Державний торговельно-економічний університет

Кафедра комп’ютерних наук та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний додаток для розпізнавання зображень на основі методів комп’ютерного зору»

Студента 2 курсу, 5м групи
спеціальності
122 «Комп’ютерні науки» _____ *підпис студента* Головейчук Олександр Головейчук

Науковий керівник
доктор фізико-математичних наук, професор _____ *підпис керівника* Пурський Олег Іванович

Гарант освітньої програми
доктор фізико-математичних наук, професор _____ *підпис керівника* Пурський Олег Іванович

Київ 2024

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«5» грудня 2023р.

Завдання
на кваліфікаційну роботу студенту

Головейчуку Олександру Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
«Програмний додаток для розпізнавання зображень на основі методів комп'ютерного зору»
Затверджена наказом ректора від *«27» листопада 2023 р. № 4144*

2. Строк здачі студентом закінченої роботи *4 листопада 2024 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: розробити програмний додаток, який застосовує методи комп'ютерного зору для точного та ефективного розпізнавання зображень.
Об'єкт дослідження: процеси розпізнавання зображень з ідентифікації стилю картин на основі методів комп'ютерного зору..
Предмет дослідження: моделі, алгоритми та методи комп'ютерного зору, що використовуються в програмних додатках для розпізнавання зображень.

4. Перелік графічного матеріалу_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Пурський О.І.	05.12.2023 р.	05.12.2023 р.
2	Пурський О.І.	05.12.2023 р.	05.12.2023 р.
3	Пурський О.І.	05.12.2023 р.	05.12.2023 р.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ МЕТОДАМИ КОМП'ЮТЕРНОГО ЗОРУ

1.1.Комп'ютерний зір у розпізнаванні зображень: історія, актуальність та перспективи використання

1.2. Основні методи комп'ютерного зору, що використовуються при розробці програмного забезпечення

1.3.Розпізнавання зображень в ідентифікації творів мистецтва

РОЗДІЛ 2. ОРГАНІЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ ТВОРІВ МИСТЕЦТВА

2.1. Огляд системи ідентифікації творів мистецтва

2.2. Підготовка датасету

2.3. Вибір нейронної мережі

2.4. Дизайн та архітектура програмного забезпечення

2.5. Вибір інструментів та технологій

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ ТВОРІВ МИСТЕЦТВА

3.1. Розробка скрипта для навчання моделі та їх тестування

3.1.1. Структура та функціональність скрипта

3.1.2. Тестування та оцінка моделі

3.2. Розробка серверного API

3.2.1. Тестування серверного API

3.3. Масштабування, розгортання та потенційні покращення системи

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми кваліфікаційної роботи*	*01.11.2023*	*01.11.2023*
2	*Розробка та затвердження завдання на кваліфікаційну роботу*	*05.12.2023*	*05.12.2023*
3	*Вступ*	*01.05.2024*	*01.05.2024*
4	*РОЗДІЛ 1. Теоретичні основи розпізнавання зображень методами комп'ютерного зору*	*14.06.2024*	*14.06.2024*
5	*Підготовка статті у збірник наукових статей магістрів*	*20.06.2024*	*20.06.2024*
6	*РОЗДІЛ 2. Організація розробки програмного забезпечення для ідентифікації творів мистецтва*	*05.09.2024*	*05.09.2024*
7	*РОЗДІЛ 3. Розробка програмного забезпечення для ідентифікації творів мистецтва*	*17.10.2024*	*17.10.2024*
8	*Висновки*	*21.10.2024*	*21.10.2024*
9	*Здача кваліфікаційної роботи на кафедру науковому керівнику*	*23.10.2024*	*23.10.2024*
10	*Попередній захист кваліфікаційної роботи*	*28.10.2024*	*28.10.2024*
11	*Виправлення зауважень, зовнішнє рецензування кваліфікаційної роботи*	*30.10.2024*	*30.10.2024*
12	*Представлення готової зшитої кваліфікаційної роботи на кафедру*	*04.11.2024*	*04.11.2024*
13	*Публічний захист кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «5» грудня 2023 р

9. Керівник випускного кваліфікаційної роботи Пурський О.І.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Пурський О.І.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Головейчук О.В
(прізвище, ініціали, підпис)

12. Відгук керівника кваліфікаційної роботи

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Керівник кваліфікаційної роботи

_____
(підпис, дата)

13. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента _____Головейчук О.В.__
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Пурський О.І.____
(підпис, прізвище, ініціали)

Завідувач кафедри _____Пурський О.І.____
(підпис, прізвище, ініціали)

«_____»_____2024 р.

