МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Інтелектуальна система генерації зображень проектних рішень на основі рукотворних графічних даних

Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп'ютерні науки»**
Освітня програма **«Комп'ютерні науки»**
Освітній рівень: бакалавр

Виконала: студентка 4 курсу, групи КН– 42

Рєзнікова А.О.

Керівник Білан С.М.

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри інтелектуальних технологій
Протокол № від р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ – 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій
Кафедра інтелектуальних технологій
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Зав. кафедри інтелектуальних технологій
к.т.н., доц. Іларіонов О.Є.
(звання, прізвище та ініціали)

(підпис)

« ____ » _____ 2023 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ
Рєзніковій Анні Олексіївні
(прізвище, ім'я, по батькові)

1. Тема роботи: «Інтелектуальна система генерації зображень проектних рішень на основі рукотворних графічних даних» затверджена наказом ректора від «11» листопада 2022 р. №4
2. Термін здачі студентом закінченого проекту (роботи): 31 травня 2023 року.
3. Вихідні дані до проекту (роботи): Розробити систему генерації зображень планів на основі рукотворних графічних даних
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
   1) Аналіз процесу створення штучної нейронної мережі для розпізнавання рукотворних ескізів
   2) Розробка архітектури програмного забезпечення
   3) Реалізація програмного забезпечення додатку
5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій):
   1. Тема, об'єкт та предмет дослідження (1 слайд)
   2. Використання інтелектуальних технологій в архітектурі
   3. Аналіз основних процесів предметного середовища
   4. Проектні рішення з реалізації застосунку

5. Реалізація застосунку
6. Висновки

6. Консультанти з випускної кваліфікаційної роботи із зазначенням її розділів, що їх стосуються

Розділ	Консультант	Завдання видав	Завдання прийняв
1			
2			
3			

7. Дата видачі завдання 15 лютого 2023 року

Керівник випускної кваліфікаційної ро _____/ С.М.Білан/
(ініціали та прізвище)

Завдання прийняв до виконання _____ / А.О.Рєзнікова/
(підпис) (ініціали та прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів	Примітка
1	Аналіз предметної області	15.02 – 27.02	
2	Робота над першим розділом. Аналіз процесу створення штучної нейронної мережі для розпізнавання рукотворних ескізів	27.02 – 08.03	
3	Робота над другим розділом. Розробка архітектури застосунку	08.03 – 08.04	
4	Робота над третім розділом. Реалізація програмного забезпечення	08.04 – 13.05	
5	Оформлення пояснювальної записки	13.05 – 25.05	
6	Робота над презентаційним матеріалом	25.05 – 29.05	

Керівник випускної кваліфікаційної роботи М. Білан/
ніціали та прізвище)

Студент _____ / А.О. Рєзнікова/

(підпис) (ініціали та прізвище)

Анотація

Рєзнікова Анна Олексіївна виконала випускну кваліфікаційну роботу на тему «Інтелектуальна система генерації зображень проектних рішень на основі рукотворних графічних даних» за спеціальністю 122 – «Комп'ютерні науки».

У випускній кваліфікаційній роботі реалізована система класифікації ескізів, яка демонструє потенціал машинного навчання та глибоких штучних нейронних мереж в автоматизації категоризації ескізів.

Ця дипломна робота досліджує трансформаційну роль штучного інтелекту (ШІ) у сфері архітектурного проектування та аналізу. Вивчаючи перетин ШІ та архітектури, це дослідження має на меті розкрити потенціал методів ШІ для вдосконалення процесу проектування, покращення експлуатаційних характеристик будівель та оптимізації просторової функціональності.

Робота починається з вивчення фундаментальних концепцій і застосувань ШІ в архітектурі, включаючи генеративний дизайн, розпізнавання образів і обчислювальний аналіз. Вона заглиблюється у розвиток технологій штучного інтелекту та їхню інтеграцію в архітектурне програмне забезпечення та інструменти. Крім того, дослідження вивчає використання алгоритмів ШІ для оптимізації, моделювання та прийняття рішень на основі даних в архітектурному дизайні.

Синтезуючи існуючу літературу, аналізуючи практичні застосування та спираючись на думки експертів, ця дипломна робота надає всебічний огляд поточного стану та майбутніх перспектив ШІ в архітектурному проектуванні та аналізі. Вона має на меті зробити внесок у дискусію про використання технологій штучного інтелекту для створення більш стійкого, ефективного та орієнтованого на людину архітектурного середовища.

Ключові слова: розпізнавання зображень, навчання нейронної мережі, нейронні мережі, машинне навчання, архітектурні креслення.

Annotation

Anna Rieznikova completed final qualification work on the topic "Intelligent system for generating images of design solutions based on man–made graphic data" in the speciality 122 – "Computer Science".

The final qualification work implements a sketch classification system that demonstrates the potential of machine learning and deep artificial neural networks in automating sketch categorisation.

This thesis explores the transformative role of artificial intelligence (AI) in the field of architectural design and analysis. By exploring the intersection of AI and architecture, this research aims to unlock the potential of AI techniques to improve the design process, enhance building performance, and optimise spatial functionality.

It begins with an exploration of the fundamental concepts and applications of AI in architecture, including generative design, pattern recognition, and computational analysis. It delves into the development of AI technologies and their integration into architectural software and tools. In addition, the study explores the use of AI algorithms for optimisation, modelling and data–driven decision–making in architectural design.

By synthesising the existing literature, analysing practical applications and drawing on expert opinions, this thesis provides a comprehensive overview of the current state and future prospects of AI in architectural design and analysis. It aims to contribute to the debate on the use of AI technologies to create more sustainable, efficient and human–centred architectural environments.

Keywords: image recognition, neural network training, neural networks, machine learning, architectural drawings.

ЗМІСТ

Перелік умовних позначень і скорочень

ШІ – стандартизована мова розмітки веб-сторінок;

API (Application Programming Interface) – прикладний програмний інтерфейс;

CNN (Convolutional neural network) – клас глибинних штучних нейронних мереж прямого поширення;

GAN (Generative adversarial networks) – клас алгоритмів штучного інтелекту;

NFR (Non-functional requirement) – вимоги до програмного забезпечення;

IDE (Integrated Development Environment) – Інтегроване середовище розробки;

CSV (Comma-Separated Values) – файл спеціального типу, який можна створити або змінити в Excel;

НМ або ШНМ – штучна нейронна мережа;

СУБД - система управління базами даних.

ВСТУП

Сьогодні розвиток технологій швидко настигає всі галузі життя, а також відкриває нові можливості для їх покращення. Однією з таких галузей є архітектура, де використання сучасних технологій може значно спростити та прискорити процес проектування. Особливо важливим є використання штучного інтелекту в архітектурі, який дозволяє розробляти проекти з високою точністю та швидкістю.

Одним з етапів проектування будь-якої будівлі є створення рукописного ескізу. Важливою складовою цього процесу є здатність перетворювати рукописний ескіз в електронний варіант, що в подальшому може бути оброблено спеціальними прикладними програмами. Для цього можна використовувати системи розпізнавання рукописного тексту, однак вони не є повністю ефективними, оскільки не враховують особливості архітектурного малюнку.

У даній дипломній роботі розроблено інформаційне забезпечення як додаток для інтелектуальної системи, яка реалізує побудову зображень на основі розпізнавання рукописних ескізів в архітектурі та будівництві. Для розпізнавання рукописного малюнку використовувалось машинне навчання, зокрема, нейронні мережі, що дозволяють досягти високої точності розпізнавання. Результатом роботи є програмний додаток, який допомагає архітекторам і дизайнерам ефективно перетворювати рукописні ескізи в електронний варіант, що може бути в подальшому оброблено в спеціалізованих програмах.

РОЗДІЛ 1. Аналіз процесу створення штучної нейронної мережі для розпізнавання рукотворних ескізів

1.1 Використання інтелектуальних технологій в будівництві та архітектурі.

Використання штучного інтелекту (ШІ) в будівництві та архітектурі стає все більш популярним в останні роки. Технологія штучного інтелекту може допомагати в широкому діапазоні завдань, від проектування та планування до управління будівлею та обслуговування. Одним із ключових застосувань штучного інтелекту в будівництві та архітектурі є розпізнавання зображень.

Розпізнавання зображень за допомогою ШІ передбачає використання алгоритмів для аналізу та інтерпретації зображень. Ця технологія має багато практичних застосувань у будівництві та архітектурі, включаючи виявлення потенційних загроз безпеці на будівельному майданчику, виявлення дефектів будівельних матеріалів та оптимізацію експлуатаційних характеристик будівлі. Як приклади того, як ШІ використовується в архітектурі та будівництві можна представити наступні дії:

1. Оптимізація дизайну: Алгоритми ШІ можна використовувати для створення та оптимізації проектів на основі різних критеріїв, таких як енергоефективність, структурна цілісність та естетика. Ці алгоритми можуть навчатися на основі існуючих проектів і даних, а також створювати нові проекти, які відповідають конкретним вимогам. Це може допомогти архітекторам і дизайнерам створювати більш ефективні та стійкі будівлі.

2. Планування і складання графіків будівництва: ШІ можна використовувати для аналізу графіків будівництва, виявлення потенційних затримок або проблем і оптимізації послідовності будівельних робіт . Це може підвищити ефективність будівельного процесу і знизити витрати.

3. Контроль якості та безпека: ШІ можна використовувати для моніторингу будівельних майданчиків на предмет загроз безпеці та контролю якості. Наприклад, алгоритми ШІ можуть аналізувати зображення з дронів або камер, щоб виявити порушення безпеки, наприклад, відсутність у робітників

касок або запобіжних ременів. Вони також можуть виявляти проблеми з якістю, наприклад, тріщини або дефекти в бетоні.

4. Прогнозоване обслуговування: ШІ можна використовувати для прогнозування того, коли будівельні системи, такі як системи опалення, вентиляції та кондиціонування повітря та електричні системи, потребуватимуть технічного обслуговування [1]. Це може допомогти запобігти поломкам і скоротити час простою, а також зменшити витрати на обслуговування.

5. Автоматизація будівель: ШІ можна використовувати для автоматизації систем будівлі, таких як освітлення, опалення та охолодження [1]. Це може підвищити енергоефективність і знизити витрати.

6. Вибір і оптимізація матеріалів: ШІ можна використовувати для вибору та оптимізації будівельних матеріалів на основі різних критеріїв, таких як довговічність, вартість і вплив на навколишнє середовище [1]. Це може допомогти архітекторам і дизайнерам створювати більш стійкі та економічно ефективні будівлі.

Таким чином, штучний інтелект використовується в архітектурі та будівництві для підвищення ефективності, точності та безпеки будівельних проектів, а також для вдосконалення процесу проектування. ШІ можна використовувати для оптимізації дизайну, планування і складання графіків будівництва, контролю якості та безпеки, прогнозного технічного обслуговування, автоматизації будівель, а також для вибору та оптимізації матеріалів.

Постановка проблеми дослідження, пов’язаної з використанням ШІ в будівництві та архітектурі, передбачає ретельний аналіз предметної області. Цей аналіз має враховувати поточний стан галузі, а також будь-які потенційні прогалини чи можливості для подальших досліджень.

1.2 Аналіз існуючих програм, що використовуються в архітектурі

Використання інтелектуальних технологій в архітектурі є відносно новим напрямком розвитку технологій архітектури та будівництва, і наразі на ринку існує кілька інноваційних рішень. Нижче перераховані деякі з них:

- Xnor.ai: Xnor.ai розробляє алгоритми штучного інтелекту для малих пристроїв та вбудованих систем . Один з їх продуктів - DeepX - може використовуватися для аналізу великих даних з веб-сайтів нерухомості та для прогнозування цін на нерухомість.
- Autodesk Dreamcatcher: Autodesk Dreamcatcher - це програма для автоматизованої генерації дизайнів за допомогою штучного інтелекту [2]. Ця програма може створювати дизайни, які відповідають вказаним параметрам, таким як кількість матеріалу, міцність та ергономіка.
- Arctuition: Arctuition - це платформа для прогнозування пожежних ризиків та аналізу пожеж в будівлях за допомогою машинного навчання . Вона використовує навчання з учителем та навчання без учителя, щоб створити прогнози та рекомендації для зменшення ризиків пожежі в будівлях.
- Space Syntax: Space Syntax - це платформа, яка використовує алгоритми машинного навчання для аналізу дизайну приміщень та його впливу на поведінку користувачів [3]. Вона допомагає архітекторам та дизайнерам зрозуміти, які елементи дизайну можуть покращити функціональність та ефективність будівель.

Існує багато додатків і програм на основі інтелектуальних технологій, які користуються популярністю серед архітекторів і дизайнерів. Найбільш значимі ключові переваги даних програм полягають у наступному.

Підвищена ефективність: інструменти ШІ можуть автоматизувати повторювані завдання та створювати оптимізовані проекти на основі конкретних параметрів, заощаджуючи час і зусилля архітекторів і дизайнерів.

Покращена точність: алгоритми на основі інтелектуальних технологій можуть аналізувати великі обсяги даних і виявляти закономірності, які людям важко виявити, створюючи більш точні проекти та моделі.

Краща продуктивність: штучний інтелект може допомогти архітекторам створювати кращі та більш стійкі будівлі шляхом оптимізації проектів на основі конкретних критеріїв, таких як енергоефективність або цілісність конструкції.

Покращена креативність: інструменти генеративного проектування на основі інтелектуальних технологій можуть генерувати велику кількість дизайнерських рішень, які неможливо уявити людям, дозволяючи архітекторам досліджувати нові та інноваційні дизайнерські ідеї.

Серед популярних програм архітектури ШІ – інструмент генеративного проектування Autodesk, який використовує алгоритми штучного інтелекту для оптимізації проектів на основі конкретних критеріїв продуктивності, і Генератор стилів SketchUp, який використовує ШІ для створення власних стилів для 3D-моделей. Іншим популярним додатком є BIM 360, який автоматизує робочі процеси і оптимізує графіки будівництва.

Загалом використання додатків з елементами штучного інтелекту в будівництві може допомогти архітекторам і дизайнерам працювати ефективніше, точніше та креативніше, що призведе до більш ефективних та екологічно чистих будівель.

1.3 Аналіз основних процесів предметного середовища

Класифікація зображень – це одна з фундаментальних задач в області штучного інтелекту. Це процес розподілу зображень на заздалегідь визначені класи або категорії на основі їхнього візуального змісту. Це фундаментальне завдання комп'ютерного зору, яке широко використовується в різних додатках, таких як розпізнавання об'єктів, розпізнавання облич, аналіз медичних зображень, безпілотні автомобілі та інші застосування [4].

Популярність класифікації зображень в основному пояснюється наступними причинами:

1. Збільшення доступності даних про зображення: З розвитком інтернету та соціальних мереж кількість доступних даних про зображення різко зросла. Класифікація зображень дозволяє автоматично аналізувати і класифікувати ці зображення, робляючи їх більш корисними для різних застосувань.

2. Швидкий розвиток алгоритмів глибокого навчання: Алгоритми глибокого навчання, зокрема, згорткові нейронні мережі (CNN), продемонстрували чудову продуктивність у задачах класифікації зображень [5]. Ці алгоритми можуть автоматично вивчати ознаки з необроблених даних зображень і класифікувати зображення з високою точністю.

3. Застосування в різних галузях: Класифікація зображень має численні практичні застосування, включаючи медичну діагностику, спостереження за безпекою та автономними транспортними засобами. Таким чином, вона стала важливим інструментом у багатьох галузях промисловості.

4. Простота впровадження: Завдяки наявності фреймворків глибокого навчання з відкритим кодом, таких як TensorFlow і PyTorch, дослідникам і розробникам стало простіше впроваджувати моделі класифікації зображень.

Отже, задача класифікації зображень є популярною та актуальною [6], оскільки вона дає змогу автоматизувати аналіз і категоризацію величезних обсягів даних зображень і має численні застосування в різних галузях. Швидкий розвиток алгоритмів глибокого навчання і простота реалізації ще більше сприяли її широкому використанню.

Ще один метод, який використовується в роботі – це метод генерації зображень за допомогою алгоримтів, які розроблені на основі інтелектуальних технологій з використанням згорткових штучних нейронних мереж.

Автоматична генерація зображень - це процес, під час якого засоби, що реалізують відповідні алгоритми, навчаються генерувати нові зображення на основі набору вхідних даних. Дана технологія набула популярності в останні роки завдяки своїй здатності створювати реалістичні зображення, які імітують зовнішній вигляд реальних об'єктів і сцен. Описані технології автоматичної генерації зображень на основі рукописних ескізів або текстового опису креслень є підрозділом технологій генеративного інтелекту.

Процес автоматичної генерації зображень зазвичай включає тип нейронної мережі, відомий як генеративна змагальна мережа (GAN) [7]. Модель GAN складається з двох нейронних мереж: генератора та дискримінатора. Генератор отримує на вхід випадковий вектор шуму (рис. 1.1) і виводить зображення, яке нагадує навчальні дані. Дискримінатор приймає зображення на вхід і видає ймовірну оцінку, яка вказує, чи є воно справжнім або підробленим (тобто, згенерованим генератором).

