Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра електронних обчислювальних машин



**ЗВІТ**

З перед дипломної практики

Студента IV курсу групи КІ-406

Ярмола Юрія Юрійовича

**Тема:** “ Програмна платформа створення штучних нейронних мереж ”

**Термін практики** з «28» квітня 2025р. по «10» травня 2025р.

**Керівники практики:**

від кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. викладач каф. ЕОМ Гузинець Н. В.

(підпис) (наук.ст, вч.звання, прізвище, ім’я, по-батькові)

**ОЦІНКА** \_\_\_\_\_\_\_\_\_

**Дата** \_\_\_\_\_\_\_\_\_\_\_\_\_

**ЗМІСТ**

[Розділ 1. АНАЛІТИЧНИЙ ОГЛЯД ТА ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ 3](#_Toc197794488)

[1.1 Аналіз сучасних підходів до створення штучних нейронних мереж 3](#_Toc197794489)

[1.2 Аналіз відомих інструментів та платформ для побудови, навчання та оцінки моделей 4](#_Toc197794490)

[1.3 Аналіз алгоритмів розв’язання задачі 7](#_Toc197794491)

[1.3.1 Алгоритми формування даних для тренування та валідації 8](#_Toc197794492)

[1.3.2 Алгоритми навчання штучних нейронних мереж 8](#_Toc197794493)

[1.3.3 Алгоритми перевірки моделі 9](#_Toc197794494)

[1.4 Основні режими функціонування платформи 9](#_Toc197794495)

[Висновки до розділу 1 10](#_Toc197794496)

[Розділ 2. ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ 12](#_Toc197794497)

[2.1. Аналіз задачі та вимог до програмної реалізації 12](#_Toc197794498)

[2.1.1 Функціональні вимоги 12](#_Toc197794499)

[2.1.2 Нефункціональні вимоги 13](#_Toc197794500)

[2.2. Розробка структурної схеми програмного рішення 14](#_Toc197794501)

[2.3. Вибір мови програмування та технологій 15](#_Toc197794502)

[2.4. Засоби розробки програмного забезпечення 17](#_Toc197794503)

[2.5. Вибрані бібліотеки та фреймворки 18](#_Toc197794504)

[2.6. Вимоги до апаратного забезпечення 20](#_Toc197794505)

[Висновки до розділу 2 22](#_Toc197794506)

[ВИСНОВКИ 23](#_Toc197794507)

[СПИСОК ЛІТЕРАТУРИ 24](#_Toc197794508)

# 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). Це можна пояснити виходом зрозумілим 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. 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, 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 на біологічні нейрони мереж, що дають змогу обробляти великі обсяги даних та знаходити складні залежності у них. Штучні нейронні мережі мають різні архітектури для моделювання нейронів, ось найпопулярніші з них:

* Багатошарові перцептрони

Основний блок для багатьох інших нейронних мереж. Використовується для бінарної класифікації.

* Згорткові нейронні мережі

Ефективні для обробки зображень і використовують згорткові шари для виявлення патернів у зображеннях.

* Рекурентні нейронні мережі

Призначені для обробки послідовних даних, таких як текст або часові ряди.

* Мережі асоціативної пам’яті

Використовуються для розв'язання завдань асоціативної пам'яті і оптимізації.

* Трансформери

Використовуються для обробки послідовних даних та роботи з прикладами з різних контекстів.

Методи навчання включають підконтрольне навчання для прогнозів, безконтрольне для кластеризації та навчання з підкріпленням для прийняття рішень. Для оптимізації використовуються алгоритми, такі як градієнтний спуск та його модифікації (Adam, RMSprop), що дозволяють зменшити функцію втрат і покращити точність моделі.

Популярні платформи для реалізації ШНМ, як TensorFlow, PyTorch та Keras, забезпечують інструменти для швидкого створення, навчання і тестування моделей. Водночас вони мають обмеження, зокрема потребу у великих обчислювальних ресурсах і залежність від якості даних.

## 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

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

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

* Keras

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

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

* Roboflow

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 алгоритмів розв’язання задачі

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

### 1.3.1 Алгоритми формування даних для тренування та валідації

Якість навчання штучної нейронної мережі значною мірою залежить від якості вхідних даних. У науковій та практичній літературі сформовано усталений підхід до створення навчальних вибірок, який включає наступні ключові етапи:

1. Збір даних. Джерелами інформації можуть бути відкриті набори даних, користувацькі записи або власноруч зібрані дані. Типи даних варіюються залежно від задачі: зображення, текст, аудіо тощо.
2. Анотація. Передбачає маркування зібраної інформації. Для зображень — це визначення координат об’єктів або присвоєння класів.
3. Попередня обробка. Застосовується для нормалізації та очищення даних: зміна розмірів, форматів, фільтрація шумів, видалення зайвого.
4. Аугментація. Метод штучного збільшення кількості прикладів за допомогою перетворень: обертання, масштабування, дзеркалювання тощо.
5. Розподіл на вибірки. Дані поділяються на навчальну, валідаційну та тестову вибірки, що забезпечує можливість точного навчання та оцінки.

