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

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 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.

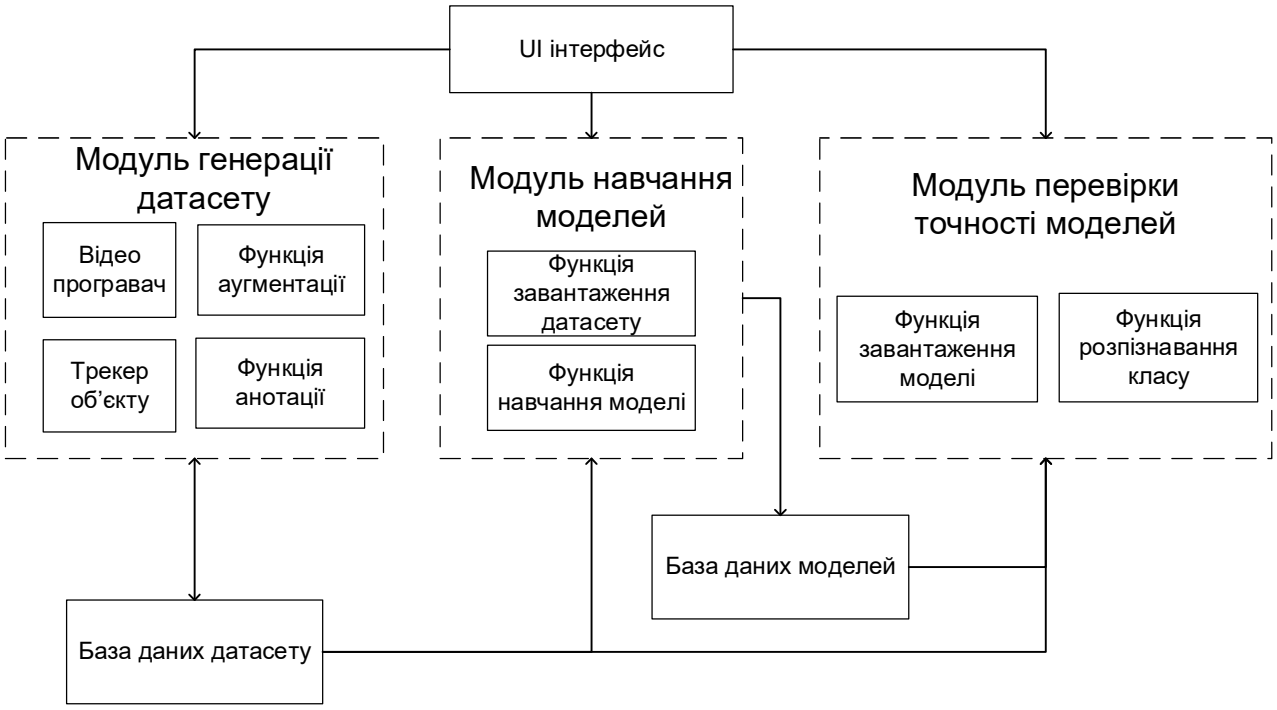

Рис. 2.1 – Структурна схема архітектури програмного рішення

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.

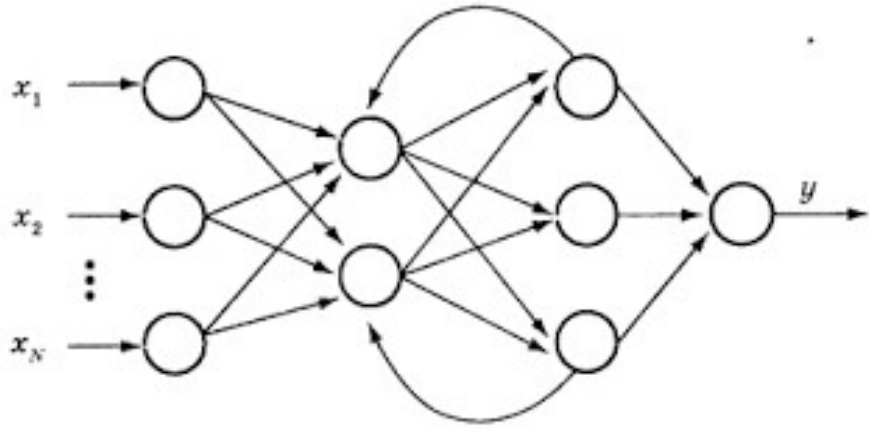

Рис.2.1 – Схема повнозв’язної ШНМ

2. **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.