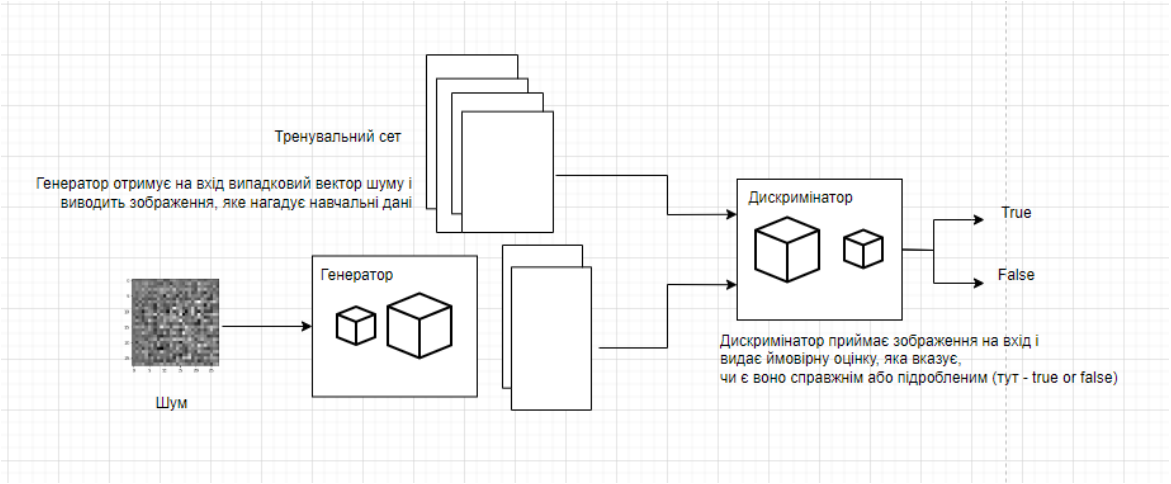

Рисунок 1.1 - Відношення генератора та дискримінатора

Під час навчання генератор намагається «обдурити» дискримінатор, генеруючи все більш реалістичні зображення, які дискримінатор приймає за справжні. У той же час дискримінатор намагається правильно визначити, які зображення є справжніми, а які фальшивими, і надає генератору зворотній

зв'язок про те, як покращити згенеровані ним зображення. Цей зворотно-поступальний процес триває доти, доки генератор не зможе генерувати зображення, які неможливо відрізнити від справжніх, а дискримінатор більше не зможе з високою впевненістю розрізняти справжні та згенеровані зображення.

Щоб використати цю модель GAN для нашої програми генерації зображень, ми навчимо її на наборі даних ескізів і відповідних реальних зображень. Після навчання ми можемо ввести ескіз у генератор, який видасть відповідне згенероване зображення, схоже на реальне. Це дозволяє користувачам швидко і легко створювати реалістичні зображення на основі власних ескізів, без необхідності мати просунуті навички малювання або знання програмного забезпечення для 3D-моделювання.

Існує багато потенційних застосувань для зображень, згенерованих такими та подібним засобами, в різних галузях, зокрема в архітектурі та дизайні. Наприклад, архітектори можуть використовувати зображення, згенеровані програмними застосунками на основі штучних нейронних мереж, для створення більш реалістичних візуалізацій своїх проектів ще до початку будівництва. Дизайнери інтер'єру можуть використовувати згенеровані зображення для попереднього перегляду різних колірних схем і розташування меблів у приміщенні, перш ніж вносити будь-які фізичні зміни. Крім того, художники та кінематографісти можуть використовувати автоматично створені зображення, для створення унікальних візуальних ефектів і цифрових творів мистецтва.

Процес автоматичної генерації зображень за допомогою програмних даданків на основі ШНМ має потенціал революціонізувати спосіб створення та візуалізації нових дизайнів, пропонуючи потужний інструмент для творчих професіоналів, щоб досліджувати нові можливості та розширювати межі можливого.

Створення системи для класифікації та обробки зображень потребує чітко поставленої задачі, яку необхідно вирішувати. Для початку створення

додатку необхідно проаналізувати основні процеси роботи алгоритму обробки та класифікації зображень.

Перший етап - це створення контекстної діаграми “ЯК БУДЕ” (рис. 1.2), який показує процес відображення найбільш схожого зображення. На вхід система отримує ескіз планування, намальований від руки користувачем та на вихід ця система має відправити користувачу зображення готового згенерованого планування. Механізмами у даній системі є користувач – людина яка буде завантажувати своє зображення в додаток, та функції обробки та класифікації зображень, яка виявляє схоже зображення в базі даних та видає результат, який має найбільшу схожість з картинкою користувача.

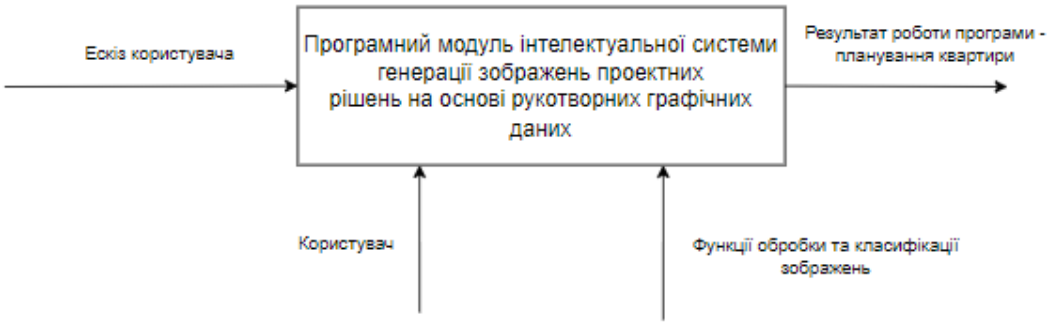

Рисунок 1.2 – Контекстна діаграма «ЯК БУДЕ»

Більш детально розглянути процес, який описаний раніше можливо завдяки декомпозиції основних функцій діаграми «ЯК БУДЕ» (рис. 1.3). Після отримання зображення система аналізує зображення для отримання інформації про основні риси або ж деталі цього рисунку. Далі система аналізує чи є в базі даних подібне за характеристичними ознаками або деталями зображення.

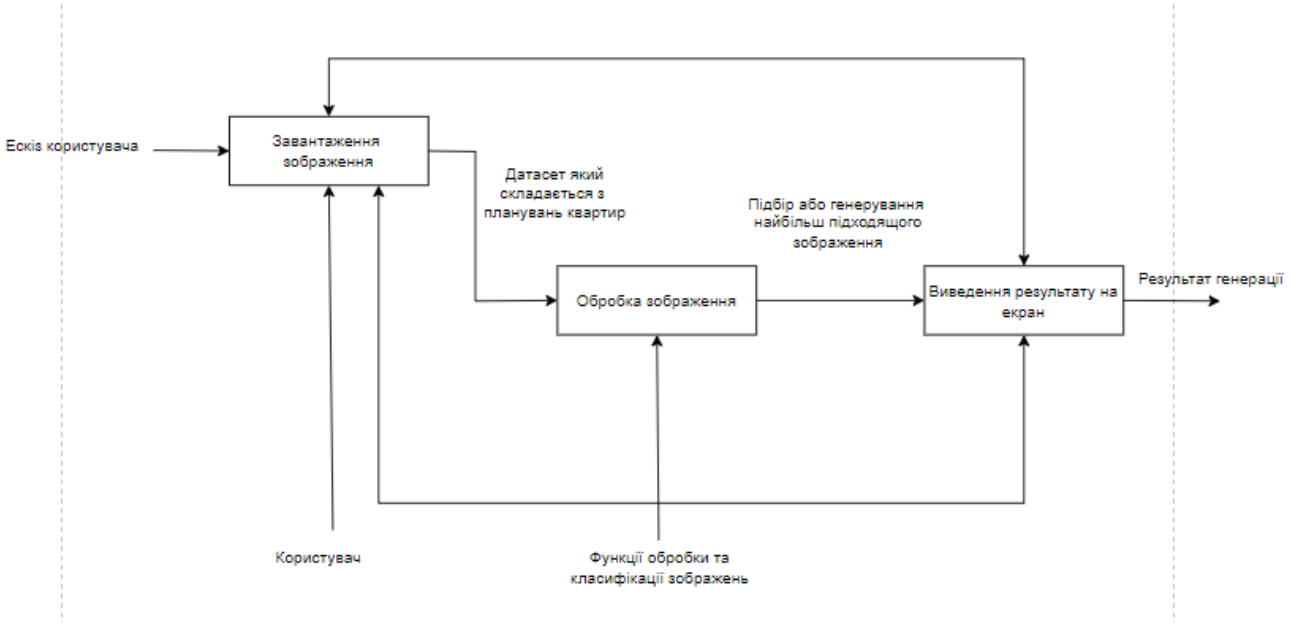

Рисунок 1.3 – Декомпозиція основних функцій діаграми «ЯК БУДЕ».

Для декомпозиції контекстної діаграми «ЯК БУДЕ» за моделлю IDEF0 визначимо два способи використання програмного модулю – розпізнавання ескізу користувача та навчання нейронної мережі (рис. 1.4).

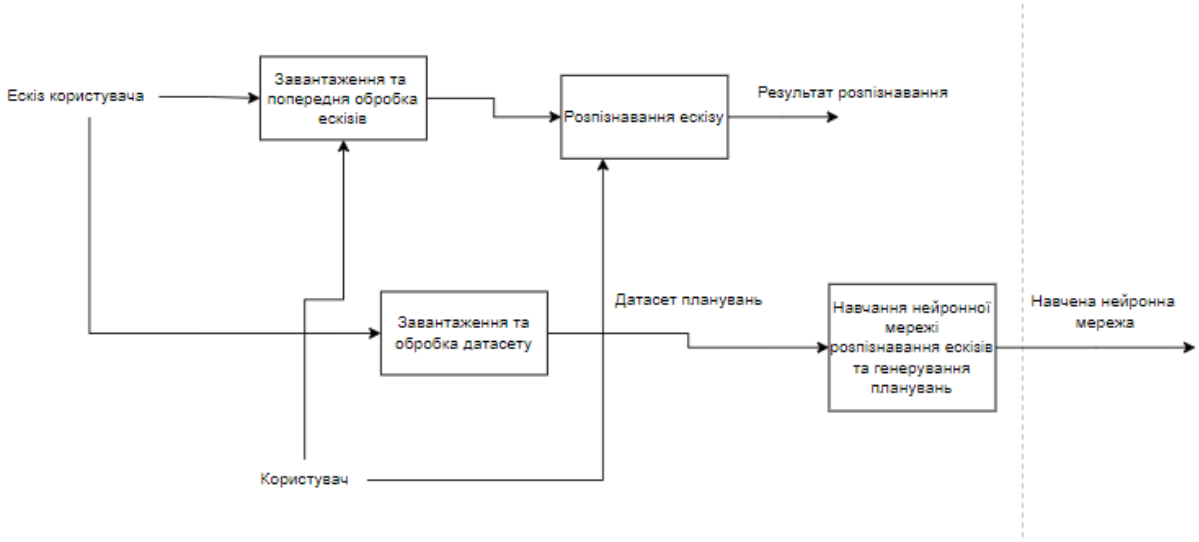

Рисунок 1.4 – Діаграма IDEF0 програмного модуля розпізнавання ескізів плануваня рівня 1

Розпізнавання ескізів користувача передбачає такі процеси:

- Завантаження та попередня обробка ескізів;

- Розпізнавання ескізів.

На виході отримаємо результат розпізнавання. Навчання нейронної мережі передбачає такі процеси:

- Завантаження та обробка датасету;
- Навчання нейронної мережі розпізнавання та генерації планувань.

На виході отримуємо навчену нейронну мережу.

На зображені 1.5 можна побачити діаграму IDEF0 рівня 1 процесу завантаження та попередньої обробки ескізів користувача.

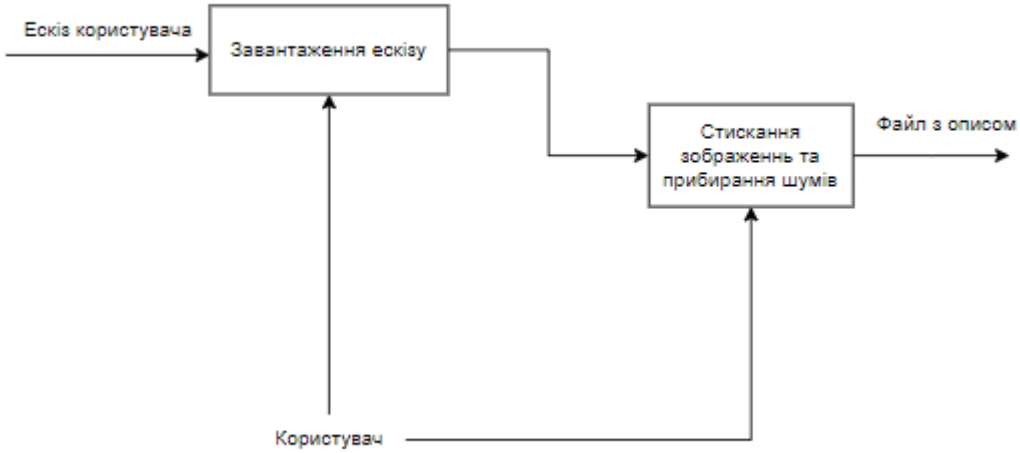

Рисунок 1.5 - діаграма IDEF0 рівня 1 процесу завантаження та попередньої обробки ескізів користувача

На рисунку 1.6 зображено діаграму діаграму IDEF0 рівня 1 результату декомпозиції навчання нейронної мережі.

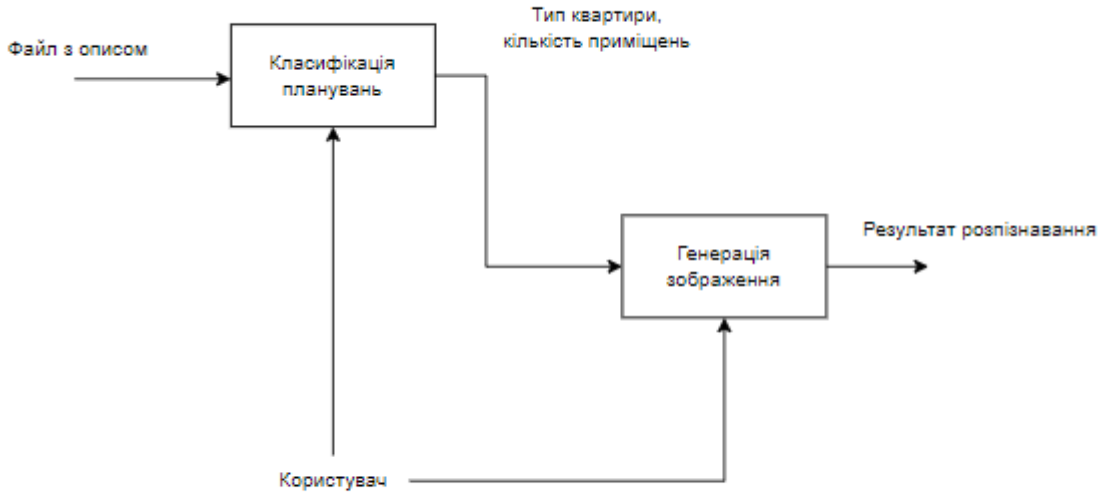

Рисунок 1.6 - діаграма IDEF0 рівня 1 результату декомпозиції навчання нейронної мережі

На рисунку 1.7 зображено діаграму IDEF0 рівня 1 процесу завантаження та обробки датасету.

Рисунок 1.7- діаграма IDEF0 рівня 1 процесу завантаження та обробки датасету

На рисунку 1.8 зображено діаграму IDEF0 рівня 1 процесу навчання нейронної мережі генерації зображень проектних рішень на основі рукотворних графічних даних.

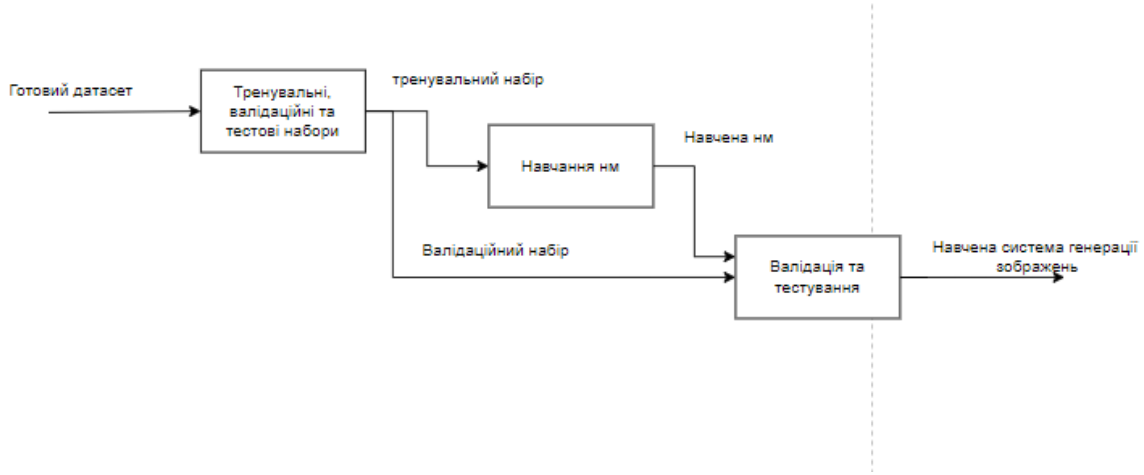

Рисунок 1.8 - діаграма IDEF0 рівня 1 процесу навчання нейронної мережі генерації зображень проектних рішень на основі рукотворних графічних даних.

1.3.2 Аналіз бізнес процесів та бізнес архітектури предметної області сканування штучного інтелекту

Аналіз бізнес процесів та бізнес архітектури для сканування за допомогою штучного інтелекту може бути корисним для розуміння того, як цей процес може бути оптимізований та вдосконалений. Нижче описані основні етапи бізнес аналізу для даної предметної області:

1. Визначення предметної області: Першим етапом аналізу бізнес-процесів є визначення предметної області. В даному випадку це аналіз зображення із застосуванням інтелектуальних технологій обробки та розпізнавання зображень. Визначення предметної області допомагає зрозуміти, які завдання потрібно вирішити та яку функціональність має мати програмне забезпечення.
2. Визначення бізнес-процесів: Наступним етапом є визначення бізнес-процесів. Це дозволяє зрозуміти, які кроки повинні бути виконані розробленим застосунком, від отримання зображення до його обробки та розпізнавання.

3. Аналіз та оптимізація бізнес-процесів: Після визначення бізнес-процесів можна проаналізувати кожен крок та знайти способи його оптимізації. Наприклад, можна зменшити час обробки зображення або поліпшити точність розпізнавання.
4. Визначення бізнес-архітектури: Після аналізу бізнес-процесів можна визначити бізнес-архітектуру. Це описує структуру програмного забезпечення та його компоненти. Бізнес-архітектура може допомогти в розробці програмного забезпечення та визначенні вимог до нього.
5. Визначення бізнес-вимог: Наступним етапом є визначення бізнес-вимог. Це допомагає зрозуміти, які функціональність має мати програмне забезпечення, щоб відповідати бізнес-потребам. Наприклад, програмне забезпечення повинно бути здатним розпізнавати різні типи зображень, досить швидко оброблювати дані та забезпечувати високу точність розпізнавання.
6. Визначення технічних вимог: Останнім етапом є визначення технічних вимог. Це описує технічні характеристики програмного забезпечення, такі як мова програмування, платформа, база даних та інші. Визначення технічних вимог допомагає розробникам програмного забезпечення створити ефективну архітектуру та програмний код.