Анотація

Кваліфікаційна робота присвячена розробці програмного додатку для ідентифікації картин на основі методів комп'ютерного зору. Робота передбачає створення та навчання моделі розпізнавання зображень спеціально для ідентифікації стилю написання картин. Було проведено комплексний аналіз сучасних методів комп'ютерного зору, обрано відповідну модель для розпізнавання творів мистецтва та навчено її з використанням набору даних картин. Крім того, було розроблено серверний API додаток для, який дозволяє користувачеві комунікувати з натреновано моделлю. Система забезпечує високу точність і зручність для користувача, використовуючи передові алгоритми розпізнавання зображень.

Ключові слова: розпізнавання картин, комп'ютерний зір, розпізнавання зображень, веб-додаток, мобільний додаток, машинне навчання.

Annotation

The qualification work is devoted to the development of a software application for identifying paintings based on computer vision methods. The work involves creating and training an image recognition model specifically for identifying the style of painting. A comprehensive analysis of modern computer vision methods was conducted, a suitable model for recognizing works of art was selected and trained using a dataset of paintings. In addition, a server-side API application was developed that allows the user to communicate with the trained model. The system provides high accuracy and user-friendliness by using advanced image recognition algorithms.

Keywords: picture recognition, computer vision, image recognition, web application, mobile application, machine learning.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасну цифрову епоху розвиток технологій штучного інтелекту та комп'ютерного зору відкрив нові можливості в різних галузях, зокрема й у мистецтві. Ідентифікація та розпізнавання картин, особливо у великих колекціях або на цифрових платформах, створює значні проблеми через різноманітність стилів, технік та історичних контекстів. Традиційні методи ідентифікації творів мистецтва вимагають значних знань і ручної праці, що може зайняти багато часу і коштувати дорого. Це призвело до зростання потреби в автоматизованих системах, здатних точно ідентифікувати та класифікувати картини. Методи комп'ютерного зору, зокрема алгоритми розпізнавання зображень, забезпечують ефективне та масштабоване вирішення цього завдання, автоматизуючи процес ідентифікації картин та покращуючи доступ до інформації про мистецтво як для експертів, так і для широкої громадськості.

Розробка програмного додатку для ідентифікації стилю картин дозволить користувачам ефективно розпізнавати художні стилі, надаючи цінну інформацію та покращуючи способи управління та оцінки мистецьких колекцій. Для забезпечення доступності та інтеграції з різними системами буде створено серверний API, що дозволяє легко взаємодіяти з моделлю розпізнавання через зовнішні клієнтські додатки.

Мета роботи: Розробити програмний додаток, що використовує методи комп'ютерного зору для ідентифікації стилю картин.

Для досягнення цієї мети необхідно виконати декілька завдань:

1. Провести комплексне дослідження технологій комп'ютерного зору для розпізнавання зображень.
2. Розробити та навчити модель для ідентифікації стилю картин на основі кураторського набору даних.
3. Розробити архітектуру серверного API для інтеграції з зовнішніми клієнтськими системами.
4. Реалізувати та протестувати робочу систему, що дозволяє користувачам ідентифікувати картини через обидві платформи.

Об’єкт дослідження: Процеси розпізнавання зображень з ідентифікації стилю картин на основі методів комп'ютерного зору.

Предмет дослідження: Моделі, алгоритми та методи, технології комп’ютерного зору що використовуються для ідентифікації стилю картин.