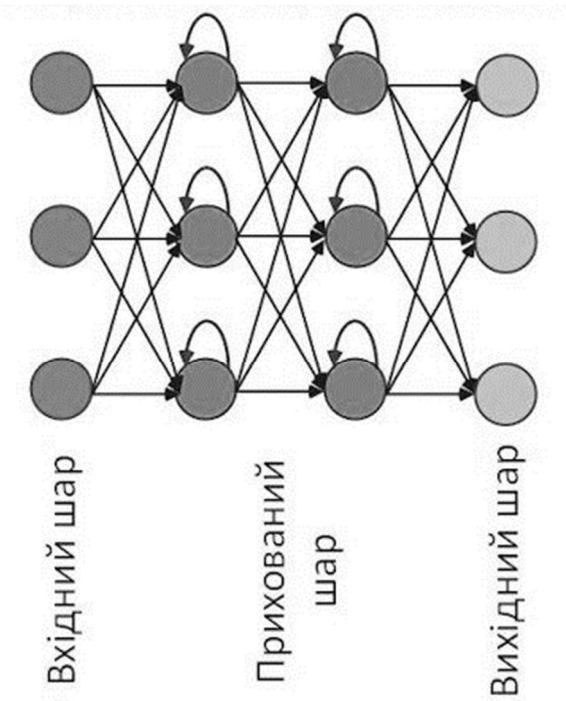

Рис.2.2 – Схема рекурентної ШНМ

3. **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.

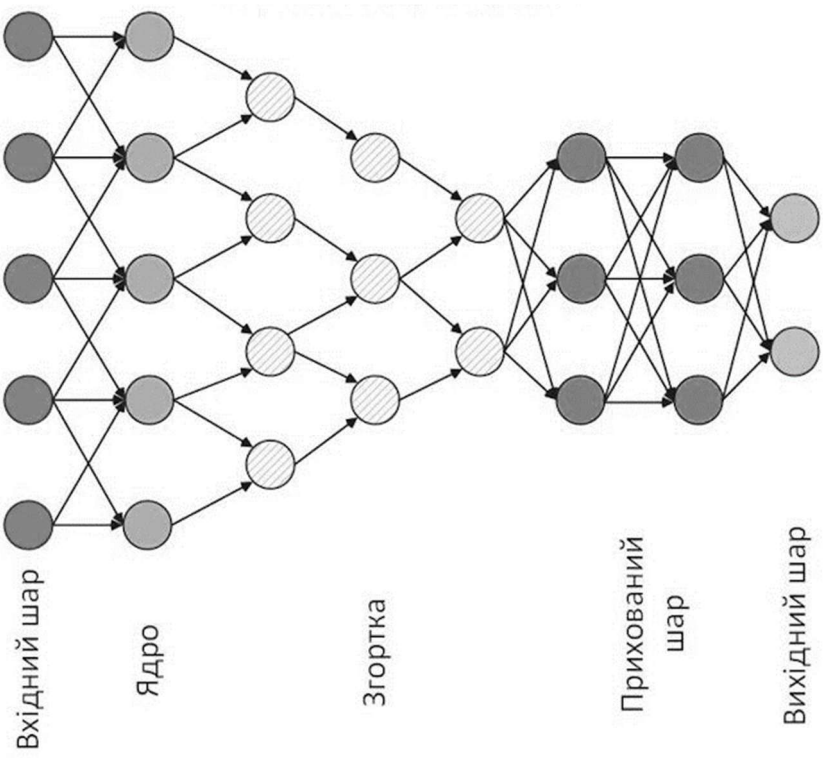

Рис. 2.3 – Схема згорткової ШНМ

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-міра 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.

2. **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.