Після визначення бізнес-вимог та технічних вимог можна розпочати розробку програмного забезпечення для сканування штучного інтелекту. При цьому важливо врахувати всі вимоги, які були визначені на попередніх етапах бізнес-аналізу.

Крім того, важливо знати, що бізнес-архітектура та бізнес-аналіз можуть бути періодично переглянуті та оновлені, оскільки вимоги бізнесу можуть змінюватись з часом. Тому, для успішної розробки та використання програмного забезпечення для сканування штучного інтелекту, необхідно мати чітку стратегію бізнес-аналізу та управління проектами.

1.4 Постановка задачі для створення застосунку

Постановка задачі для створення застосунку полягає в розробці програмного забезпечення для роботи з штучною нейронною мережею згорткового типу, яке забезпечуватиме можливість автоматичної генерації зображень планування будівлі на основі рукописних ескізів, завантажених користувачем у застосунок.

Для досягнення цієї мети, розроблено програмний модуль, який здійснює обробку та аналіз вхідних даних - рукописних ескізів будівлі, завантажених користувачем у застосунок. На основі поставленої задачі та описаних даних необхідно розробити модель штучної нейронної мережі з використанням алгоритмів машинного навчання, яка зможе навчатися розпізнавати форми та контури будівель з рукописних ескізів.

Навчена модель повинна задовільняти розробленому алгоритму автоматичної генерації зображень планування будівлі на основі рукописних ескізів, який буде використовувати навчену модель. Для ефективної роботи користувача з програмним застосунком важливо розробити користувацький інтерфейс, в якому передбачена можливість завантаження рукописних ескізів.

Отже, головною метою розробки даного застосунку є забезпечення користувачам можливості автоматичної генерації зображень планування будівель на основі їх рукописних ескізів, що дозволить збільшити продуктивність та ефективність процесу проектування будівель.

1.4.1 Функціональні вимоги

Функціональні вимоги [8] визначають, що повинна робити програмна система. Він визначає функцію програмної системи або її модуля. Функціональність вимірюється як набір входів в систему, що перевіряється, на виході з системи.

Реалізація функціональних вимог в системі планується на етапі проектування системи, тоді як, у випадку нефункціональних вимог, це планується в документі Архітектура системи. Функціональна вимога підтримує генерування нефункціональних вимог.

До функціональних вимог можна віднести:

1. Можливість приймати зображення від користувача.
2. Система повинна обробляти ескізи користувачів.
3. Система повинна знаходити серед датасету зображення, яке найбільш схоже на ескіз користувача (за заданими метричними характеристиками) та відправляти його в клієнтську частину додатку.
4. Система має підраховувати час, за який виконується обробка, пошук та відображення результату.
5. Підтримка різноманітних форматів вхідних даних, таких як скановані зображення, фотографії або векторні малюнки.
6. Користувач повинен мати змогу завантажити результат.
7. Підтримка можливості коригування та редагування результатів розпізнавання.
8. Підтримка інтеграції з іншими системами та додатками.

1.4.2 Нефункціональні вимоги.

Нефункціональні вимоги (NFR) – це обмеження або вимоги, що накладаються на систему [8]. Вони визначають атрибут якості програмного забезпечення. Нефункціональні вимоги стосуються таких проблем, як масштабованість, зручність обслуговування, продуктивність, портативність, безпека, надійність і багато інших. Нефункціональні вимоги стосуються життєво важливих питань якості програмних систем.

До нефункціональних вимог можна віднести:

1. Система повинна мати інтуїтивно зрозумілий інтерфейс.

2. Помилка при розпізнаванні має бути не більше 10%.
3. Має бути зібраний великий та різноманітний датасет.
4. Продуктивність та швидкість роботи застосунку.
5. Надійність та стійкість до помилок та збоїв.
6. Зручність та простота використання застосунку для користувачів.
7. Сумісність з різними операційними системами та пристроями.
8. Цілісність.

1.5 Висновки до першого розділу

Використання програмного застосунку для ШНМ в будівництві та архітектурі має великий потенціал для підвищення ефективності, безпеки та продуктивності в цих галузях. Однак ретельний розгляд дослідницької проблеми та функціональних і нефункціональних вимог необхідний для забезпечення ефективного та відповідального використання технології ШІ. Майбутні дослідження в цій галузі можуть суттєво вплинути на те, як ми проектуємо, створюємо та керуємо нашим архітектурним середовищем.

Для реалізації задуманого проекту, а саме - системи побудови зображень на основі розпізнавання рукописних графічних ескізів, було виконано декілька важливих кроків - спочатку це була розробка діаграми, яка пояснює процес виконання алгоритму, що відображений через діаграми “ЯК Є”. Потім визначались функціональні та нефункціональні вимоги, для того щоб краще розуміти задачу, яка стоїть перед виконанням цієї роботи.

РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ПРОЕКТНИХ РІШЕНЬ НА ОСНОВІ РУКОТВОРНИХ ГРАФІЧНИХ ДАНИХ

2.1 Розробка архітектури

2.1.1 Функціональний аналіз

Функціональний аналіз є важливим етапом у розробці додатку штучного інтелекту, який дозволяє визначити його основні функціональні вимоги та можливості. Основною метою функціонального аналізу є визначення того, які функції має виконувати програмне забезпечення, щоб задовольнити вимоги користувачів та бізнес-потреби.

Основні функціональні вимоги до програмного додатку штучної нейронної мережі можуть включати такі функції:

- Зчитування та обробка зображень: Однією з основних функцій додатку є зчитування та обробка зображень, що вводяться користувачем. Для цього можна використовувати навчену модель, яка буде визначати, що зображено на зображенні.

- Класифікація зображень: Додаток може класифікувати зображення на основі попередньої обробки та алгоритму навченої ШНМ. Наприклад, додаток може визначати, що на зображенні зображено будівлю, інтер'єр або елементи дизайну.

- Пошук інформації: Додаток може використовуватись для пошуку інформації про будівлі, дизайнерів, стилі та інші аспекти. Наприклад, можна розробити функцію, яка дозволяє знайти інформацію про будь-яку будівлю на основі зображення.

Для отримання повної картини архітектури інтелектуальної системи для побудови зображень на основі розпізнавання рукописних ескізів, необхідно розробити дерево функцій (рис. 2.1) для відображення базових клієнтських та адміністративних функцій.

Вершина дерева функцій – модуль пошуку та аналізу ескізів, який включає в собі основні функції: клієнтські та адміністративні функції.

Клієнтські функції включають 5 елементів: Завантаження ескізів, обробка ескізів, розпізнавання ескізів та генерація планувань та завантаження згенерованого зображень.

До адміністративних функцій входять функції роботи з датасетом та функції навчання системи. Функції роботи з датасетом включають 6 елементів: завантаження планувань, обробка зображення, ресайз зображення, класифікація планувань квартир за різними параметрами, збереження датасету, а також можливість збільшення кількості еталонних зображень.

Функції навчання системи включає 4 елементи: обробка даних, налаштування, навчання та тестування НМ.

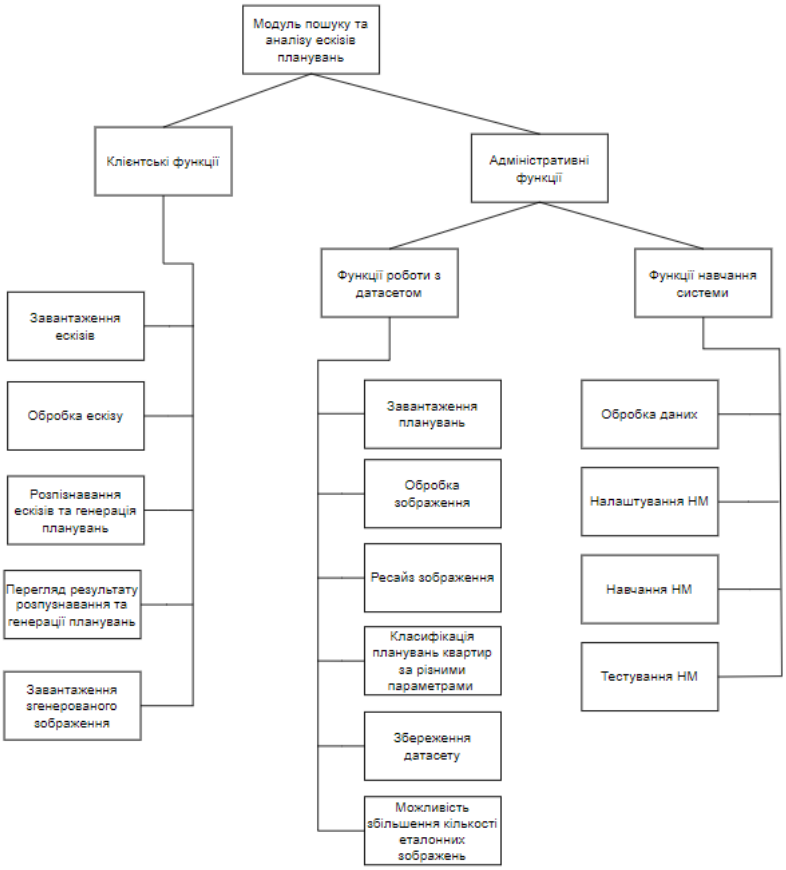

Рисунок 2.1 – Дерево функцій

Для кращого розуміння побудуємо діаграму варіантів використання (рис. 2.2). Діаграма варіантів використання (Use Case Diagram) - це графічне зображення, яке моделює взаємодію акторів (користувачів) з системою та її функціональні можливості. Вона допомагає визначити основні сценарії використання системи та взаємодію між акторами та функціями системи.

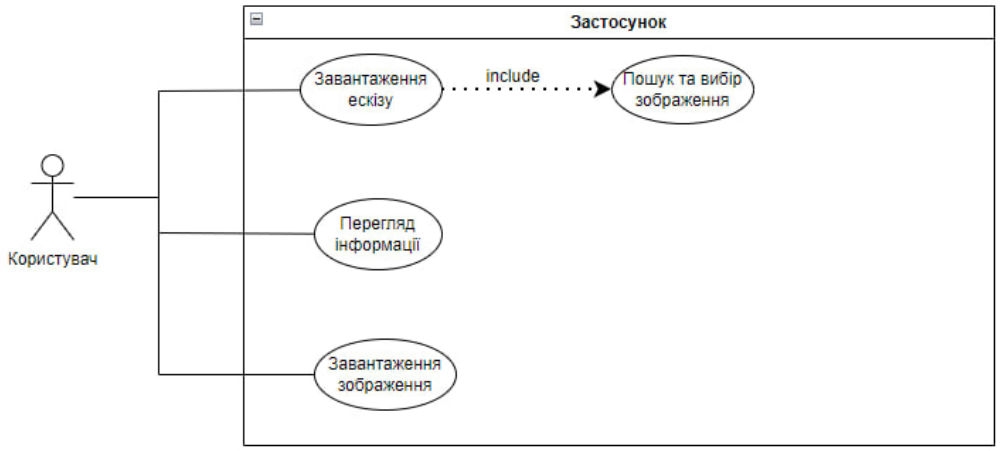

Рисунок 2.2 – діаграма варіантів використання

Варіанти використання:

- Завантаження ескізу;
- Перегляд інформації;
- Завантаження зображення.

Сценарії:

Основні сценарії:

1. Користувач відкриває застосунок;
2. Користувач завантажує свій ескіз;
3. Користувач переглядає інформацію;
4. Користувач отримує в результаті готове зображення, яке може завантажити.

Альтернативні сценарії:

1. Якщо користувач вибрав помилково невірний ескіз, то він може переобрати його завантаживши інший;
2. Якщо користувач випадково перезавантажив сторінку, то його обраний ескіз та згенероване зображення зберігаються на сторінці.

Користувач взаємодіє з програмою, завантажуючи зображення ескізу (рис. 2.3). Потім програма використовує навчену модель машинного навчання, щоб класифікувати ескіз і створити відповідне архітектурне креслення. Користувач може візуалізувати та завантажити отримане креслення для подальшого використання або аналізу. Програма спрощує процес перетворення ескізів на архітектурні креслення, забезпечуючи ефективне та зручне рішення для завдань архітектурного проектування. Наступним етапом є опис функціонування додатку.

Рисунок 2.3 – Діаграма діяльності користувача

Початком роботи є відкритий додаток який завантажує інтерфейс додатку. Наступним кроком є генерування поля для прийняття зображення. Користувач завантажує зображення яке необхідно проаналізувати. На виході система видає результат у вигляді зображення, яке може бути завантажене користувачем (рис. 2.4).

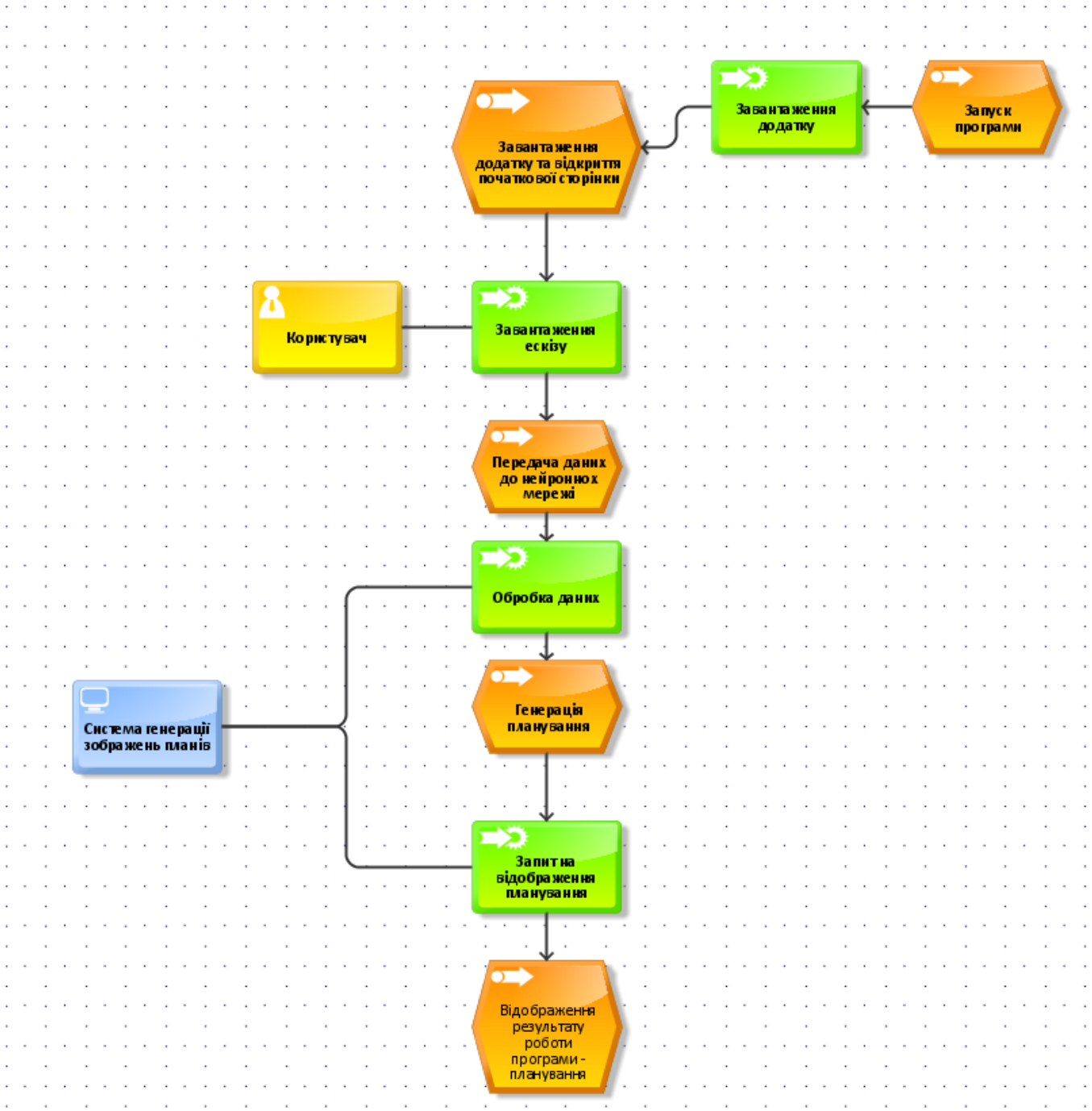

Рисунок 2.4 – Діаграма Event-Driven Process Chain для функціонування додатку

2.2 Розробка інформаційного забезпечення

Загальна структура веб-додатку складається з фронтенду та бекенду, які взаємодіють між собою для забезпечення повноцінного функціонування системи. Така двошарова структура додатку дозволяє розділити логіку від представлення та забезпечує гнучкість та масштабованість.

Фронтенд є частиною додатку, з якою користувач взаємодіє безпосередньо. Фронтенд створюваного додатку складається з модулю генерації сторінок.

Бекенд є серверною частиною додатку, яка відповідає за обробку логіки, зберігання та доступ до даних, а також взаємодію з іншими системами або сервісами. Він забезпечує обробку запитів, аутентифікацію та авторизацію користувачів, логіку бізнес-процесів та виконання операцій, необхідних для функціонування системи. Бекенд створюваного додатку складається з модулю нейромережі, який виконує три основні функції:

- Обробка зображень;
- Розпізнавання зображень;
- Пошук та генерація зображень.

Дана навчальна система для правильної роботи потребує правильно розподілені дані. Для початку розробимо схему взіємодії підсистем з базою даних (рис. 2.5). Схема взаємодії підсистем з базою даних відображає спосіб, яким різні компоненти системи взаємодіють з базою даних для зберігання, отримання та оновлення інформації.