Методи дослідження: Дослідження спирається на загальнонауковий аналітичний метод та системний підхід, спираючись на ідеї провідних дослідників та сучасні дослідження в галузі комп'ютерного зору та розпізнавання зображень. Інформаційну базу дослідження склали різноманітні джерела, такі як книги, наукові статті та матеріали з мережі

Інтернет, включаючи документацію з технологій машинного навчання та фреймворків, що використовуються при розробці системи.

Для вирішення поставлених завдань і практичних проблем були застосовані наступні методи дослідження:

- Загальнонауковий аналітичний метод (Розділ 1);
- метод системного моделювання для проектування архітектури додатку (Розділ 2);
- методи функціонального та алгоритмічного програмування для реалізації моделі розпізнавання зображень та програмного додатку (розділ 3).

Наукова новизна одержаних результатів полягає у створенні програмного забезпечення для ідентифікації стилю картин, яке використовує сучасні методи комп'ютерного зору та алгоритми глибокого навчання. Розроблено серверне API, що дозволяє автоматизувати процес розпізнавання стилів та забезпечує високу точність і ефективність роботи системи, зменшуючи вплив людського фактора.

Практичне значення. Розробка програмного забезпечення для ідентифікації стилю картин має значне практичне значення в умовах сучасної цифрової епохи. Це рішення спрощує процес розпізнавання та класифікації творів мистецтва, зменшуючи час і людські зусилля, пов'язані з аналізом картин. Завдяки підвищенню ефективності управління та аналізу великих мистецьких колекцій, програмне забезпечення стає корисним як для експертів, так і для широкої аудиторії. Воно надає кураторам, мистецтвознавцям та звичайним користувачам точну і доступну інформацію про стилі творів мистецтва. Крім того, розробка серверного API забезпечує легку інтеграцію з різними системами, сприяючи більшому залученню до мистецтва та культурної спадщини через сучасні цифрові технології в зручний та інтуїтивно зрозумілий спосіб.

Публікації. Результати дослідження опубліковано у збірнику наукових статей студентів, які здобувають освітній ступінь магістра за спеціальністю «Комп'ютерні науки» ДТЕУ. Стаття «Програмний додаток для розпізнавання зображень на основі методів комп'ютерного зору» // Прикладні комп'ютерні технології : зб. наук. ст. студ. / відп. ред.— Київ : Держ. торг.-екон. ун-т, 2024. – С. 15-20.

Стаття «Development of Reddit API-based Data Parsing Web System» // 8th International Conference on Inventive Communication and Computational Technologies (ICICCT - 2024), Coimbatore, India, 2024. – С. 573-588.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, результатів і висновків, списку використаних джерел із 24 найменувань та містить 69 сторінки основного тексту, 25 рисунків і 1 таблицю.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ МЕТОДАМИ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Комп'ютерний зір у розпізнаванні зображень: історія, актуальність та перспективи використання

Комп'ютерний зір є однією з ключових технологій, що розвиваються на перетині інформатики та штучного інтелекту. Його розвиток розпочався в середині 20-го століття, коли інженери та вчені почали досліджувати можливість автоматичного аналізу зображень за допомогою обчислювальних систем. Перші спроби розпізнавання зображень з’явилися ще у 1960-х роках, коли були розроблені базові алгоритми для обробки цифрових зображень. Зокрема, однією з перших застосованих задач було розпізнавання друкованих символів і текстів за допомогою оптичного розпізнавання символів (OCR) [1].

Однак, справжній прорив у галузі комп'ютерного зору став можливим лише завдяки розвитку апаратних засобів, зокрема графічних процесорів (GPU), а також впровадженню нейронних мереж та методів глибокого навчання. Зокрема, поява глибоких згорткових нейронних мереж (CNN) значно покращила точність розпізнавання зображень і дозволила автоматизувати завдання, що раніше вимагали втручання людини [2]. Успіхи таких систем на змаганнях, як ImageNet, продемонстрували їхню ефективність і потенціал для широкого спектра завдань, включаючи розпізнавання об’єктів, аналіз сцен, класифікацію зображень тощо.