3. **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;

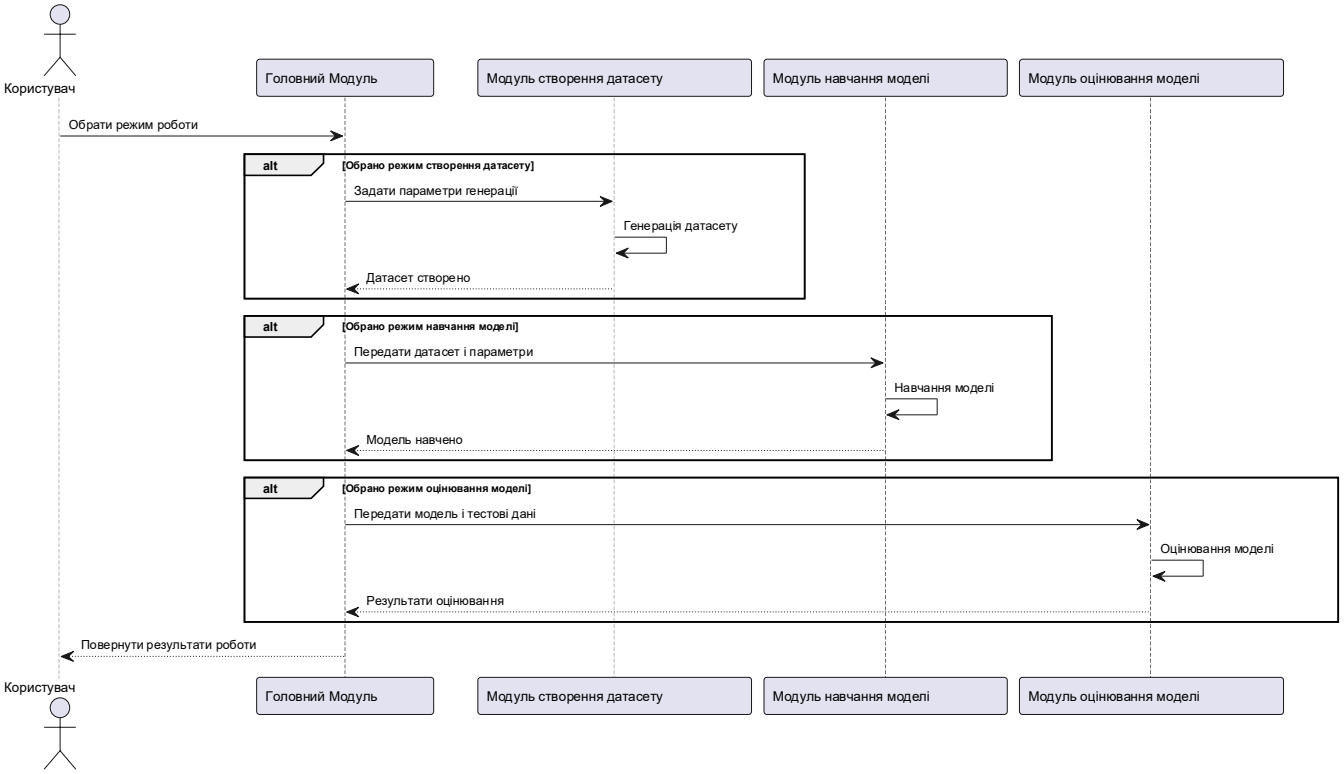

Рис. 3.1 – Взаємодія користувача з модулями системи. Діаграма послідовностей

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.