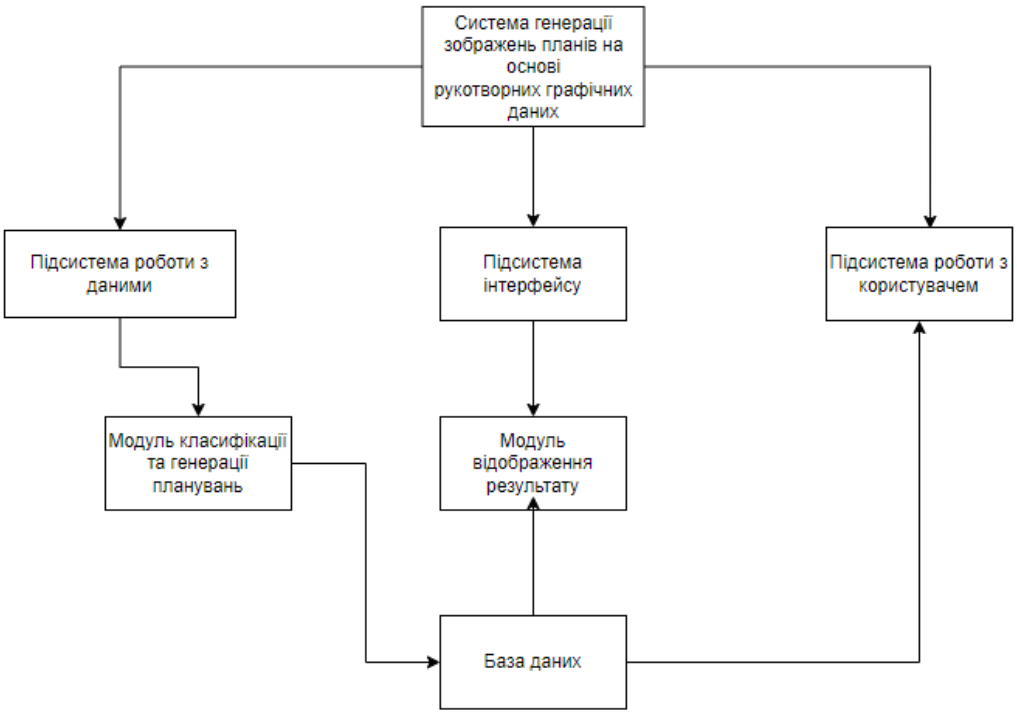

Рисунок 2.5 – Схема взаємодії підсистем з базою даних

Наступним кроком розробимо концептуальну модель бази даних (рис. 2.6). На ній для заданої предметної області відображено наступні сутності:

- Зображення планувань квартир;
- Планування квартири;
- Додаткова інформація про кожен план;
- Ескіз користувача.

У концептуальній моделі бази даних були виокремлені такі сутності:

Зображення планування квартири - сутність, яка містить зображення кожного плану квартири. Ця сутність має такі атрибути, як ідентифікатор зображення, шлях до файла з зображенням, дата створення, розмір файлу.

План квартири - сутність, яка містить інформацію про кожен план, збережений у вигляді зображення. Ця сутність має такі атрибути, як

ідентифікатор плану, дата створення, назва проекту, розміри квартири, кількість кімнат, тип планування.

Додаткова інформація про кожен план - сутність, яка містить додаткову інформацію про кожен план квартири, таку як розміри квартири, кількість кімнат, тип планування тощо. Ця сутність має такі атрибути, як ідентифікатор додаткової інформації, ідентифікатор плану, розміри квартири, кількість кімнат, кількість кутів, кількість вікон та дверей, тип планування.

Ескіз користувача – сутність, яка містить зображення, що надсилає користувач. Ця сутність має такі атрибути, як ідентифікатор зображення, дата створення, розмір файлу, назва.

Зв'язок між сутностями:

У концептуальній моделі бази даних існує прямий зв'язок між сутностями "План квартири" та "Зображення планування квартири", оскільки кожен план квартири повинен мати принаймні одне зображення планування, але зв'язок між "Додаткова інформація про кожен план" та іншими сутностями може бути менш прямим.

Кожен план квартири повинен мати ідентифікатор, який є унікальним для кожного запису в таблиці "План квартири". Цей ідентифікатор може використовуватися як зовнішній ключ у таблиці "Зображення планування квартири", щоб встановити зв'язок між зображеннями та відповідними планами квартир.

Також ідентифікатор плану може бути використаний як зовнішній ключ у таблиці "Додаткова інформація про кожен план", щоб встановити зв'язок між інформацією про кожен план та відповідними планами квартир.

Отже, зв'язок між цими сутностями може бути встановлений за допомогою зовнішніх ключів, що забезпечить цілісність даних та дозволить ефективно здійснювати запити до бази даних.

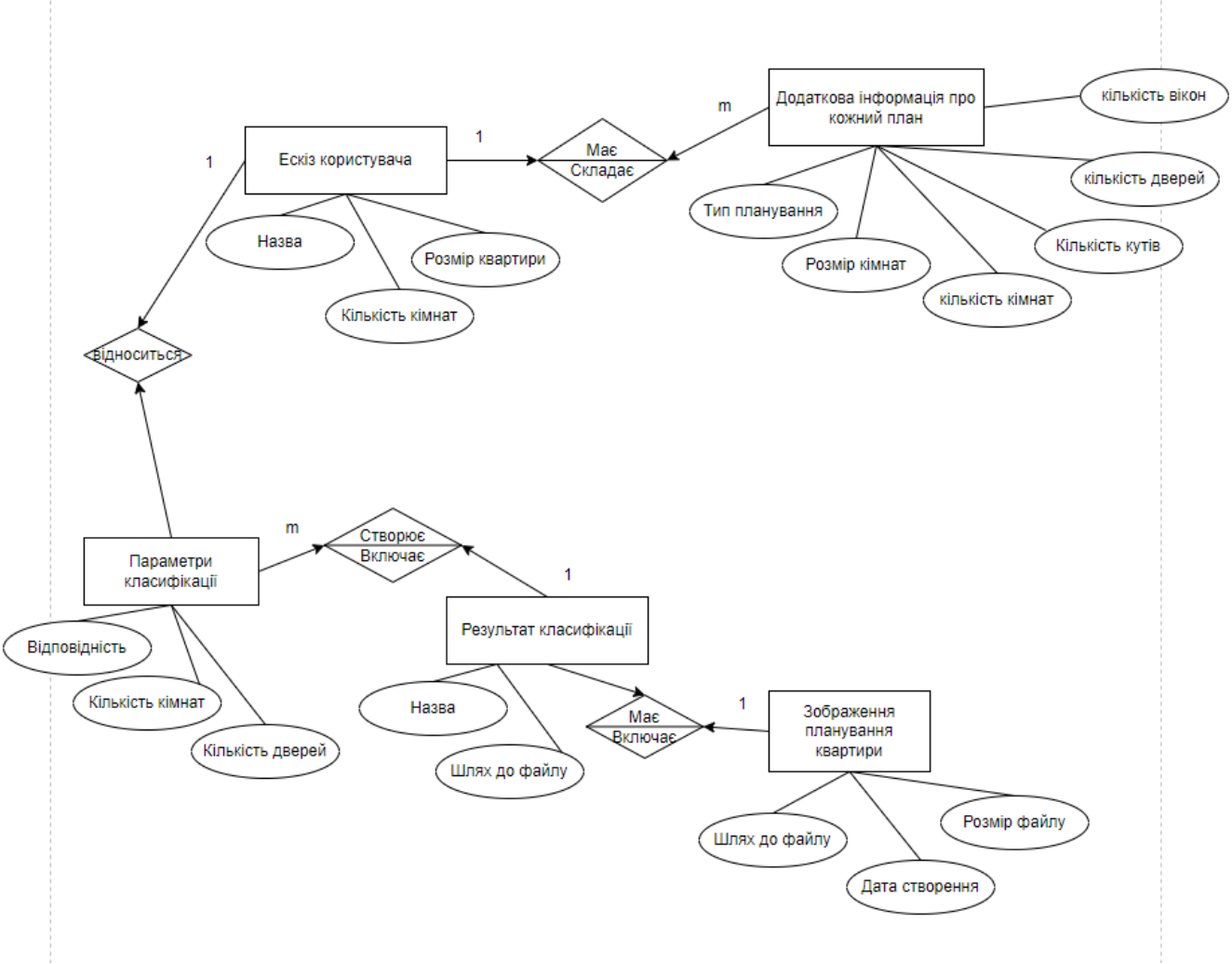

Рисунок 2.6 – Концептуальна модель бази даних

Наступною є логічна модель, яка більш детально відображає поля бази даних, що зберігаються в системі.

У логічній моделі бази даних були створені такі таблиці (рис. 2.7):

Таблиця "Плани квартир", яка містить ідентифікатори та описи кожного плану квартири.

image_name (ідентифікатор плану квартири)

Name (ідентифікатор плану квартири, унікальний для кожного запису)

image_date (дата)

Таблиця "Ескіз користувача", яка містить зображення кожного плану квартири.

id_image (ідентифікатор зображення, унікальний для кожного запису)

Name (унікальне ім’я)

Date (дата)

Img_res (деталізація зображення)

Таблиця "Додаткова інформація про кожен план" (Scetch parameter), яка містить додаткову інформацію про кожен план квартири.

id_info (ідентифікатор інформації про план, унікальний для кожного запису)

id_plan (зовнішній ключ, що вказує на ідентифікатор плану квартири)

floor_area (площа квартири)

number_of_rooms (кількість кімнат)

number_of_angles (кількість кутів)

windows doors (вікна та двері)

Таблиця "Параметр класифікації", яка містить інформацію про зображення, яке завантажується користувачем.

id_result (ідентифікатор зображення, унікальний для кожного запису)

number_of_rooms (кількість кімнат)

plan_probab (можливий результат)

number_of_angles (кількість кутів)

Таблиця "Результат класифікації", яка містить інформацію про зображення, яке завантажується користувачем.

id_image (ідентифікатор зображення)

Name (ім'я, унікальне для кожного запису)

Зв'язок між таблицями може бути встановлений за допомогою зовнішніх ключів, які забезпечать цілісність даних та дозволять ефективно здійснювати запити до бази даних.

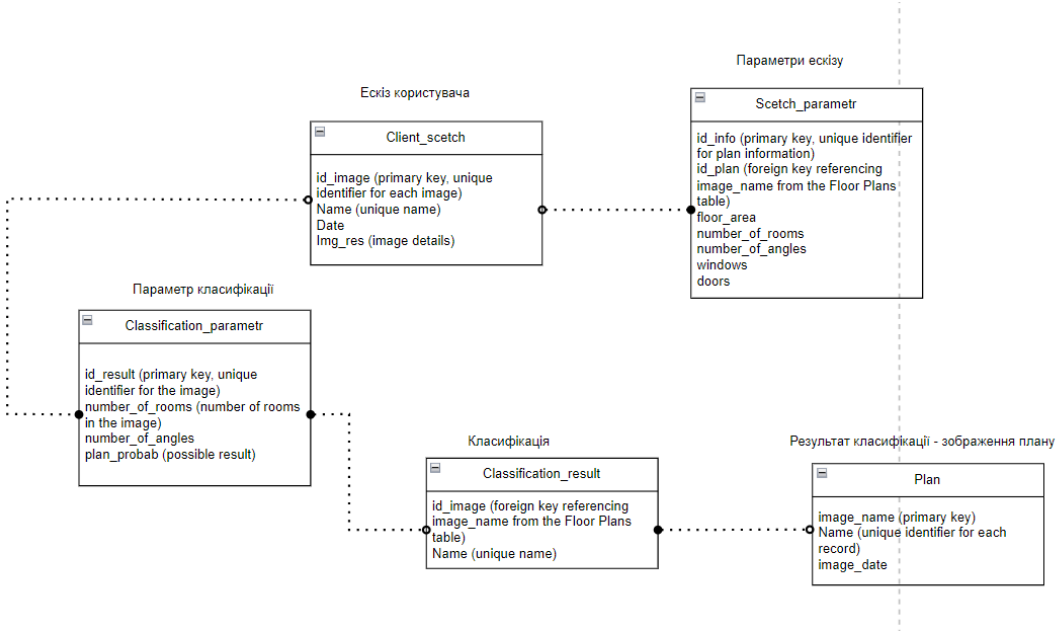

Рисунок 2.7 – Логічна модель бази даних

Отже, логічна база даних для зберігання зображень планувань квартир може мати 5 таблиць, які містять інформацію про ескізи користувача, плани квартир, додаткову інформацію про кожен ескіз, параметр класифікації та результат.

Останньою є фізична модель бази даних, розроблена на основі даталогічної. Фізична модель бази даних визначає фактичну реалізацію бази даних у конкретній системі управління базами даних (СУБД). Вона описує структуру таблиць, індексів, зв'язків між таблицями та інші технічні деталі, необхідні для зберігання та оптимізації доступу до даних.

Для фізичної реалізації бази даних, описаної у логічній моделі, можна використати СУБД, яка підтримує реляційну модель даних, наприклад, MySQL або PostgreSQL.

У фізичній моделі бази даних були створені такі таблиці (рис. 2.8):

Таблиця "Плани квартир" для зберігання ідентифікаторів та описів планів квартир.

image_name (первинний ключ)

Name (унікальний ідентифікатор для кожного запису)

image_date

Таблиця "Ескізи користувачів":

id_image (первинний ключ, унікальний ідентифікатор для кожного зображення)

Name (унікальна назва)

Date

Img_res (деталі зображення)

Таблиця "Додаткова інформація про план" (Sketch_parameter):

id_info (первинний ключ, унікальний ідентифікатор для інформації про план)

id_plan (зовнішній ключ, що посилається на ім'я зображення image_name з таблиці "Плани поверхів")

floor_area

number_of_rooms

number_of_angles

windows

doors

Таблиця "Параметр класифікації":

id_result (первинний ключ, унікальний ідентифікатор зображення)

number_of_rooms (кількість кімнат на зображенні)

plan_probab (можливий результат)

Таблиця "Результат класифікації":

id_image (зовнішній ключ, що посилається на ім'я зображення image_name з таблиці "Плани поверхів")

Name (унікальна назва)

Для забезпечення ефективного доступу до даних можна створити індекси на полях, які використовуються для пошуку та зв'язку даних між таблицями.

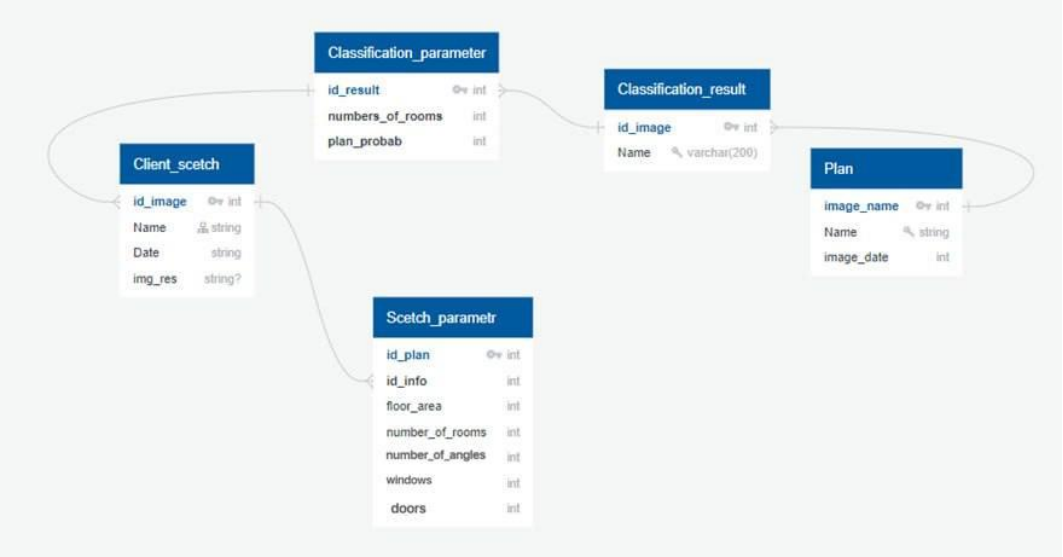

Рисунок 2.8 – Фізична модель бази даних

Отже, фізична модель бази даних для зберігання зображень планування квартир містить три таблиці, які містять інформацію про плани квартир, зображення та додаткову інформацію про кожен план, індекси та зв'язки між таблицями.

2.3 Висновки до другого розділу

В результаті проведення етапу розробки інтелектуальної системи для побудови зображень на основі розпізнавання рукописних ескізів було розроблено та спроектовано систему для подальшої її реалізації.

За результатами функціонального аналізу побудовано дерево функцій, яке відображає бізнес-процеси інформаційної системи. Розроблено діаграми Event-Driven Process Chain, які показують процес проектування та опису логіки роботи боту.

На основі отриманих результатів аналізу сформовано архітектуру системи для відображення її внутрішніх компонентів та їх взаємозв'язок.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ПРОЕКТНИХ РІШЕНЬ НА ОСНОВІ РУКОТВОРНИХ ГРАФІЧНИХ ДАНИХ

3.1 Загальний опис методів використаних в додатку

Для початку створення додатку потрібно обрати середовище програмування. Через крос-платформну сумісність, можливість інтегрування з іншими мовами програмування та широку екосистему бібліотек було обрано середовище Python. Для початку роботи потрібно встановити Python у системі. Завантажити та встановити Python можна з офіційного сайту Python [9].

Після встановлення Python можна використовувати будь-який текстовий редактор або інтегроване середовище розробки (IDE) для написання та виконання коду. Наступним етапом є нлаштування TensorFlow.

Щоб налаштувати TensorFlow потрібно виконати наступні кроки:

Встановлення TensorFlow - відкрити термінал або командний рядок і виконати наступну команду, щоб встановити TensorFlow за допомогою pip, менеджера пакетів Python:

```
pip install tensorflow
```

Перевірка встановлення - після завершення встановлення можна перевірити його, імпортувавши TensorFlow у скрипт Python або в інтерактивний сеанс Python:

```
import tensorflow as tf
print(tf.version)
```

Це виведе версію TensorFlow, встановлену у системі.