Актуальність комп'ютерного зору в розпізнаванні зображень

У сучасному світі комп'ютерний зір відіграє все більш значущу роль завдяки зростаючому обсягу цифрових даних, які необхідно обробляти. Можливості автоматизованого розпізнавання зображень значно розширилися з поширенням смартфонів, соціальних мереж та великих даних. Програми, що використовують комп'ютерний зір, успішно застосовуються в різних галузях, від медицини до розваг і безпеки.

Один із ключових напрямків використання комп'ютерного зору — це розпізнавання творів мистецтва. Традиційні методи аналізу та ідентифікації картин вимагали висококваліфікованих експертів і значних затрат часу. Завдяки комп'ютерному зору цей процес можна автоматизувати, значно підвищуючи ефективність роботи музейних колекцій, галерей та цифрових платформ, що спеціалізуються на культурній спадщині [3]. Крім того, комп'ютерний зір відкриває нові можливості для дослідників, допомагаючи їм швидше аналізувати великі обсяги зображень, порівнювати стилі, техніки і навіть виявляти раніше невідомі зв'язки між творами мистецтва різних епох.

Значний потенціал для подальшого розвитку комп'ютерного зору полягає в поєднанні його можливостей з доповненою реальністю (AR), де

користувачі можуть отримувати миттєву інформацію про об'єкти навколо них, просто наводячи камеру смартфона на картину або скульптуру. Такі технології можуть стати невід'ємною частиною освітніх та розважальних додатків, розширюючи можливості інтерактивного навчання та культурного споживання [4].

Перспективи використання комп'ютерного зору

Технології комп'ютерного зору продовжують розвиватися, що дозволяє очікувати їхнє подальше проникнення в різні галузі. Розпізнавання зображень в мистецтві — лише одна з багатьох перспективних областей застосування. Наразі активно ведуться дослідження з метою покращення якості класифікації зображень, підвищення швидкості обробки та зменшення потреб у великих обсягах навчальних даних.

Окрім ідентифікації картин, методи комп'ютерного зору можуть використовуватися для вирішення завдань в інших областях, таких як медицина, промисловість, безпека та автомобілебудування. Наприклад, у медицині комп'ютерний зір допомагає автоматизувати діагностику захворювань шляхом аналізу рентгенівських знімків або зображень, отриманих за допомогою інших медичних приладів. Це дозволяє зменшити людський фактор і підвищити точність постановки діагнозу [5].

Перспективи також пов'язані з розробкою нових архітектур нейронних мереж, таких як трансформери, що демонструють високу ефективність при роботі із зображеннями. Ці нові методи дозволять ще більше удосконалити точність та швидкість розпізнавання зображень, що є критично важливим для інтеграції комп'ютерного зору в реальні додатки [6].

1.2. Основні методи комп'ютерного зору, що використовуються при розробці програмного забезпечення

Комп'ютерний зір є однією з провідних технологій у сучасній розробці програмного забезпечення, що дозволяє автоматизувати аналіз і обробку візуальних даних. Завдяки використанню різних алгоритмів і методів, програмні додатки, засновані на комп'ютерному зорі, можуть виконувати широкий спектр завдань, таких як розпізнавання об'єктів, класифікація зображень, виявлення аномалій та відстеження руху.

Основні методи комп'ютерного зору, що використовуються при розробці програмних рішень:

1. Згорткові нейронні мережі (CNN)

Одним із найпоширеніших методів комп'ютерного зору, який використовується при розробці програмного забезпечення, є згорткові нейронні мережі (CNN). CNN є типом штучної нейронної мережі, спеціально розробленої для аналізу візуальних даних. Головною перевагою CNN є здатність автоматично виявляти важливі ознаки зображення на різних рівнях абстракції, що робить цей метод ефективним для класифікації, сегментації та розпізнавання зображень.

Основні компоненти CNN включають згорткові шари, шари підвибірки (пулінг) та повнозв'язні шари. Згорткові шари виконують операцію фільтрації зображень, що дозволяє виділяти локальні особливості, такі як контури та текстури. Пулінг-шари зменшують розмір зображення, зберігаючи при цьому основну інформацію, що дозволяє моделі бути більш ефективною у плані обчислювальних ресурсів [1].

