0 and images\_processed >= settings['stop\_at']:  extra\_stop = 1  print(f"Зупинка обробки зображень. Збережено {images\_processed} зображень.")  return   images\_processed += 1  img\_name = (f"{save\_path}/img\_{frame\_count:04d}\_{i}"  f"{'\_' + str(uuid.uuid4().hex) if not settings['rewrite'] else ''}.jpg")  cv2.imwrite(img\_name, img)  img\_height, img\_width, \_ = frame.shape  x\_center = (x + w / 2) / img\_width  y\_center = (y + h / 2) / img\_height  w\_norm = w / img\_width  h\_norm = h / img\_height  annotations.append(  {"image": img\_name, "bbox": [x\_center, y\_center, w\_norm, h\_norm], "class": class\_name})   def apply\_augmentations(img, settings):  augmented\_images = [img]  if settings.get("rotate", False):  angle = random.randint(-15, 15)  (h, w) = img.shape[:2]  M = cv2.getRotationMatrix2D((w // 2, h // 2), angle, 1.0)  rotated = cv2.warpAffine(img, M, (w, h))  augmented\_images.append(rotated)  if settings.get("flip", False):  flipped = cv2.flip(img, 1)  augmented\_images.append(flipped)  if settings.get("blur", False):  blurred = cv2.GaussianBlur(img, (5, 5), 0)  augmented\_images.append(blurred)  if settings.get("brightness", False):  bright = cv2.convertScaleAbs(img, alpha=settings["brightness\_level"], beta=30)  augmented\_images.append(bright)  if settings.get("contrast", False):  contrast = cv2.convertScaleAbs(img, alpha=1.5, beta=0)  augmented\_images.append(contrast)  if settings.get("noise", False):  noise = np.random.randint(0, settings["noise\_level"], img.shape, dtype='uint8')  noisy = cv2.add(img, noise)  augmented\_images.append(noisy)  return augmented\_images   def main():  # cv2.cuda.setDevice(0)  cv2.setUseOptimized(True)  cv2.CAP\_PROP\_BUFFERSIZE = 1  video\_file = select\_video()  selected\_settings = select\_settings()  process\_video(video\_file, selected\_settings)   if \_\_name\_\_ == "\_\_main\_\_":  main() |

train\_own\_dataset.py

|  |
| --- |
| import json import torch import torch.nn as nn import torch.optim as optim import torchvision.transforms as transforms from sklearn.model\_selection import train\_test\_split from torch.utils.data import DataLoader, SubsetRandomSampler from sklearn.metrics import confusion\_matrix, classification\_report, ConfusionMatrixDisplay import matplotlib.pyplot as plt from datetime import datetime from custom\_dataset\_loader import CustomDataset from model\_get import get\_model from normalizer import compute\_mean\_std import numpy as np import time  # --- Гіперпараметри --- BATCH\_SIZE = 32 EPOCHS = 20 LEARNING\_RATE = 0.001 IMG\_SIZE = 128 STOP\_CRITERION = 99 # Валідаційна точність для зупинки навчання. -1 - не враховувати MODEL\_TYPE = "resnet50" MODEL\_PATH = f"{MODEL\_TYPE}\_custom\_with\_metadata.pth" DATASET\_PATH = r"D:\LPNU\DIPLOMA\DIPLOMA\dataset" DEVICE = torch.device("cuda" if torch.cuda.is\_available() else "cpu") date = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")  # --- Початкові трансформації --- transform = transforms.Compose([  transforms.ToTensor() ])  def make\_normalize\_transform(dataset):  dataloader = DataLoader(dataset, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=4)  if len(dataset) == 0:  raise ValueError("❌ Помилка: датасет порожній. Переконайтеся, що дані коректні.")  mean, std = compute\_mean\_std(dataloader)  transform = transforms.Compose([  transforms.Resize((IMG\_SIZE, IMG\_SIZE)),  transforms.ToTensor(),  transforms.Normalize(mean=mean, std=std)  ])  return transform, mean, std  def train(dataset, mean, std, show\_plot=False):  indices = list(range(len(dataset)))  train\_indices, val\_indices = train\_test\_split(indices, test\_size=0.2,  stratify=[dataset[idx][1].item() for idx in indices])  train\_sampler = SubsetRandomSampler(train\_indices)  val\_sampler = SubsetRandomSampler(val\_indices)  train\_loader = DataLoader(dataset, batch\_size=BATCH\_SIZE, sampler=train\_sampler, num\_workers=2)  val\_loader = DataLoader(dataset, batch\_size=BATCH\_SIZE, sampler=val\_sampler, num\_workers=2)  num\_classes = len(dataset.class\_to\_idx)  model = get\_model(MODEL\_TYPE, num\_classes).to(DEVICE)  criterion = nn.CrossEntropyLoss()  optimizer = optim.Adam(model.parameters(), lr=LEARNING\_RATE)   def evaluate\_model(model, dataloader):  model.eval()  