Після встановлення потібного середовища програмування та налаштування TensorFlow потрібно обрати підхід для вирішення задачі класифікації зображеня. При вирішенні задач класифікації зображень дуже часто використовують архітектурний підхід CNN [10]. CNN розшифровується як згорткова штучна нейронна мережа - тип моделі глибокого навчання, спеціально розроблена для обробки та аналізу візуальних даних, таких як зображення. Вона широко використовується в задачах комп'ютерного зору,

включаючи класифікацію зображень, виявлення об'єктів і сегментацію зображень.

На відміну від традиційних штучних нейронних мереж, CNN характеризуються унікальним архітектурним дизайном, який включає згорткові шари, об'єднані шари та повністю з'єднані шари. Згорткові шари застосовують фільтри до вхідного зображення, вловлюючи локальні шаблони та особливості. Шари об'єднання зменшують просторові розміри карт об'єктів, зменшуючи обчислювальну складність і забезпечуючи інваріантність перекладу. Повністю з'єднані шари відповідають за остаточні прогнози на основі вивчених ознак.

CNN навчаються на великому наборі даних із маркованих зображень, де ваги мережі коригуються ітеративно за допомогою процесу, який називається зворотним поширенням. Під час навчання мережа вчиться автоматично виокремлювати з вхідних зображень релевантні ознаки, такі як краї, текстури та форми, які є важливими для розрізнення різних класів або об'єктів.

CNN зробили революцію в галузі комп'ютерного зору і допомогли досягти найсучасніших результатів у таких завданнях, як класифікація зображень, виявлення об'єктів і сегментація зображень. Вони успішно застосовуються в різних галузях, включаючи охорону здоров'я, автономні транспортні засоби, робототехніку та багато інших.

У згортковій штучній нейронній мережі зазвичай є три основні типи шарів (рис. 3.1): згорткові шари, об'єднані шари та повністю з'єднані шари. Ці шари працюють разом, щоб обробити і виділити ознаки з вхідних зображень.

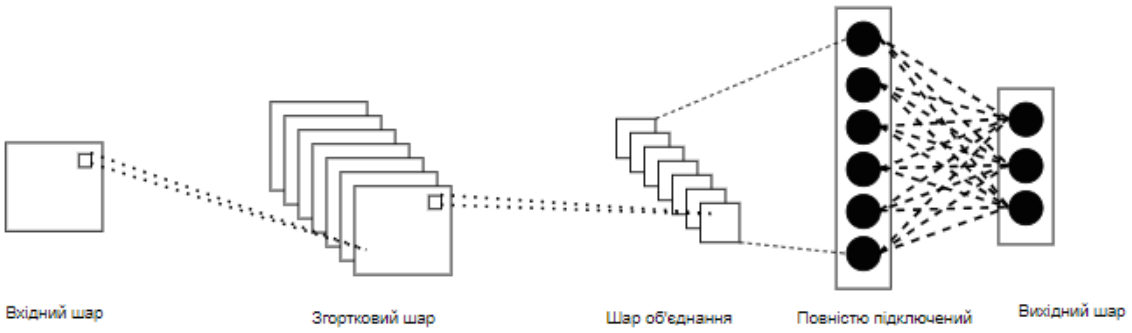

Рисунок 3.1 – три типи шарів у згортковій нейронній мережі

Згорткові шари - це основні будівельні блоки CNN. Вони застосовують до вхідного зображення набір фільтрів, що навчаються (також відомих як ядра або детектори ознак). Кожен фільтр виконує операцію згортки, ковзаючи по вхідному зображенню і обчислюючи добутки точок між вагами фільтра і локальним сприйнятливим полем вхідного зображення. Цей процес допомагає вловити локальні патерни та особливості, такі як краї, текстури або форми. Згорткові шари дозволяють мережі вивчати ієрархічні представлення вхідних даних.

Об'єднувальні шари зазвичай вставляються після згорточних шарів для зменшення просторових розмірів (ширини і висоти) карт об'єктів. Найпоширенішим типом об'єднання, що використовується в CNN, є максимальне об'єднання, коли вхідні дані розбиваються на області, що не перекриваються, і зберігається максимальне значення в кожній області. Така операція зменшення вибірки зменшує обчислювальну складність і кількість параметрів у мережі, а також забезпечує певний ступінь інваріантності перекладу. Об'єднання шарів допомагає виокремити найбільш релевантні та характерні ознаки, відкидаючи менш важливу інформацію.

Повністю з'єднані шари: Після кількох шарів згортки та об'єднання результат часто «згладжується» і додається до одного або кількох повністю з'єднаних шарів. Ці шари схожі на шари в традиційних штучних нейронних мережах. Кожен штучний нейрон у повністю зв'язаному шарі з'єднаний з

кожним штучним нейроном у попередньому шарі, що дозволяє мережі вивчати складні взаємозв'язки між характеристичними ознаками. Останній повністю пов'язаний шар зазвичай відповідає за подальші дії, наприклад, за класифікацію зображень за різними категоріями.

Розташування і кількість цих шарів може змінюватися залежно від конкретної архітектури ШНМ і поставленого завдання [10]. Глибокі архітектури ШНМ, такі як VGGNet, ResNet або InceptionNet, поєднують декілька згорткових та об'єднувальних шарів для вивчення все більш складних зображень. Повністю з'єднані шари в кінці мережі виконують остаточну класифікацію або регресію на основі вивчених ознак [11].

Чому такі шари використовуються у штучних нейроних мережах? У нейронних мережах шари використовуються для структурування та організації обчислень, що виконуються над вхідними даними. Кожен шар у нейронній мережі виконує певну операцію і відіграє вирішальну роль у навчанні та представленні складних закономірностей і взаємозв'язків у даних. Наведемо декільк причин, чому шари використовуються в штучних нейроних мережах:

Ієрархічне представлення – штучні нейронні мережі призначені для вивчення ієрархічних представлень вхідних даних. Шари дозволяють мережі охоплювати різні рівні абстракції, поступово комбінуючи і трансформуючи вхідну інформацію. Нижчі шари фіксують низькорівневі ознаки, такі як краї та текстури, тоді як вищі шари фіксують більш складні та абстрактні ознаки [11].

Виділення об'єктів - такі шари, як шари згортки, особливо ефективні для вилучення релевантних ознак з вхідних даних. Ці шари виконують локальні операції з використанням фільтрів, що навчаються, дозволяючи мережі виявляти закономірності та особливості в різних просторових точках. Виділені ознаки стають все більш абстрактними та репрезентативними, коли вони проходять через шари.

Нелінійність - функції активації, що застосовуються в шарах, вносять нелінійність у мережу. Нелінійність має вирішальне значення для штучних нейронних мереж, щоб навчатися і представляти складні взаємозв'язки в даних. Без нелінійних функцій активації мережа могла б навчатися лише лінійним перетворенням, що суттєво обмежувало б її виразність.

Навчання параметрів - кожен шар штучної нейронної мережі має власний набір параметрів, які вивчаються в процесі навчання. Ці параметри визначають ваги та зсуви, які мережа застосовує до вхідних даних. Регулюючи параметри за допомогою алгоритмів оптимізації, таких як градієнтний спуск, мережа може навчитися робити точні прогнози та класифікації на основі заданих вхідних даних.

Узагальнення та стійкість - використання шарів, таких як шари відсіву, допомагає покращити узагальнення та робастність моделі. Шари відсіву випадковим чином встановлюють частину вхідних одиниць на нуль під час навчання, що зменшує надмірну залежність від специфічних особливостей і запобігає надмірному пристосуванню. Цей метод регуляризації заохочує мережу вивчати більш узагальнені уявлення, які можуть краще обробляти невидимі або зашумлені дані.

Складність та потужність моделі - шари дозволяють будувати глибокі штучні нейронні мережі з декількома рівнями абстракції. Глибинні мережі мають потенціал для вивчення більш складних і заплутаних взаємозв'язків у даних. Шляхом накладання шарів збільшується потужність моделі, що дозволяє їй вловлювати та представляти дуже складні закономірності та особливості у вхідних даних.

Таким чином, шари в штучних нейронних мережах забезпечують ієрархічне представлення, виокремлення ознак, нелінійність, вивчення параметрів, узагальнення та збільшення потужності моделі. Вони забезпечують основу для мережі для перетворення та обробки вхідних даних, дозволяючи їй навчатися і робити прогнози на основі основних закономірностей і взаємозв'язків.

У даній нейронній мережі використовується кілька типів шарів (Рис. 3.2):

Згорткові шари (Conv2D):

Згорткові шари відповідають за вилучення ознак з вхідних даних.

Шар Conv2D виконує операції згортки над вхідними даними.

Кожна операція згортки передбачає застосування набору фільтрів, що навчаються, до вхідних даних, в результаті чого генеруються карти ознак.

Фільтри ковзають над вхідними даними, виконуючи поелементне множення та підсумовування, щоб захопити локальні шаблони та особливості.

Функції активації, такі як "relu", застосовуються для введення нелінійності та покращення здатності моделі вивчати складні взаємозв'язки в даних.

Кількість і розмір фільтрів можна задавати, що визначає глибину і складність вивчених особливостей.

Максимальне об'єднання шарів (MaxPooling2D):

Шари максимального об'єднання зменшують просторові розміри вхідних даних, в результаті чого створюються карти ознак з меншою вибіркою.

Шар MaxPooling2D оперує з картами об'єктів, створеними шарами згортки.

Він ділить кожну карту об'єктів на області, що не перекриваються (вікна об'єднання), і зберігає лише максимальне значення в межах кожного вікна.

Цей процес зменшення вибірки допомагає зменшити обчислювальну складність і забезпечує інваріантність перекладу, зосереджуючи увагу на найбільш помітних ознаках.

Шари пакетної нормалізації (BatchNormalization):

Шари пакетної нормалізації нормалізують активації попереднього шару.

Вони коригують і масштабують активації до нульового середнього значення та одиничної дисперсії, що допомагає стабілізувати та прискорити процес навчання.

Нормалізація допомагає зменшити внутрішній зсув коваріацій, роблячи модель більш стійкою та зменшуючи потребу в ретельній ініціалізації ваг.

Нормалізуючи активації, пакетна нормалізація дозволяє кожному шару мережі навчатися незалежно і покращувати загальну продуктивність.

Вирівнювання шару (Flatten):

Шар Flatten перетворює вихідні дані попереднього шару в одновимірний вектор.

Він перетворює багатовимірні карти об'єктів у лінійний масив, готовий до подачі в повністю з'єднані шари [11].

Щільні шари (Dense):

Щільні шари, також відомі як повністю зв'язані шари, відповідають за завдання класифікації або регресії.

Щільний шар з'єднує кожен штучний нейрон з попереднього шару з кожним штучним нейроном поточного шару, утворюючи повністю зв'язну мережу.

Кожен штучний нейрон у щільному шарі обчислює зважену суму своїх входів, застосовує функцію активації і генерує вихід.

Функції активації, такі як "relu", вносять нелінійність, дозволяючи моделі вивчати складні взаємозв'язки.

Кількість штучних нейронів у щільних шарах може бути задана, що визначає складність і потужність моделі.

Шари, що відсіваються (Dropout):

Шари, що відсіваються, використовуються для зменшення перенавчання моделі.

Шар відсіву випадковим чином встановлює частину вхідних одиниць в 0 під час навчання.

Цей процес відсіювання допомагає запобігти надмірній залежності мережі від специфічних особливостей або спільної адаптації до шуму в навчальних даних.

Видаляючи частину штучних нейронів під час навчання, шари відсіву заохочують мережу вивчати більш надійні та узагальнені репрезентації [10].

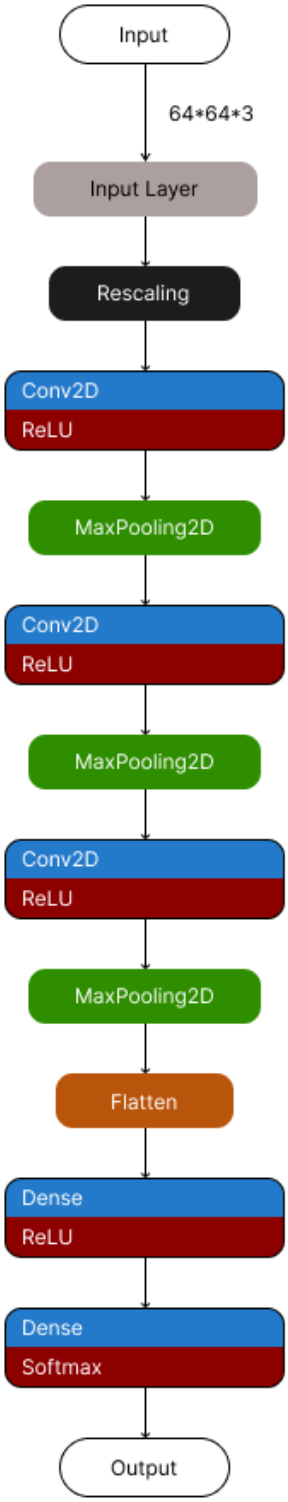

Рисунок 3.2 – архітектура нейронної мережі для генерації планувань

Загальна архітектура штучної нейронної мережі призначена для вилучення ієрархічних ознак з вхідних даних за допомогою згорткових шарів, зменшення вибірки карт ознак за допомогою максимального об'єднання, нормалізації активацій за допомогою пакетної нормалізації, перетворення ознак у лінійний масив за допомогою плаского шару і, нарешті, класифікації ознак за допомогою щільних шарів. Функції активації, шари відсіву та методи нормалізації допомагають покращити продуктивність, узагальнення та робастність моделі [12].

3.2 Розробка програми

Для побудови та тренування моделі використовується спеціальна бібліотека TensorFlow. TensorFlow - це фреймворк машинного навчання з відкритим вихідним кодом, розроблений компанією Google. Він надає комплексну екосистему інструментів, бібліотек та ресурсів для побудови та розгортання моделей машинного навчання. TensorFlow широко використовується для різних завдань, включаючи штучні нейронні мережі, глибоке навчання, обробку природної мови та комп'ютерний зір [13].

Передбачається шість етапів навчання за допомогою TensorFlow:

1 етап. Встановлення

2 етап. Підготовка даних

3 етап. Визначення моделі

4 етап. Навчання моделі

5 етап. Оцінка й налаштування моделі

6 етап. Розгортання та висновки

Після того, як модель навчена і оцінена, її можна розгорнути для виводу на нових, раніше не бачених даних.

TensorFlow надає варіанти розгортання моделі, включаючи обслуговування моделі за допомогою TensorFlow Serving або перетворення її у формат, сумісний з TensorFlow Lite для мобільних і вбудованих пристроїв.

TensorFlow пропонує широку документацію, навчальні посібники та спільноту підтримки, що робить його доступним для початківців і гнучким для досвідчених користувачів [13]. Це потужна платформа для навчання та розгортання моделей інтелектуальних застосунків в широкому діапазоні областей і додатків.

Скрипт будемо писати використовуючи PowerShell. PowerShell - це потужна мова сценаріїв і середовище автоматизації, розроблена компанією Microsoft. Вона призначена для автоматизації адміністративних завдань і керування системами, що робить її цінним інструментом для системних адміністраторів, IT-фахівців і розробників. Ось кілька причин, чому PowerShell вважається кращим у використанні:

Глибока інтеграція - PowerShell тісно інтегрований з операційною системою Windows і різними технологіями Microsoft. Вона надає широкий доступ до Windows Management Instrumentation (WMI) і Component Object Model (COM), що дозволяє автоматизувати адміністративні завдання, пов'язані з конфігурацією, управлінням і моніторингом системи. PowerShell також підтримує пряму інтеграцію з іншими продуктами Microsoft, такими як Active Directory, Exchange Server і SharePoint.

Об'єктно-орієнтований підхід - PowerShell використовує об'єктно-орієнтовану модель сценаріїв, де все представлено як об'єкт з властивостями і методами. Ця об'єктно-орієнтована природа спрощує маніпулювання та автоматизацію системних ресурсів. Команди PowerShell, відомі як команди, можуть бути об'єднані в конвеєр, що дозволяє використовувати вихідні дані однієї команди як вхідні для іншої, забезпечуючи потужне і стисле написання сценаріїв.

Розширюваність - PowerShell є дуже розширюваним, що дозволяє вам включати додаткові функціональні можливості за допомогою модулів, сценаріїв і користувацьких функцій. Ви можете використовувати існуючі модулі з галереї PowerShell або створювати власні модулі для інкапсуляції багаторазового коду. Крім того, PowerShell може інтегруватися з іншими

мовами програмування, такими як C# і .NET, що дозволяє використовувати існуючі бібліотеки та фреймворки.

Автоматизація та управління завданнями - PowerShell відмінно справляється з автоматизацією повторюваних завдань, дозволяючи вам економити час і зусилля. Вона надає багатий набір вбудованих команд для маніпуляцій з файлами, редагування реєстру, управління процесами тощо. Можна писати сценарії для виконання складних завдань, планувати їх запуск у певний час або виконувати їх як частину більшого робочого процесу. Здатність PowerShell автоматизувати завдання в різних системах робить його цінним інструментом для управління та обслуговування масштабних середовищ.

Перенесення сценаріїв - Сценарії PowerShell можна легко поширювати і виконувати на різних системах, якщо PowerShell встановлено. Ця переносимість дозволяє запускати сценарії на декількох машинах, що робить його зручним для розгортання і управління конфігурацією в мережі систем.

Активна спільнота і підтримка - PowerShell має велику і активну спільноту користувачів і розробників. Вона виграє від постійного розвитку, частих оновлень і безлічі ресурсів, створених спільнотою, включаючи форуми, блоги і навчальні посібники. Спільнота надає підтримку, ділиться знаннями і пропонує рекомендації, що полегшує навчання і вирішення проблем при роботі з PowerShell.