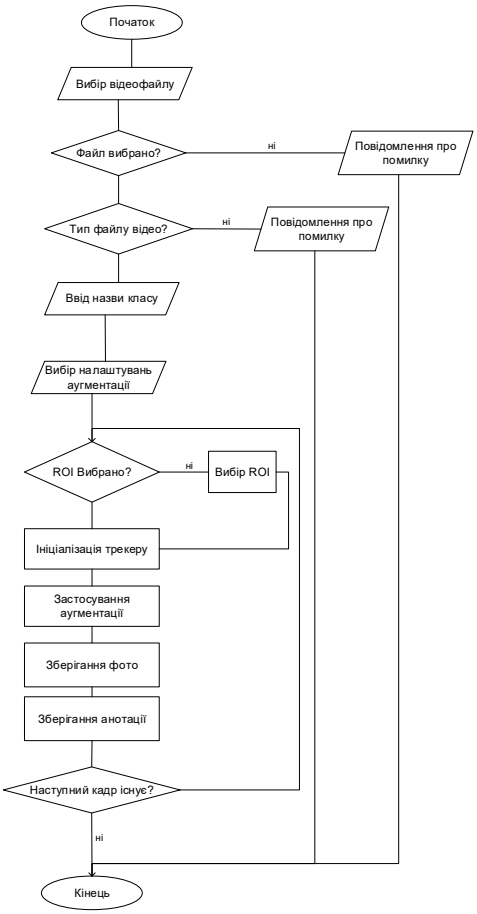

Рис. 3.2 – Генерація даних. Блок-схема алгоритму

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.

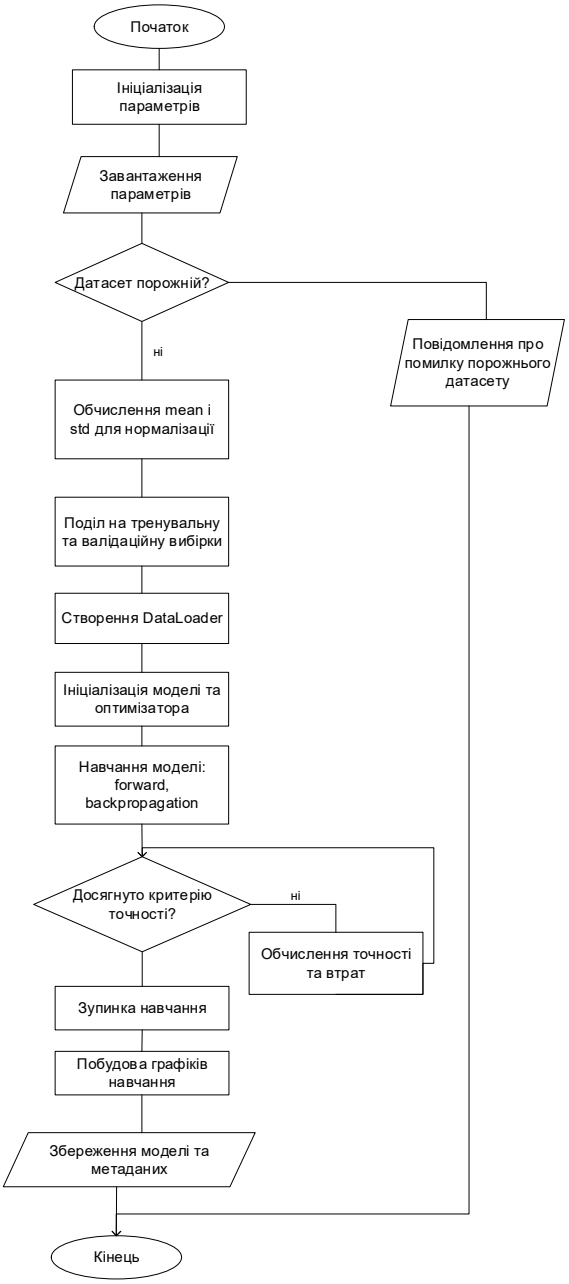

Рис.3.3 – Навчання моделей. Блок-схема алгоритму

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.