### 1.3.2 Алгоритми 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 алгоритму:

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 алгоритму 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 Алгоритми перевірки моделі

Оцінка ефективності моделі — завершальний етап у загальній схемі роботи з нейронною мережею. Основні етапи перевірки включають:

1. Формування тестової вибірки. Створюється до початку навчання для забезпечення об'єктивності.
2. Завантаження навченої моделі. Модель, збережена після навчання, використовується без повторної оптимізації.
3. Генерація передбачень. На основі вхідних даних модель генерує результати.
4. Порівняння з еталонними значеннями. Дає змогу оцінити точність.
5. Обчислення метрик. Наприклад, точність, повнота, F1-міра — залежно від типу задачі.
6. Аналіз результатів. У разі незадовільної точності можливий перегляд підходу до навчання або структури мережі.

## 1.4 Основні режими функціонування платформи

Розроблена платформа функціонує в кількох основних режимах, що забезпечують її повноцінну роботу та взаємодію з користувачем. Кожен режим виконує окрему функціональну задачу, спрямовану на ефективне використання моделі штучного інтелекту. Основні режими функціонування включають:

1. Створення та оновлення датасету - у цьому режимі здійснюється збір, аналіз та попередня обробка даних. Користувач може завантажити нові приклади, а система автоматично проводить очищення, анотацію та структурування даних для подальшого навчання моделі.
2. Навчання моделі - платформа дозволяє запускати процес навчання моделі на підготовленому датасеті. У цьому режимі виконується побудова нейронної мережі, оптимізація ваг, налаштування гіперпараметрів та збереження навченої моделі у відповідному форматі.
3. Перевірка точності - цей режим призначений для оцінювання якості побудованої моделі на тестовому наборі даних. За його допомогою користувач може переглянути метрики ефективності та зробити висновки щодо необхідності покращення моделі.
4. Прогнозування (інференсу) - після успішного навчання модель використовується для передбачення результатів на нових, раніше невідомих даних. Це є основна практична функція платформи, яка забезпечує користувача готовими результатами аналізу або класифікації.
5. Візуалізація - для зручності користувача платформа пропонує візуальні засоби відображення результатів роботи моделі: графіки, таблиці, приклади передбачень, а також звіти щодо точності та якості моделі.

Кожен з режимів є частиною загальної архітектури системи та може запускатися окремо або в складі автоматизованого робочого циклу.

## Висновки до розділу 1

У першому розділі було розглянуто основні теоретичні аспекти, пов’язані з побудовою інтелектуальної системи на основі штучного інтелекту. Визначено мету та завдання розробки, сформульовано вимоги до функціонування платформи, а також обґрунтовано вибір інструментів і технологій для реалізації.

Було детально описано алгоритм створення навчального датасету, навчання моделі та її перевірки на відповідних даних. Також наведено основні режими роботи платформи, які забезпечують повний цикл функціонування: від збору даних до отримання передбачень та аналізу результатів.

Проведений аналіз дає підґрунтя для переходу до практичної реалізації системи, що буде розглянуто у наступних розділах.

# 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 вимог до програмної реалізації

У межах програмної платформи основною метою є розробка навчальної системи, 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.
* **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.

### 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.
* Процес класифікації одного зображення повинен займати не більше 3 секунд.
* 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.

## 2.2. 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.

Конкретно в даному програмному рішенні було 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.

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 виявилася найбільш збалансованим 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 та супутніх технологій є критично важливим етапом проектування програмного забезпечення, оскільки він визначає не лише технічні можливості системи, але й гнучкість, масштабованість, продуктивність та зручність подальшої підтримки коду. У межах цього проекту основною мовою програмування була обрана Python, оскільки вона поєднує у собі високу читабельність, велику кількість бібліотек для роботи з даними та потужну екосистему для розробки і навчання моделей штучного інтелекту.

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

* NumPy — 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. 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 проєкті. 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. Усі зміни відслідковуються та документуються за допомогою репозиторію, розміщеного на платформі GitHub, 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 проєкті. 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.

## 2.5. 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 –** 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 -** 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) -** 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 -** 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 -** 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 -** 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.