Після визначення способу побудови та навчання моделі з інтелектуальними функціями, почнемо написання коду. У функції main (рис. 3.3) задаємо шлях до csv файлу, а також до директорії, у якій знаходиться датасет для тренування штучного інтелекту.

```
def main():
    train_data, validation_data = csv_dataset(
        csv_file="./dataset/data.csv",
        image_directory="./dataset/floor_plan_images",
        batch_size=50,
        image_column_name="Sample",
        class_column_names=["class A"],  # Set the category to fit to
        validation_split=0.2,
    )
```

Рисунок 3.3 – функція main

На даному етапі датасет, на якому буде відбуватися тренування штучного інтелекту виглядає (рис. 3.4) так:

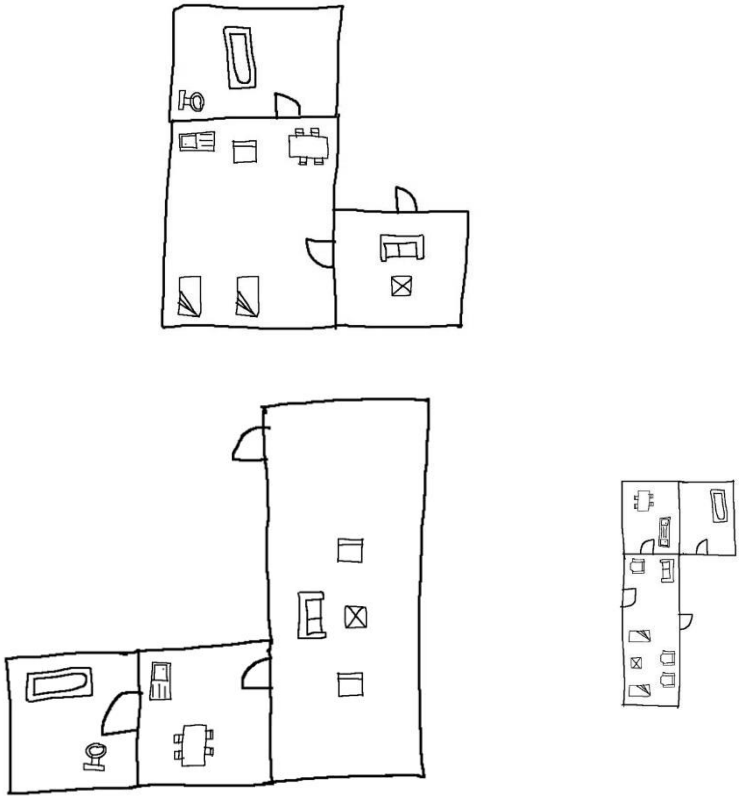

Рисунок 3.4 – приклад ескізів планування квартир

В датасеті є 510 зображень для початку тренування штучної нейромережі. З яких, 170 зображень – двокімнатні квартири, 170 зображень – трьохкімнатні квартири та ще 170 зображень – це чьотирьохкімнатні квартири.

Кожне зображення класифікується до одного з попередньо визначених класів. Класифікація - це процес присвоєння мітки або категорії вхідному зображенню на основі його особливостей або змісту.

Завдання класифікації полягає в тому, щоб навчити модель, яка може точно передбачити клас або категорію даного зображення. Модель навчається за допомогою маркованого набору даних, де кожне зображення асоціюється з певним класом або категорією. У процесі навчання модель вчиться розпізнавати шаблони та ознаки на вхідних зображеннях, які вказують на відповідні класи.

Результати класифікації зазвичай представляються у вигляді матриці, відомої як матриця помилок або ж матриця плутанини (confusion) (рис. 3.5). Матриця помилок надає узагальнену інформацію про роботу моделі класифікації, показуючи кількість правильних і неправильних прогнозів для кожного класу. Вона дозволяє оцінити, наскільки добре працює модель з точки зору класифікації зображень у відповідні категорії.

```
Sample,Cat1_1,Cat1_2,Cat1_3,Cat1_4,Cat1_5,Cat1_6,Cat1_7,Cat1_8,Cat1_9,Cat1_10,Cat2_1,Cat2_2,Cat2_
Cat1_1_wf.jpg, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_2_wf.jpg, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_3_wf.jpg, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_4_wf.jpg, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_5_wf.jpg, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_6_wf.jpg, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_7_wf.jpg, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_8_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_9_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat1_10_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
Cat2_1_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_2_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_3_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_4_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_5_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_6_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_7_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_8_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_9_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat2_10_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
Cat3_1_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat3_2_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cat3_3_wf.jpg, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0
```

Рисунок 3.5 – матриця плутанини

Матриця плутанини організована в табличному форматі, де рядки представляють фактичні або істинні класи зображень, а стовпці - класи, передбачені моделлю. Кожна клітинка матриці вказує на кількість або частоту зображень, які належать до певного істинного класу і були передбачені як

певний передбачуваний клас. Діагональні клітинки матриці представляють правильні прогнози, тоді як клітинки поза діагоналлю представляють неправильні прогнози.

Аналіз матриці помилок може дати уявлення про сильні та слабкі сторони моделі класифікації. Він допомагає визначити, які класи класифікуються точно, а які класи можуть мати вищі показники плутанини або помилкової класифікації. Ця інформація може бути використана для подальшого покращення продуктивності моделі шляхом вирішення конкретних проблем або упереджень у завданні класифікації.

Загалом, в процесі класифікації кожному вхідному зображенню присвоюється мітка або категорія, а результат класифікації відображається в матриці плутанини, яка надає оцінку ефективності моделі з точки зору класифікації зображень у відповідні категорії.

Через те що була незадовільна точність віндносно поставленої задачі при навчанні штучної нейроної мережі, необхідно було розширити початкову вибірку використовуючи аугментацію [14]. Аугментація даних - це метод, який використовується в машинному навчанні та комп'ютерному зорі для штучного збільшення різноманітності та розміру навчального набору даних шляхом застосування різних перетворень або модифікацій до наявних даних. Це допомагає поліпшити продуктивність і здатність до узагальнення моделей машинного навчання, вносячи варіації у вхідні дані [14].

Мета доповнення даних - створити додаткові навчальні вибірки, які схожі на вихідні дані, але мають невеликі відмінності, імітуючи реальні сценарії і підвищуючи здатність моделі обробляти різні ситуації. Піддаючи модель ширшому діапазону варіацій, вона стає більш надійною, інваріантною до певних перетворень і менш схильною до надмірного пристосування.

Деякі поширені методи доповнення даних включають:

- Перевертання зображення по горизонталі або вертикалі. Часто використовується в таких завданнях, як виявлення або класифікація об'єктів, де орієнтація об'єкта не впливає на ціль.

- Обертання - поворот зображення на певний кут. Це допомагає моделі вивчити інваріантність обертання і робить її більш стійкою до змін орієнтації об'єкта.

- Масштабування та обрізання - зміна розміру або обрізання зображення до різних розмірів. Це вводить варіації розміру об'єкта і допомагає моделі обробляти об'єкти в різних масштабах.

- Переміщення - переміщення зображення по горизонталі або вертикалі. Допомагає моделі навчитися бути інваріантною до невеликих переміщень.

- Введення шуму - додавання випадкового шуму до зображення. Це допомагає моделі стати більш стійкою до шуму в реальних даних.

- Зсув - Застосування зсувних перетворень до зображення, які нахиляють об'єкти на зображенні. Це може допомогти моделі обробляти об'єкти під різними кутами.

- Варіації кольору та контрасту: Регулювання колірного балансу, яскравості, контрастності або насиченості зображення. Це допомагає моделі навчитися справлятися з різними умовами освітлення.

Застосовуючи ці методи доповнення, доповнений набір даних стає більш різноманітним, надаючи моделі ширший спектр прикладів для навчання. Це зменшує ризик надмірної підгонки і допомагає моделі краще узагальнювати невидимі дані під час тестування і розгортання.

Розширення даних особливо корисне, коли навчальний набір даних обмежений, оскільки воно ефективно збільшує кількість доступних даних без необхідності додаткового ручного маркування або збору даних. Це стало

стандартною практикою в задачах глибокого навчання і комп'ютерного зору, сприяючи підвищенню продуктивності і точності моделі.

3.3 Процес тренування моделі

Навчання моделі машинного навчання починається з підготовки даних і визначення архітектури моделі.

Підготовка даних: Вхідні дані завантажуються, часто з файлу CSV або бази даних, і попередньо обробляються, щоб переконатися, що вони мають відповідний формат для навчання. Це може включати зміну розміру зображень, нормалізацію значень пікселів, поділ даних на навчальні та перевірочні набори, а також застосування методів доповнення даних для збільшення різноманітності навчальних даних.

Архітектура моделі: Визначається архітектура моделі, де вказуються шари та зв'язки між ними. Це визначає структуру моделі і те, як вона буде навчатися на вхідних даних. У наданому коді модель визначена за допомогою Keras Sequential API, який дозволяє накладати шари один на одного.

Компіляція: Після того, як архітектура моделі визначена, її потрібно скомпілювати. Це передбачає визначення функції втрат, оптимізатора та метрик оцінки, які будуть використовуватися під час навчання. Функція втрат вимірює, наскільки добре працює модель, оптимізатор налаштовує параметри моделі для мінімізації втрат, а метрики оцінювання надають додаткові показники ефективності.

Навчання: Процес навчання починається з виклику методу fit() на моделі. Він подає навчальні дані в модель партіями та ітеративно підлаштовує параметри моделі для мінімізації втрат. Процес навчання включає пряме поширення, коли вхідні дані проходять через шари моделі для генерації прогнозів, і зворотне поширення, коли обчислюються градієнти функції втрат відносно параметрів моделі і використовуються для оновлення параметрів за допомогою градієнтного спуску.

Оцінювання: Під час навчання продуктивність моделі оцінюється на валідаційному наборі даних. Це дає змогу оцінити, наскільки добре модель узагальнює невидимі дані, і допомагає відстежувати прогрес моделі. Метрики оцінки, визначені під час компіляції, використовуються для вимірювання продуктивності моделі.

Збереження моделі: Після того, як навчання завершено і модель досягла задовільної продуктивності, її можна зберегти на диску для подальшого використання або розгортання в додатках.

Перед початком тренування потрібно підібрати параметри, такі як швидкість навчання та кількість епох. В даному проекті початкова швидкість навчання дорівнює 0.01.

Загалом, швидкість навчання - це гіперпараметр, який визначає розмір кроку, з яким оновлюються параметри моделі в процесі навчання. Він контролює, наскільки швидко або повільно модель навчається на основі даних.

Вища швидкість навчання може призвести до швидшої збіжності під час навчання, оскільки модель робить більші оновлення своїх параметрів з кожною ітерацією. Однак використання дуже високої швидкості навчання може призвести до того, що модель вийде за межі оптимальних значень і не зможе збігтися.

З іншого боку, при меншій швидкості навчання параметри моделі змінюються рідше, що може призвести до повільнішої збіжності, але більш точного налаштування моделі. Однак використання дуже низької швидкості навчання може призвести до того, що процес навчання буде повільним або застрягне на неоптимальних рішеннях.

Встановлення відповідної швидкості навчання має вирішальне значення для успішного навчання моделі [15]. Для того, щоб знайти оптимальну швидкість навчання для конкретної задачі та архітектури моделі, часто потрібні експерименти та налаштування. Такі методи, як графіки швидкості навчання та адаптивні методи швидкості навчання, такі як оптимізатор Адама,

зазвичай використовуються для динамічного налаштування швидкості навчання під час навчання на основі прогресу моделі.

Після визначення швидкості навчання, визначимо кількість епох. В даному проекті параметр кількості епох має значення 92.

Загалом, епоха - це один повний прохід всього навчального набору даних через модель машинного навчання в процесі навчання. Під час кожної епохи модель ітерує навчальні дані, оновлює свої параметри на основі розрахованих втрат і прагне покращити свою продуктивність.

Використання епох у навчанні моделі є важливим, оскільки дозволяє моделі навчатися на даних за кілька ітерацій. Кожна епоха допомагає моделі вдосконалювати свої внутрішні параметри, вивчати закономірності та оптимізувати свою роботу. Зі збільшенням кількості епох модель отримує більше прикладів і має можливість вивчати більш складні репрезентації.

Однак при виборі кількості епох важливо знайти правильний баланс. Занадто мала кількість епох може призвести до недостатньої пристосованості, коли модель не зможе вловити всі відповідні закономірності в даних. З іншого боку, занадто велика кількість епох може призвести до надмірного припасування, коли модель стає занадто спеціалізованою на навчальних даних і погано працює на непередбачуваних даних.

Графік точності та епох (рис. 3.6) показує, як змінюється точність моделі в різні епохи в процесі навчання. Він відкладає точність по осі Y і кількість епох по осі X.

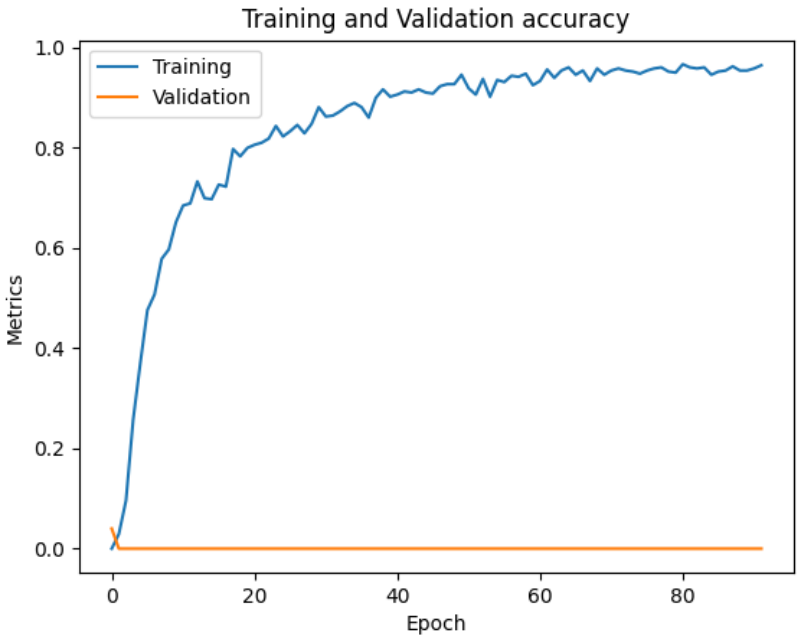

Рисунок 3.6 – графік точності епох

Як правило, очікується, що зі збільшенням кількості епох точність моделі покращується. Це пов'язано з тим, що модель має більше можливостей вчитися на даних і налаштовувати свої внутрішні параметри для кращої відповідності навчальним прикладам. З кожною епохою модель вдосконалює своє представлення даних і намагається мінімізувати помилки між своїми прогнозами та фактичними мітками.

Однак важливо зазначити, що існує точка спадання прибутковості. Після певної кількості епох збільшення кількості епох може призвести до надмірного припасування, коли модель стає занадто спеціалізованою для навчальних даних і погано працює на невидимих даних. Оптимальна кількість епох залежить від складності задачі, розміру набору даних та інших факторів. Часто її визначають шляхом експериментів та перевірки на окремому валідаційному наборі даних.

Таким чином, збільшення кількості епох може підвищити точність моделі до певного моменту, дозволяючи їй краще навчатися та узагальнювати навчальні дані. Однак дуже важливо знайти правильний баланс, щоб запобігти

надмірному пристосуванню і досягти оптимальної продуктивності на непередбачуваних даних.

Визначення відповідної кількості епох часто вимагає експериментів та моніторингу роботи моделі на перевірочному наборі даних. Такі методи, як рання зупинка, яка зупиняє навчання, якщо продуктивність моделі на валідаційному наборі не покращується, можуть допомогти визначити, коли зупинити навчання і уникнути надмірного пристосування.

Після завершення навчання моделі отримуємо такі результати:

- Параметри моделі: Параметри моделі, такі як ваги та зміщення, оновлюються в процесі навчання для оптимізації роботи моделі. Ці параметри відображають вивчені уявлення та взаємозв'язки в навчальних даних.

- Показники ефективності моделі: Продуктивність моделі як на навчальних, так і на перевірочних даних можна оцінити за допомогою різних метрик. Найпоширеніші метрики включають точність, втрати, точність, пригадування та оцінку F1, залежно від конкретного завдання та проблеми. Ці показники дають уявлення про те, наскільки добре працює модель, і можуть бути використані для порівняння різних моделей або для відстеження прогресу моделі в часі.

- Прогнози: Після навчання модель можна використовувати для прогнозування нових, ще не бачених даних. Це робиться шляхом пропускання вхідних даних через навчену модель і отримання відповідних вихідних прогнозів. Для задач класифікації модель може виводити ймовірності класів або мітки класів, тоді як для задач регресії вона може передбачати безперервні значення. Ці прогнози можна далі аналізувати або використовувати для прийняття рішень у додатках.

- Збереження моделі: Навчену модель можна зберегти на диску, щоб потім завантажити і використовувати її без перенавчання. Збереження моделі зберігає вивчені параметри та архітектуру, що робить її зручною для розгортання та висновків на нових даних.

Важливо зазначити, що конкретні результати, отримані після навчання, залежать від проблеми, даних та обраних метрик оцінювання. Метою є отримання добре працюючої моделі, яка добре узагальнює невидимі дані і досягає високої точності.

3.4 Створення додатку користувацького інтерфейсу

Для створення веб-додатку для перетворення ескізів у креслення використовуємо бібліотеку Streamlit [16]. Загальний огляд того, як він працює подано наступними діями:

1. Імпортування необхідних бібліотек: Імпортуються необхідні бібліотеки, зокрема TensorFlow, Streamlit, NumPy та PIL.
2. Завантаження навченої моделі: Функція load_model() завантажує попередньо навчену модель з файлу model.h5 за допомогою функції load_model TensorFlow.
3. Попередня обробка вхідного зображення: Функція preprocess_image() приймає зображення і виконує кроки попередньої обробки, такі як декодування зображення, зміна його розміру до певного розміру (64x64 пікселів) і застосування попередньої обробки, специфічної для моделі EfficientNet. Потім повертається попередньо оброблене зображення.
4. Завантаження зображення ескізу: Функція load_image() дозволяє користувачеві завантажити зображення ескізу. Завантажене зображення відображається за допомогою функції Streamlit image(), і повертаються дані зображення.