correct, total = 0, 0  with torch.no\_grad():  for inputs, labels in dataloader:  inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)  outputs = model(inputs)  \_, predicted = torch.max(outputs, 1)  correct += (predicted == labels).sum().item()  total += labels.size(0)  return correct / total \* 100   def plot\_training\_results(train\_losses, train\_accuracies, val\_accuracies, epochs, save\_path=f"training\_plot {date}.png"):  plt.figure(figsize=(10, 5))  plt.plot(range(1, epochs + 1), train\_losses, label="Втрати (Train)", marker="o")  plt.plot(range(1, epochs + 1), train\_accuracies, label="Точність (Train)", marker="s")  plt.plot(range(1, epochs + 1), val\_accuracies, label="Точність (Validation)", marker="^")  plt.xlabel("Епохи")  plt.ylabel("Значення")  plt.title("Динаміка навчання моделі")  plt.legend()  plt.grid(True)  plt.savefig(save\_path)  plt.show()  print(f"📊 Графік навчання збережено у {save\_path}")   def train\_model(model, train\_loader, val\_loader, criterion, optimizer, epochs):  train\_losses, train\_accuracies, val\_accuracies = [], [], []  start\_time = time.time()  for epoch in range(epochs):  model.train()  running\_loss, correct, total = 0.0, 0, 0  for inputs, labels in train\_loader:  inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)  optimizer.zero\_grad()  outputs = model(inputs)  loss = criterion(outputs, labels)  loss.backward()  optimizer.step()  running\_loss += loss.item()  \_, predicted = torch.max(outputs, 1)  correct += (predicted == labels).sum().item()  total += labels.size(0)  train\_losses.append(running\_loss / len(train\_loader))  train\_accuracies.append(correct / total \* 100)  val\_acc = evaluate\_model(model, val\_loader)  val\_accuracies.append(val\_acc)  print(f"🌀 Епоха {epoch + 1}/{epochs} | Втрата: {train\_losses[-1]:.4f} | "  f"Тренувальна точність: {train\_accuracies[-1]:.2f}% | Валідаційна точність: {val\_acc:.2f}%")  if val\_acc >= STOP\_CRITERION and STOP\_CRITERION != -1:  print(f"🔴 Зупинка навчання: досягнуто критерію зупинки. Валідаційна точність {val\_acc}")  break  if show\_plot:  plot\_training\_results(train\_losses, train\_accuracies, val\_accuracies, len(train\_losses))  torch.save({  "model\_state\_dict": model.state\_dict(),  "metadata": {  "normalization\_mean": list(mean),  "normalization\_std": list(std),  "img\_size": IMG\_SIZE,  "model\_type": MODEL\_TYPE,  "class\_names": list(dataset.class\_to\_idx.keys())  }  }, MODEL\_PATH)  print(f"✅ Модель збережена в {MODEL\_PATH}")   train\_model(model, train\_loader, val\_loader, criterion, optimizer, EPOCHS)  def validate\_model(model, dataloader, class\_names):  model.eval()  all\_preds, all\_labels = [], []  with torch.no\_grad():  for inputs, labels in dataloader:  inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)  outputs = model(inputs)  \_, predicted = torch.max(outputs, 1)  all\_preds.extend(predicted.cpu().numpy())  all\_labels.extend(labels.cpu().numpy())  accuracy = np.mean(np.array(all\_preds) == np.array(all\_labels)) \* 100  print(f"📊 Загальна точність: {accuracy:.2f}%")  cm = confusion\_matrix(all\_labels, all\_preds)  disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=class\_names)  disp.plot(cmap="viridis", xticks\_rotation="vertical")  plt.title("Матриця неточностей")  plt.show()  print(f"🔹 Звіт про класифікацію:\n{classification\_report(all\_labels, all\_preds, target\_names=class\_names)}")  if \_\_name\_\_ == "\_\_main\_\_":  dataset = CustomDataset(root\_dir=DATASET\_PATH, transform=transform)  transform, mean, std = make\_normalize\_transform(dataset)  dataset.transform = transform  train(dataset, mean, std)  checkpoint = torch.load(MODEL\_PATH, map\_location=DEVICE)  model = get\_model(MODEL\_TYPE, len(checkpoint["metadata"]["class\_names"])).to(DEVICE)  model.load\_state\_dict(checkpoint["model\_state\_dict"])  validate\_model(model, DataLoader(dataset, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=4),  checkpoint["metadata"]["class\_names"]) |

dir\_class\_predict.py

|  |
| --- |