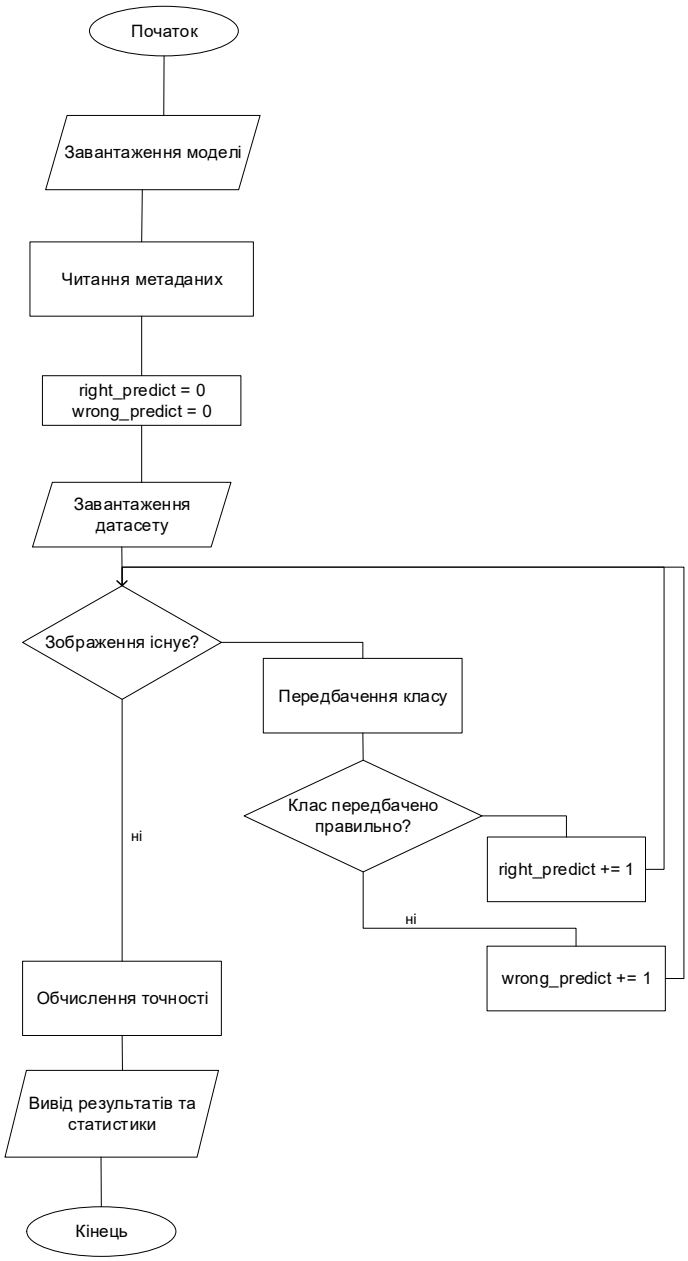

Рис. 3.4 – Перевірка роботи моделей. Блок-схема алгоритму

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.

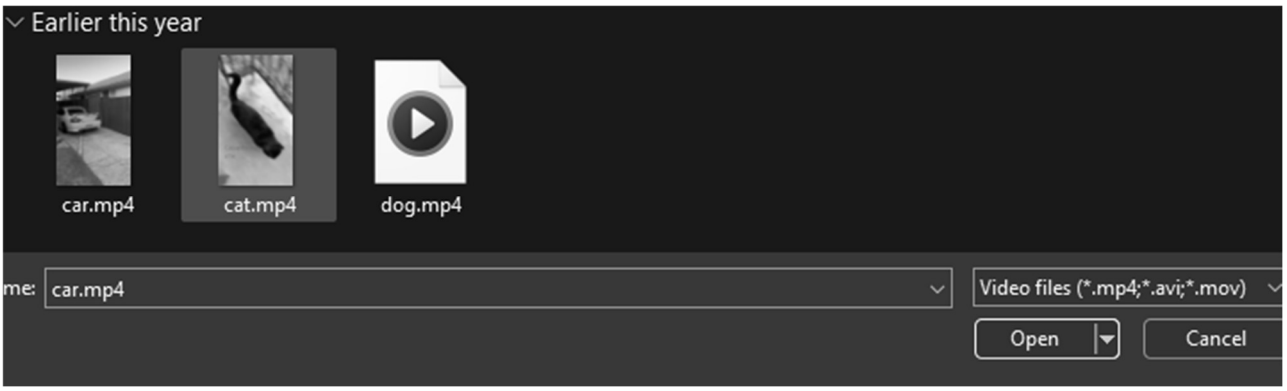

Рис. 3.5 – Вигляд інтерфейсу при виборі відеофайлу

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.