5. Генерування малюнку: Коли користувач натискає кнопку "Відобразити креслення", код продовжує генерувати креслення на основі завантаженого ескізу.
6. Класифікування ескізу: Попередньо оброблені дані зображення подаються в завантажену модель, і модель прогнозує ймовірності класів для кожної категорії. Клас з найвищою ймовірністю вважається прогнозованим класом.
7. Результат (отриманий малюнок): Прогнозована назва класу використовується для побудови шляху до файлу відповідного зображення креслення. Зображення креслення завантажується за допомогою функції Image.open() PIL.
8. Відображення креслення: Згенерований малюнок відображається за допомогою функції Streamlit image().
9. Завантаження креслення: Користувачеві надається кнопка завантаження «Завантажити креслення» для збереження згенерованого креслення на його комп'ютері.

Частини «Отримати малюнок» та «Показати малюнок» відповідають за отримання зображення малюнка на основі результату класифікації та відображення його користувачеві. Крім того, код надає можливість завантажити зображення креслення.

«Отримати малюнок»:

Після того, як користувач натискає кнопку «Відобразити малюнок» (рис. 3.7), код визначає передбачений клас з найбільшою ймовірністю.

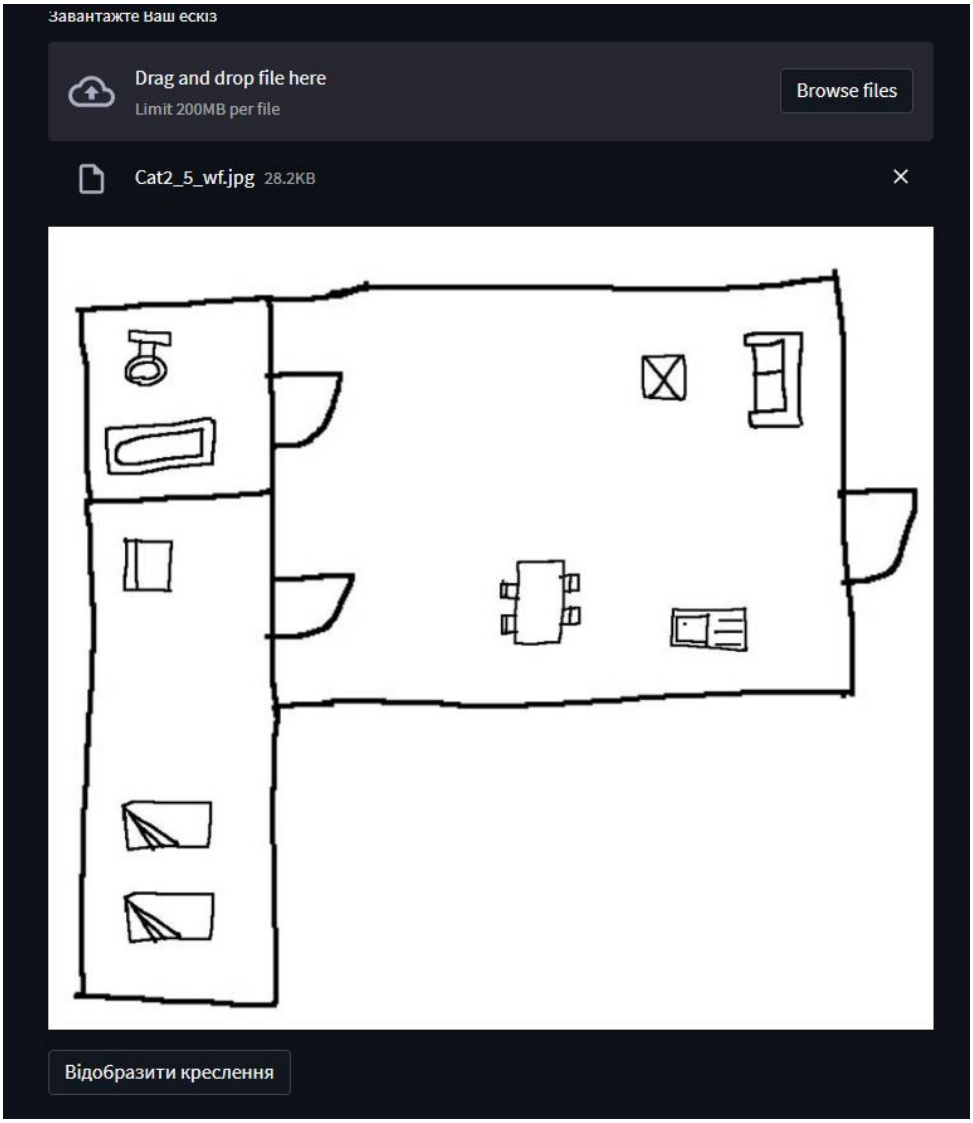

Рисунок 3.7 – кнопка "Відобразити малюнок" в додатку

Використовуючи передбачену назву класу, будується шлях до файлу відповідного зображення креслення. Код припускає, що зображення креслень зберігаються в каталозі з назвою "./drawings" з іменами файлів, що відповідають певній схемі іменування.

Зображення креслення завантажується за допомогою функції Image.open() з бібліотеки PIL.

Завантажене зображення малюнка зберігається у змінній guessed_img.

"Вивести малюнок на екран":

Після завантаження зображення креслення воно відображається користувачеві за допомогою функції st.image() з бібліотеки Streamlit.

Функція st.image() приймає на вхід guessed_img і автоматично відображає його на сторінці додатку Streamlit.

"Завантажте малюнок":

Код надає можливість завантажити відображене зображення малюнка.

Він використовує функцію st.download_button() з Streamlit, яка створює кнопку завантаження.

Функція st.download_button() отримує назву кнопки ("Завантажити креслення") і додаткові параметри, такі як дані (зображення креслення), ім'я файлу ("drawing.jpg") і тип MIME ("image/jpeg").

Коли користувач натискає кнопку завантаження (рис. 3.8), зображення креслення завантажується на його комп'ютер.

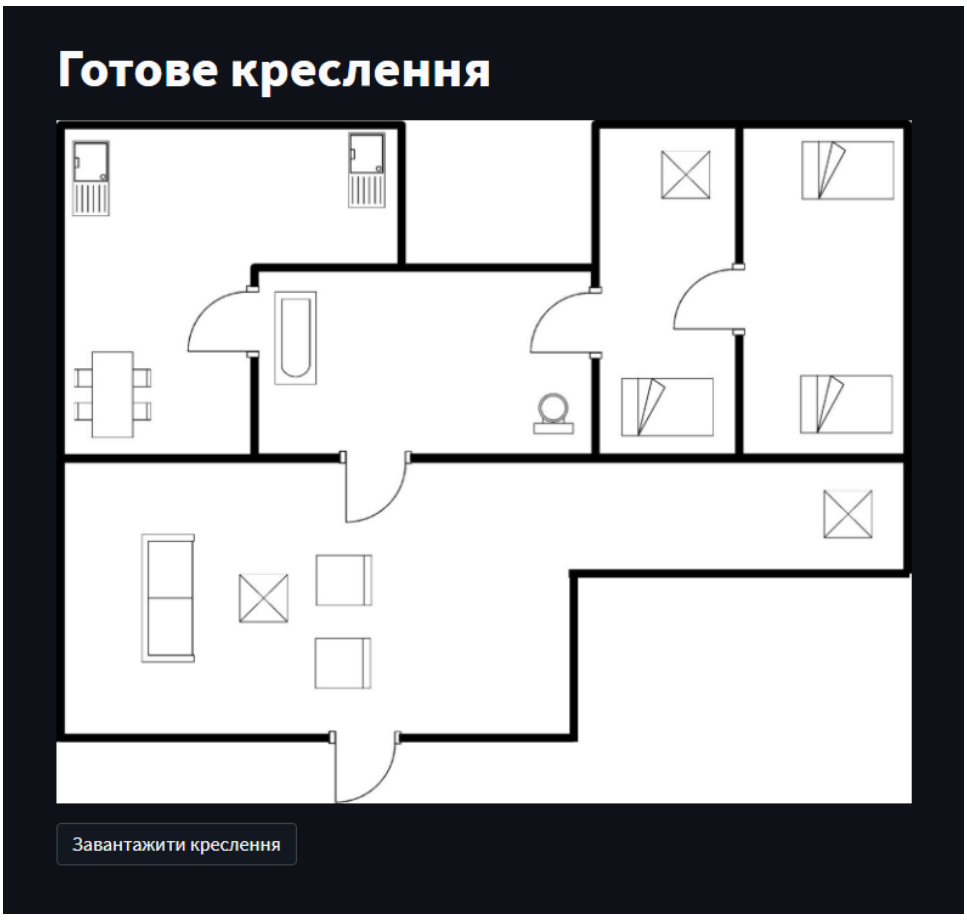

Рисунок 3.8 – кнопка завантаження зображення в додатку

В цілому, ці частини працюють разом, щоб отримати зображення малюнка на основі результату класифікації, відобразити його користувачеві та

надати можливість завантажити зображення малюнка. Функція Image.open() використовується для завантаження зображення малюнка, функція st.image() відображає зображення, а функція st.download_button() створює кнопку для завантаження зображення.

В результаті, отримуємо додаток що виглядає так (рис. 3.9, 3.10, 3.11):

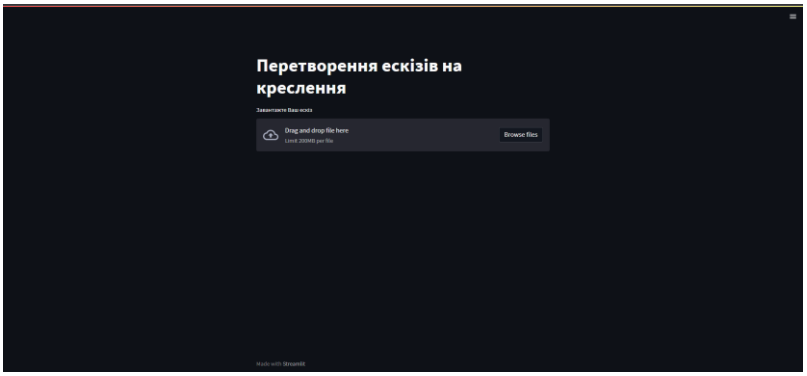

Рисунок 3.9 – відкриття застосунку

Рисунок 3.10 – завантаження ескізу в додаток

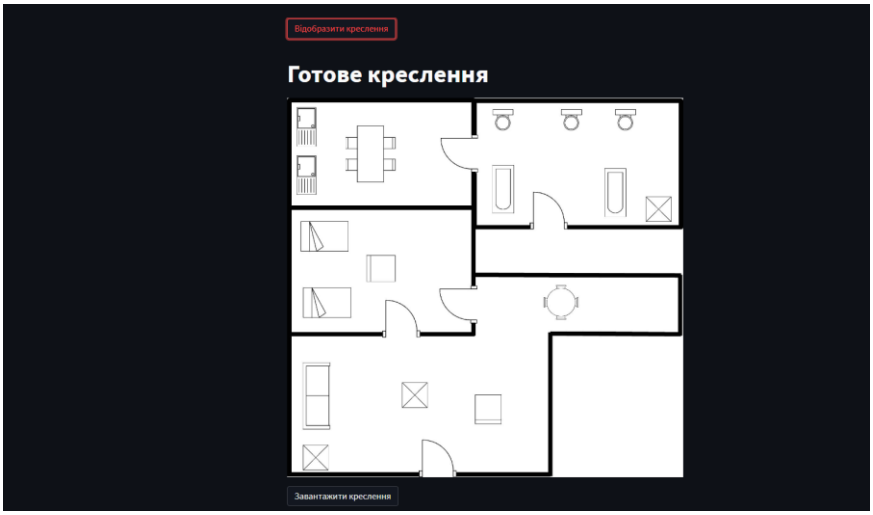

Рисунок 3.11 – Результат роботи додатку

Проведемо ще одне тестування, але вже з використанням схожих ескізів в різних форматах та з невеликою відміннюстю в плануванні. На рисунку 3.12 та 3.13 можна побачити різницю цих зображень.

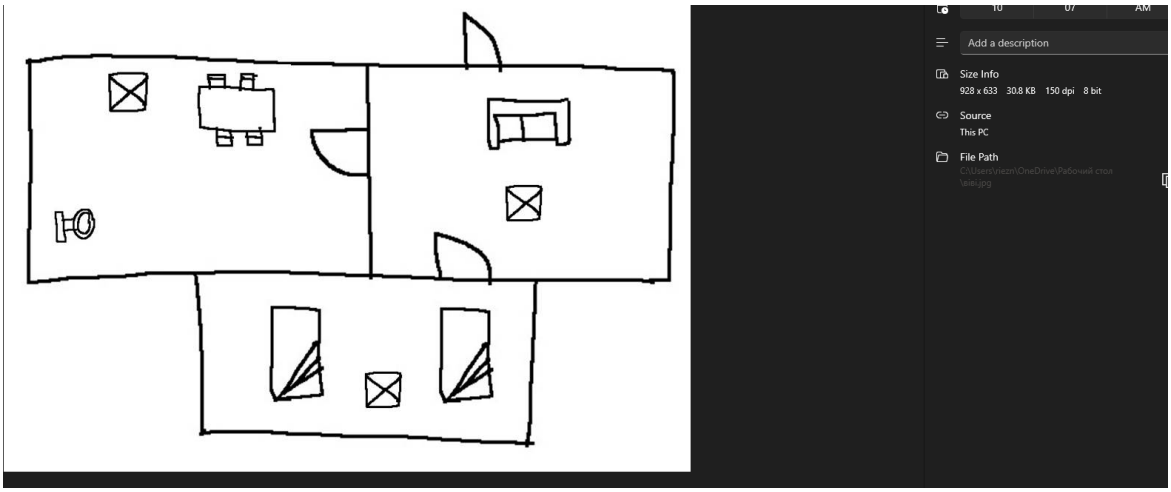

Рисунок 3.12 – перший ескіз в форматі 928x633

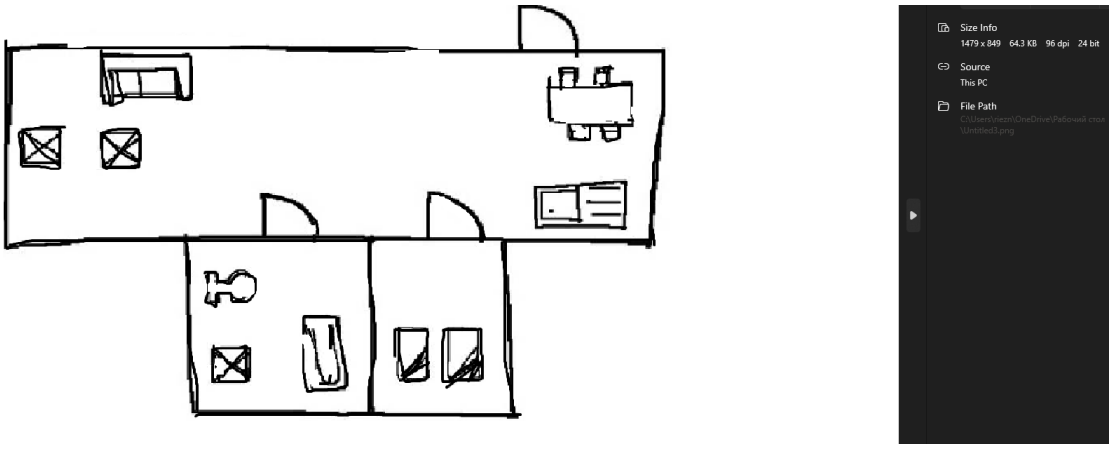

Рисунок 3.13 – другий ескіз в форматі 1479x849

Результат роботи програми у випадку завантаження першого та другого ескізу можна побачити відповідно на рисунку 3.14 та 3.15.

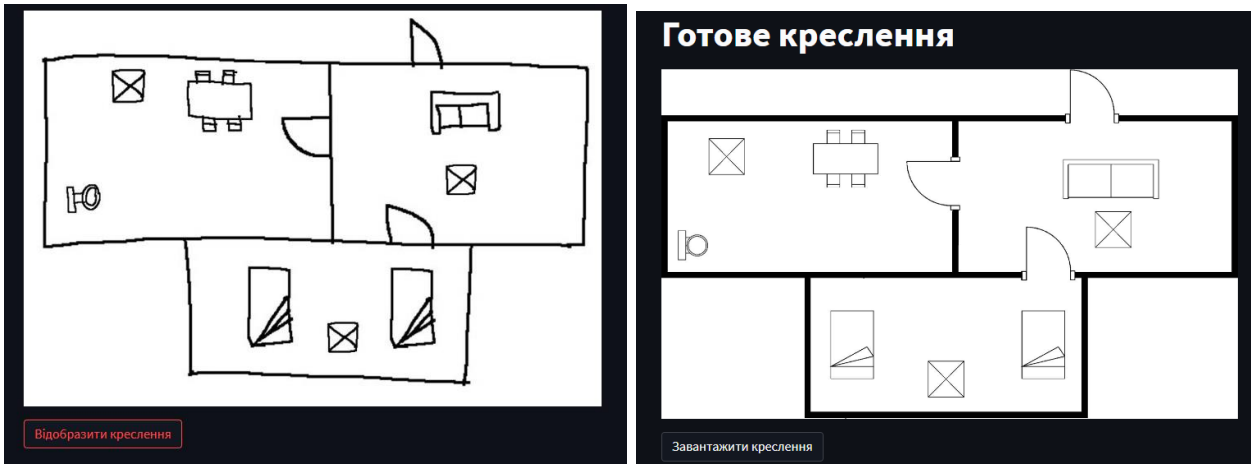

Рисунок 3.14 – результат роботи програми з першим ескізом

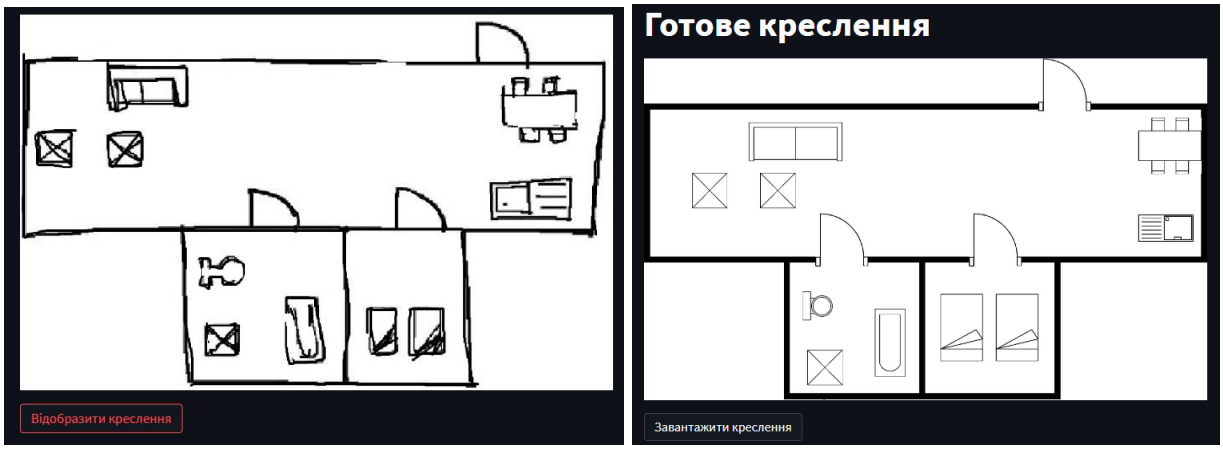

Рисунок 3.15 – результат роботи з другим ескізом в іншому форматі

Можна побачити, що програма генерує зображення, яке відповідає намальованому ескізу, навіть якщо розмір квартири та основна форма однакова, але наповнення, тобто кімнати, двері або розміщення мебелі різне.