| import time  import torch import torchvision.transforms as transforms from PIL import Image import os from torchvision import models import torch.nn as nn from sklearn.metrics import classification\_report, confusion\_matrix import numpy as np from functools import wraps   def evaluate\_time\_stats\_per\_func():  stats\_dict = {}   def decorator(func):  stats = stats\_dict.setdefault(func.\_\_name\_\_, {'calls': 0, 'total\_time': 0.0, 'max\_time': 0.0, 'min\_time': 0.0})   @wraps(func)  def wrapper(\*args, \*\*kwargs):  stats['calls'] += 1  start\_time = time.time()  result = func(\*args, \*\*kwargs)  elapsed = time.time() - start\_time  stats['max\_time'] = max(stats['max\_time'], elapsed)  stats['min\_time'] = min(stats['min\_time'], elapsed) if stats['calls'] > 1 else elapsed  stats['total\_time'] += elapsed  avg\_time = stats['total\_time'] / stats['calls']  print(f"[{func.\_\_name\_\_}] Call #{stats['calls']}: Time = {elapsed:.4f}s | Avg = {avg\_time:.4f}s | Total = {stats['total\_time']:.4f}s")  return result  return wrapper   decorator.stats\_dict = stats\_dict  return decorator  time\_decorator = evaluate\_time\_stats\_per\_func()  @time\_decorator def load\_checkpoint(model\_path, device):  checkpoint = torch.load(model\_path, map\_location=device)  return checkpoint, checkpoint['metadata']  @time\_decorator def build\_model(metadata, num\_classes):  model = getattr(models, metadata.get('model\_type', 'resnet50'))(pretrained=False)  if hasattr(model, "classifier"):  in\_features = model.classifier[-1].in\_features  model.classifier[-1] = nn.Linear(in\_features, num\_classes)  elif hasattr(model, "fc"):  in\_features = model.fc.in\_features  model.fc = nn.Linear(in\_features, num\_classes)  return model  @time\_decorator def get\_transform(metadata):  return transforms.Compose([  transforms.Resize((metadata.get("img\_size", 32), metadata.get("img\_size", 32))),  transforms.ToTensor(),  transforms.Normalize(mean=metadata.get("normalization\_mean"),  std=metadata.get("normalization\_std"))  ])  @time\_decorator def predict\_image(model, image\_tensor, device):  model.eval()  with torch.no\_grad():  output = model(image\_tensor.to(device))  predicted\_class = torch.argmax(output, dim=1).item()  return predicted\_class  @time\_decorator def evaluate\_folder(model, image\_folder, class\_names, transform, device):  y\_true = []  y\_pred = []  actual\_class = class\_names.index(os.path.basename(image\_folder))  for filename in os.listdir(image\_folder):  if filename.endswith((".jpg", ".png", ".jpeg")):  image\_path = os.path.join(image\_folder, filename)  image = Image.open(image\_path).convert("RGB")  image\_tensor = transform(image).unsqueeze(0)  predicted\_class = predict\_image(model, image\_tensor, device)  y\_true.append(actual\_class)  y\_pred.append(predicted\_class)  print(  f"{filename}: Передбачено - {class\_names[predicted\_class]}, Реальність - {class\_names[actual\_class]}, {'✅' if predicted\_class == actual\_class else '❌'}"  )  accuracy = (np.array(y\_true) == np.array(y\_pred)).mean() \* 100 if len(y\_true) > 0 else 0  print(f"\nЗагальна точність: {accuracy:.2f}%")   # Друк матриці неточностей та метрик  if y\_true:  print("\nМатриця неточностей:")  cm = confusion\_matrix(y\_true, y\_pred, labels=range(len(class\_names)))  print(cm)  print("\nЗвіт про класифікацію:")  print(classification\_report(  y\_true, y\_pred, target\_names=class\_names, labels=range(len(class\_names)), zero\_division=0  ))   return accuracy  @time\_decorator def main(model\_path, image\_folder):  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  checkpoint, metadata = load\_checkpoint(model\_path, device)  class\_names = metadata['class\_names']  num\_classes = len(class\_names)  model = build\_model(metadata, num\_classes)  model.load\_state\_dict(checkpoint['model\_state\_dict'])  model = model.to(device)  transform = get\_transform(metadata)  return evaluate\_folder(model, image\_folder, class\_names, transform, device)   if \_\_name\_\_ == "\_\_main\_\_":  MODEL\_PATH = "mobilenet\_v3\_small\_custom\_with\_metadata.pth"  IMAGE\_FOLDER = "dataset128/dog"  main(MODEL\_PATH, IMAGE\_FOLDER)   print("\n--- Summary ---")  for func, stats in time\_decorator.stats\_dict.items():  avg = stats['total\_time'] / stats['calls'] if stats['calls'] else 0  print(f"{func}: calls={stats['calls']}, total={stats['total\_time']:.4f}s, avg={avg:.4f}s, max={stats['max\_time']:.4f}s, min={stats['min\_time']:.4f}s") |