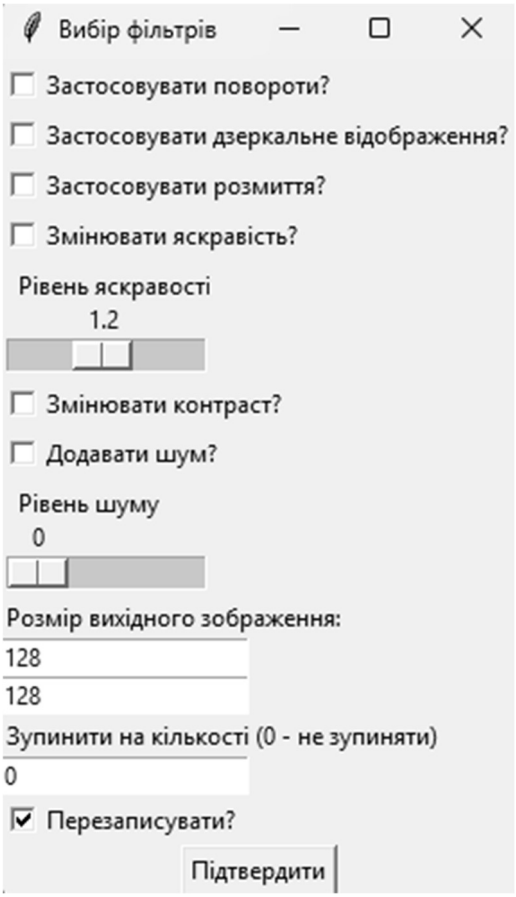

Рис. 3.6 – Інтерфейс вибору налаштувань аугментації

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.

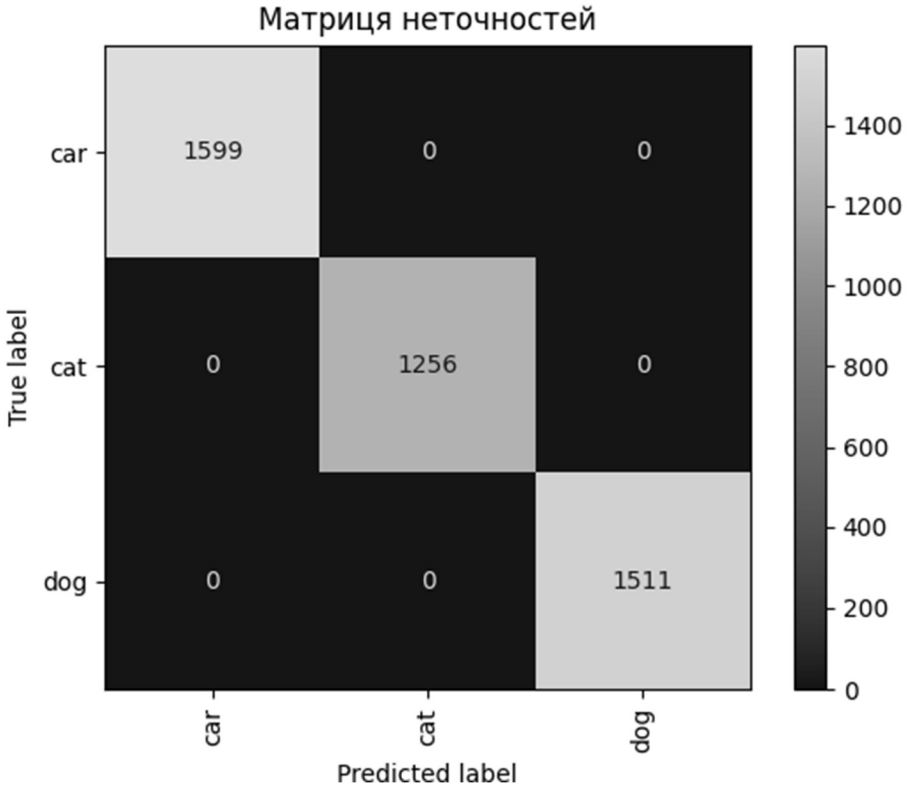

Рис 3.8 – Матриця неточностей

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.

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