3.5 Висновки до третього розділу

В результаті реалізації програмного забезпечення для побудови зображень на основі розпізнавання рукотворних графічних даних створено робочий додаток, який дозволяє використовувати його фахівцем у будівництві без попереднього знання спеціальних програмних САПР та виконання креслень планувань квартир.

Описано методи, що використовувалися для створення додатку. Також було описано процес тренування моделі штучної нейроної мережі та результати отримані після завершення тренування моделі.

ВИСНОВОК

Розроблена система успішно вирішує завдання аналізу графічних даних створених рукотворно проектувальником схем планування приміщень у будівництві для здійснення класифікації графічних ескізів на різні категорії проектів планів житлових приміщень, які у коректному вигляді зберігаються у базі даних. Завдяки використанню згорткових штучних нейронних мереж (CNN) модель демонструє здатність вивчати релевантні ознаки вхідних зображень і робити точні прогнози. Навчання ШНМ дозволяє реалізувати модель, яка здійснює віднесення вхідного графічного ескізу до відповідного класу, що дозволяє їй класифікувати ескізи з певним рівнем точності.

Процес навчання передбачає оптимізацію параметрів моделі за допомогою зворотного поширення та градієнтного спуску. Ітеративно змінюючи ваги та зміщення мережі, модель поступово покращує свою продуктивність. Вибір відповідних гіперпараметрів, таких як швидкість навчання та кількість епох, суттєво впливає на процес навчання та кінцеву точність моделі.

Набір даних, який використовується для навчання та оцінювання, відіграє вирішальну роль у продуктивності системи. Добре підібраний набір даних з різноманітними прикладами з кожного класу допомагає моделі узагальнювати і робити точні прогнози на невидимих ескізах. Методи доповнення даних, такі як перегортання, обертання та додавання шуму, можуть збільшити різноманітність набору даних та підвищити надійність моделі.

Оцінка продуктивності моделі зазвичай базується на таких показниках, як точність, достовірність, пригадування та оцінка F1. Ці показники дають уявлення про здатність моделі правильно класифікувати ескізи з різних класів і визначати потенційні області для вдосконалення.

Розроблена система також включає компонент користувацького інтерфейсу, який дозволяє користувачам завантажувати свої ескізи та отримувати миттєвий зворотній зв'язок щодо відповідних класифікованих

малюнків. Інтеграція веб-технологій, таких як Streamlit, забезпечує безперебійну та інтерактивну роботу користувачів.

На завершення, реалізована система класифікації ескізів демонструє потенціал машинного навчання та глибоких штучних нейронних мереж в автоматизації категоризації ескізів. Точність моделі, вибір відповідних гіперпараметрів та якість набору даних є вирішальними факторами для досягнення надійної та стійкої роботи. Користувацький інтерфейс системи підвищує її зручність і забезпечує практичний інструмент для різних застосувань, таких як художній дизайн, архітектурне планування та освітні цілі.

Крім того, розроблена система демонструє ефективність використання попередньо навчених моделей, таких як EfficientNet, як екстракторів ознак. Використовуючи вивчені репрезентації з великомасштабного набору даних, модель може вловлювати складні патерни та абстрактні ознаки з ескізів, що призводить до покращення ефективності класифікації.

Візуалізація результатів класифікації та відображення відповідних малюнків надає користувачам візуальне підтвердження точності системи. Такий візуальний зворотний зв'язок покращує роботу користувача і вселяє довіру до процесу класифікації.

Крім того, можливість завантажити класифіковані креслення у високоякісних форматах зображень дозволяє користувачам легко отримати доступ до згенерованих результатів і використовувати їх для подальшого аналізу, обміну або друку. Ця функціональність додає системі практичної цінності та підвищує її зручність у використанні.

Важливо відзначити, що хоча розроблена система демонструє багатообіцяючі результати, все ще існують потенційні області для майбутніх удосконалень. До них відносяться дослідження більш досконалих мережевих архітектур, більш широке налаштування гіперпараметрів та розширення набору даних, щоб охопити ширший діапазон варіацій та категорій ескізів. Крім того, система могла б виграти від включення механізмів зворотного

зв'язку з користувачем, щоб постійно покращувати точність класифікації та адаптуватися до конкретних уподобань користувачів.

Загалом, впроваджена система класифікації ескізів є цінним рішенням для автоматизації категоризації ескізів. Вона поєднує в собі найсучасніші методи машинного навчання, зручний дизайн інтерфейсу та ефективні можливості візуалізації, пропонуючи потужний інструмент для художників, дизайнерів, архітекторів та освітян.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Knight, T. John Wiley & Sons. "Artificial Intelligence in Architecture: Towards a New Materiality." - 2019.
2. Autodesk. "Dreamcatcher-Help-Web-DCHelp". Офіційна документація [Електронний ресурс]. – Режим доступу: https://help.autodesk.com/cloudhelp/ENU/Dreamcatcher-Help-Web-DCHelp/index.html .
3. Білл Хіллер. "Соціальна логіка простору." Cambridge University Press, 1996.
4. Гудфеллоу, І., Бенгіо, Ю. та Курвіль, А. (2016). "Глибоке навчання." MIT Press.
5. Каллан, Р. "Нейронні мережі: Короткий довідник.": Вільямс І.Д., 2017. – 288 с.
6. Симонян, К., & Зіссерман, А. (2014). "Дуже глибокі згорткові мережі для розпізнавання великомасштабних зображень." arXiv препринт arXiv:1409.1556.
7. Тан, М., та Ле, К. В. (2019). "EfficientNet: Переосмислення масштабування моделі для згорткових нейронних мереж." В Міжнародна конференція з машинного навчання (с. 6105-6114).
8. Функціональні та нефункціональні вимоги. “v/sure” [Електронний ресурс]. Доступно за посиланням: https://visuresolutions.com/uk/blog/non-functional-requirements/ .
9. Офіційна документація Python [Електронний ресурс]. Доступно за посиланням: https://www.python.org/ .
10. Андреас М. "Введення в машинне навчання за допомогою Python."
11. Шолле, Ф. (2017). "Xception: Глибоке навчання із згортками, що відокремлюються в глибину." Матеріали конференції IEEE з комп'ютерного зору та розпізнавання образів (с. 1251-1258).
12. Kingma, D. P., & Ba, J. (2014). "Adam: метод стохастичної оптимізації." arXiv препринт arXiv:1412.6980.

13. Абаді, М. та ін. (2016). "TensorFlow: система для великомасштабного машинного навчання." У 12-му симпозіумі USENIX з розробки та впровадження операційних систем (OSDI 16) (с. 265-283).
14. Іоффе, С., та С егеді, К. (2015). Пакетна нормалізація: Прискорення глибокого навчання мережі шляхом зменшення внутрішнього зсуву коваріації. Міжнародна конференція з машинного навчання (с. 448–456).
15. Гудфеллоу, І., Бенгіо, Ю. та Курвіль, А. (2016). "Глибоке навчання." MIT Press.
16. Офіційна документація Streamlit [Електронний ресурс]. Доступно за посиланням: https://streamlit.io/ .

ДОДАТОК А

Код додатку

App.py

```
# Імпортуємо потрібні для роботи бібліотеки
import io
import numpy as np
import streamlit as st
from PIL import Image
import tensorflow as tf
from tensorflow.keras.preprocessing import image

# функція завантаження моделі
def load_model():
    model = tf.keras.models.load_model('./model.h5')
    return model

# функція попередньої обробки зображення для подання його на вхід моделі
def preprocess_image(img):
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.resize(img, [64, 64])
    img = tf.keras.applications.efficientnet.preprocess_input(img)
    img = tf.expand_dims(img, axis=0)
    return img

# функція завантаження ескізу
def load_image():
    uploaded_file = st.file_uploader(label="Завантажте Ваш ескіз")
    if uploaded_file is not None:
        image_data = uploaded_file.getvalue()
        st.image(image_data)
        return Image.open(io.BytesIO(image_data)), image_data
    else:
        return None, None

# завантажуємо модель
model = load_model()

# Виведення заголовку на сторінку
st.title('Перетворення ескізів на креслення')

# завантажуємо ескіз
img, image_data = load_image()

# якщо ескіз було завантаженно
if img is not None:
    result = st.button('Відобразити креслення')

    if result:
        # генеруємо імена класів
        class_names = []
        for i in range (1, 51):
            for j in range(1, 11):
```

```
                class_name = f"Cat{i}_{j}"
                class_names.append(class_name)

        x = preprocess_image(image_data)

        preds = model(x)

        # Отримання імені класу з найбільшим відсотком вірогідності
        preds_np = preds.numpy()
        max_index = np.argmax(preds_np)

        # Отримання імені класу
        class_name = class_names[max_index]

        # Составление пути к файлу изображения
        file_path = f"./drawings/{class_name}.jpg"

        # Загрузка изображения
        guessed_img = Image.open(file_path)

        st.title('Готове креслення')

        # відображення зображення у Streamlit
        st.image(guessed_img)

        # Виведення імені класу для перевірки на коректність розпізнавання (для
тестування)
        # st.write(f"Распознанный класс: {class_name}")

        # Завантаження креслення на комп'ютер
        with io.BytesIO() as buffer:
            guessed_img.save(buffer, format='JPEG')
            buffer.seek(0)
            st.download_button(
                label='Завантажити креслення',
                data=buffer,
                file_name='drawing.jpg',
                mime='image/jpeg'
            )
```

Training.py

```
import pandas as pd
from keras_preprocessing.image import ImageDataGenerator
from typing import List

def csv_dataset(csv_file: str, image_directory: str, batch_size: int,
image_column_name: str,
                class_column_names: List[str], validation_split: float):

    dataframe = pd.read_csv(csv_file)

    train_datagen = ImageDataGenerator(
        rescale=None,
```

```
        validation_split=validation_split
    )

    augmentations = iaa.Sequential([
        iaa.Fliplr(0.5),
        iaa.Flipud(0.5),
        iaa.Rotate((-45, 45)),
        iaa.GaussianBlur(sigma=(0, 1.0)),
        iaa.AdditiveGaussianNoise(scale=(0, 0.05 * 255)),
        iaa.Multiply((0.8, 1.2)),
        iaa.ContrastNormalization((0.8, 1.2))
    ])

    train_generator = train_datagen.flow_from_dataframe(
        dataframe=dataframe,
        directory=image_directory,
        x_col=image_column_name,
        y_col=class_column_names,
        class_mode="raw",
        batch_size=batch_size,
        subset="training",
        shuffle=True,
        target_size=(64, 64),
        augmentations=augmentations
    )

    validation_generator = train_datagen.flow_from_dataframe(
        dataframe=dataframe,
        directory=image_directory,
        x_col=image_column_name,
        y_col=class_column_names,
        class_mode="raw",
        batch_size=batch_size,
        shuffle=True,
        subset="validation",
        target_size=(64, 64)
    )

    return train_generator, validation_generator

# start training / fit the model / save results

def main():
    batch_size = 1

    class_names = []
    for i in range(1, 52):
        for j in range(1, 11):
            class_name = f"Cat{i}_{j}"
            class_names.append(class_name)

    train_generator, validation_generator = csv_dataset(
        csv_file="./dataset/data.csv",
        image_directory="./dataset/floor_plan_images",
```

```
        batch_size=batch_size,
        image_column_name="Sample",
        class_column_names=class_names,
        validation_split=0.05,
    )

    # Set up training parameters
    initial_learning_rate = 0.01
    epochs = 92

    from models import binary_classifier
    from models import multi_class_classifier
    from training_utilities import training_callbacks,
model_performance_plotting, save_classifier

    # Set model architecture
    classifier = multi_class_classifier(input_dim=(64, 64, 3),
feature_detectors=32,
                                        size_feature_detectors=(4, 4),
learning_rate=initial_learning_rate)

    history = classifier.fit(x=train_generator,
                             steps_per_epoch=None,
                             epochs=epochs,
                             validation_data=validation_generator,
                             validation_steps=None,
                             callbacks=training_callbacks(ProgbarLogger=False,
TensorBoard=True, CSVLogger=True,
                                                          ModelCheckpoint=True,
LearningRateScheduler=True))

    model_performance_plotting(record=history)
    save_classifier(fitted_model=classifier, format="hdf5")

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

visualize_patterns.py

```
from keras.models import load_model
from keras.preprocessing import image
from typing import Tuple
import numpy as np

# Create a list to put in the label predictions
label_list_Numpy_2D = []

# path to test image at first!
test_image_path = 'predictions/test.png'

# load test img and convert to Numpy array
test_image = image.load_img(test_image_path,
                            target_size=(64, 64), color_mode="rgb")
```

```
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)


# Model to distinguish rectangular-shaped architecture
def classifier_rectangle():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish composite-rectangular shapes
def classifier_longitudinal():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish linear-shaped architecture
def classifier_linear():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish circular-shaped architecture
def classifier_circle():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish polygon-shaped architecture
def classifier_polygon():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)
```

```
    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish organically-shaped architecture
def classifier_organic():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish if the architecture has an Atrium or not
def classifier_atrium():
    classifier = load_model(
        'trained model as h5 file'
        , compile=True)

    result = classifier.predict(test_image)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish if the building has a column grid
def classifier_column_grid():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    test_image_Column = image.load_img(test_image_path,
                                       target_size=(128, 128, 1),
color_mode="grayscale")
    test_image_Column = image.img_to_array(test_image_Column)
    test_image_Column = np.expand_dims(test_image_Column, axis=0)
    result = classifier.predict(test_image_Column)
    label_list_Numpy_2D.append(result)
    return classifier


# Model to distinguish if the plan have staircases
def classifier_staircase():
    classifier = load_model(
        'trained model as h5 file',
        compile=True)

    test_image_Treppe = image.load_img(test_image_path,
                                       target_size=(64, 64, 1),
color_mode="grayscale")
    test_image_Treppe = image.img_to_array(test_image_Treppe)
    test_image_Treppe = np.expand_dims(test_image_Treppe, axis=0)
```

```
    result = classifier.predict(test_image_Treppe)
    label_list_Numpy_2D.append(result)
    return classifier


def Visualizer():
    # Call the predictions
    classifier_rectangle()
    classifier_longitudinal()
    classifier_linear()

    classifier_circle()
    classifier_polygon()
    classifier_organic()

    classifier_atrium()
    classifier_column_grid()
    classifier_staircase()

    # Convert a 2D Numpy Array to a Python list to parse it
    from numpy import ndarray
    label_list = np.array(label_list_Numpy_2D)
    list(label_list_Numpy_2D)
    ndarray.tolist(label_list)

    print("Printing a python list of estimated patterns...")
    print(label_list)

    # create the visualization
    def add_legend_to_image(image_path: test_image_path,
                            feature_vector: Tuple[int, int, int, int, int, int,
int, int, int]):

        from PIL import Image
        from PIL import ImageFont
        from PIL import ImageDraw

        # resize the test image and paste on new canvas for visualization
        img = Image.open(image_path)
        img = img.resize((1600, 1024), Image.ANTIALIAS)
        new_image = Image.new("RGB", (1500, 1400), color="white")
        new_image.paste(img, (0, 0, 1600, 1024))

        draw = ImageDraw.Draw(new_image)
        font = ImageFont.truetype(
            "your favourite typeface as .ttf or equivalent",
            19,
            encoding="unic")

        # List to process the predictions from the classifiers
        labels_to_add = []
        if feature_vector[0]:
            labels_to_add.append(("Rectangle", "black"))
        if feature_vector[1]:
```

```
            labels_to_add.append(("Composite", "pink"))
        if feature_vector[2]:
            labels_to_add.append(("Longitudinal", "blue"))
        if feature_vector[3]:
            labels_to_add.append(("Circle", "violet"))
        if feature_vector[4]:
            labels_to_add.append(("Polygonal", "gray"))
        if feature_vector[5]:
            labels_to_add.append(("Organic", "purple"))
        if feature_vector[6]:
            labels_to_add.append(("Atrium", "magenta"))
        if feature_vector[7]:
            labels_to_add.append(("Columns", "Orange"))
        if feature_vector[8]:
            labels_to_add.append(("Staircase", "green"))

        rectangle_x1 = 15
        text_x1 = 20

        if labels_to_add:
            draw.text(xy=(10, 610), text="Architectural Patterns", fill="black",
font=font)
            for label in labels_to_add:
                draw.rectangle(xy=(rectangle_x1, 640, rectangle_x1 + 110, 670),
fill=label[1])
                draw.text(xy=(text_x1, 645), text=label[0], font=font,
fill="white")
                rectangle_x1 += 130
                text_x1 += 130
        else:
            draw.text(xy=(10, 610), text="Did not find any pattern",
fill="black")

        new_image.show()
        new_image.save('predicted_1.png')

    # load the image again for display
    add_legend_to_image(image_path=test_image_path, feature_vector=label_list)


# Run Visualizer
if __name__ == '__main__':
    Visualizer()
```