## 2.6. Вимоги до апаратного забезпечення

Для забезпечення коректної, ефективної та стабільної роботи програмного забезпечення, яке реалізує автоматичне формування анотованих датасетів та класифікацію зображень за допомогою алгоритмів 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

**Специфічні вимоги до середовища розробки:**

* IDE: JetBrains PyCharm Professional або Community Edition — для зручної роботи з Python-кодом, налагодженням та інтеграцією з віртуальними середовищами
* Підтримка Python 3.9+, сумісність з бібліотеками OpenCV, PyTorch, Pillow тощо

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.

## 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 сформовано уявлення про необхідні ресурси, продуктивність та безпеку системи, що забезпечує її стабільну роботу в умовах реального навантаження.

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.

У якості основної мови програмування обрано Python, що 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 здійснено 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

# ВИСНОВКИ

Проведено аналіз сучасних підходів до створення штучних нейронних мереж та порівняння популярних платформ для їх розробки: TensorFlow, PyTorch, Keras, Transformers і Roboflow. Визначено основні обмеження існуючих рішень та обґрунтовано необхідність розробки спеціалізованої платформи для автоматизованого формування датасетів і навчання моделей.

Детально розглянуто алгоритми формування навчальних даних, процеси тренування нейронних мереж та методи оцінки їх ефективності. Визначено п'ять ключових режимів функціонування розробленої платформи: створення датасету, навчання моделі, перевірка точності, прогнозування та візуалізація результатів.

На основі порівняння різних архітектурних підходів обрано модульну структуру як оптимальне рішення для забезпечення гнучкості розробки та масштабованості системи. Виділено основні функціональні модулі: генерація даних, формування датасету, навчання ШНМ, класифікація та допоміжні утиліти.

Для реалізації платформи обрано мову програмування Python та стек технологій, що включає PyTorch для побудови нейронних мереж, OpenCV і Pillow для обробки зображень, NumPy для обчислень, Matplotlib для візуалізації та Tkinter для створення інтерфейсу користувача.

Визначено мінімальні та рекомендовані вимоги до апаратного забезпечення з урахуванням необхідності обробки великих обсягів даних та використання GPU для прискорення навчання моделей. Для забезпечення якісного процесу розробки обрано середовище PyCharm та систему контролю версій Git.

Розроблена платформа забезпечує автоматизацію повного циклу роботи з нейронними мережами – від підготовки даних до практичного застосування навчених моделей для класифікації зображень. Архітектурні та технологічні рішення формують надійну основу для подальшого розвитку системи та її адаптації до більш складних задач машинного навчання.

# СПИСОК ЛІТЕРАТУРИ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016. – 800 с.
2. Chollet F. Deep Learning with Python. – Manning Publications, 2021. – 504 с.
3. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2020. – 480 с.
4. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521. – P. 436-444.
5. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. – 2016. – P. 770-778.
6. Abadi M., Barham P., Chen J. et al. TensorFlow: A System for Large-Scale Machine Learning // OSDI. – 2016. – Vol. 16. – P. 265-283.
7. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. – 2019. – P. 8026-8037.
8. Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. – 2000.
9. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. – O'Reilly Media, 2022. – 550 с.
10. Hunter J.D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. – 2007. – Vol. 9(3). – P. 90-95.
11. Li S., Zhao Z., Hu R. et al. Conceptual design and implementation of a modular architecture for intelligent robots // IEEE Transactions on Systems, Man, and Cybernetics: Systems. – 2019. – Vol. 51(5). – P. 3077-3090.
12. Ткаченко Р.О., Іваніюк В.М., Ткаченко П.Р., Кустра Н.О. Глибинні нейронні мережі у комп'ютерному зорі. – Львів: Видавництво Львівської політехніки, 2021. – 340 с.
13. Дорошенко А.Ю., Бекетов О.Г. Мови програмування та програмування систем штучного інтелекту. – К.: НАУ, 2019. – 192 с.
14. Бідюк П.І., Кориченко Л.О. Проектування комп'ютерних інформаційних систем підтримки прийняття рішень. – Київ: ННК "ІПСА" НТУУ "КПІ", 2020. – 340 с.
15. Субботін С.О. Нейронні мережі: теорія та практика. – Житомир: Вид-во ЖДУ ім. І. Франка, 2020. – 184 с.
16. Шаховська Н.Б., Литвин В.В., Камінський Р.М. Системи штучного інтелекту. – Львів: Видавництво Львівської політехніки, 2022. – 392 с.
17. Шинкаренко В.І., Ільман В.М. Структури даних та алгоритми обробки даних. – Дніпро: Видавництво ПФ "Стандарт-Сервіс", 2021. – 154 с.