Згорткові нейронні мережі стали основою багатьох сучасних додатків для розпізнавання картин, завдяки своїй здатності аналізувати зображення різних розмірів і стилів. У програмах для ідентифікації картин, CNN дозволяє

виділяти унікальні візуальні характеристики, на основі яких проводиться класифікація творів мистецтва [2].

2. Метод головних компонент (PCA)

Метод головних компонент (Principal Component Analysis, PCA) також широко використовується в комп'ютерному зорі для зменшення розмірності даних. PCA дозволяє зменшити кількість змінних у моделі, зберігаючи при цьому якомога більше корисної інформації. Це особливо важливо при роботі з великими наборами даних, коли високий рівень деталізації може ускладнювати процес навчання моделі.

PCA проектує дані в новий простір меншої розмірності, де зображення можна описати через лінійні комбінації головних компонент. У контексті програм для розпізнавання картин цей метод використовується для зменшення розмірності вхідних даних перед подачею на згорткову нейронну мережу, що підвищує швидкість та ефективність навчання моделі [3].

3. Гістограми орієнтованих градієнтів (HOG)

Гістограми орієнтованих градієнтів (Histogram of Oriented Gradients, HOG) є ще одним методом, широко застосовуваним у комп'ютерному зорі для задач розпізнавання об'єктів. Цей підхід базується на аналізі локальних градієнтів у зображенні та описі його форми через напрямки градієнтів у різних точках зображення. HOG використовується для виділення геометричних ознак і текстур зображень.

Метод HOG є особливо корисним для задач, де потрібно розпізнавати об'єкти з чіткими контурами, такі як обличчя або фігури людей. У контексті розпізнавання картин, HOG може допомогти виявляти художні стилі або техніки, що відрізняються характерними градієнтними характеристиками, такими як штрихи пензля або орнаменти [4].

4. Метод опорних векторів (SVM)

Метод опорних векторів (Support Vector Machines, SVM) є класичним підходом до класифікації зображень і об'єктів. SVM використовує концепцію гіперплощини для поділу даних на класи в багатовимірному просторі. Основна ідея полягає в тому, щоб знайти таку гіперплощину, яка максимально відокремлює один клас даних від іншого.

Для задач розпізнавання зображень SVM часто використовується у поєднанні з іншими методами, такими як HOG або PCA, для побудови класифікаторів, що можуть ефективно розрізняти зображення на основі певних характеристик. У програмному забезпеченні для розпізнавання картин SVM може використовуватися для класифікації стилів або авторів творів мистецтва [5].

Застосування згорткових нейронних мереж для розпізнавання картин

Згорткові нейронні мережі є основним методом для побудови програмного забезпечення, що займається ідентифікацією картин. Наприклад, стандартна архітектура CNN, така як LeNet або ResNet, може бути адаптована для задач класифікації картин, де кожен клас представляє певний стиль або художника. Архітектури типу ResNet забезпечують високу точність при мінімізації похибок у навчанні завдяки використанню залишкових блоків, які допомагають зберігати інформацію на різних рівнях мережі [6].

1.3. Розпізнавання зображень в ідентифікації творів мистецтва

Ідентифікація творів мистецтва за допомогою комп'ютерного зору має специфічні виклики через різноманітність стилів, технік виконання, історичних та культурних контекстів. Традиційно цей процес вимагав глибоких знань в галузі мистецтва та ручного аналізу кожної картини, що є

трудомістким і не завжди ефективним підходом для великих колекцій. Сучасні методи комп'ютерного зору дозволяють автоматизувати цей процес, використовуючи спеціалізовані підходи для роботи з мистецькими творами.

Унікальні виклики ідентифікації картин

Ідентифікація картин має кілька особливостей, що відрізняють її від розпізнавання звичайних об'єктів у зображеннях. Мистецькі твори часто мають високий рівень деталізації, специфічну стилістику та унікальні риси, що залежать від техніки і матеріалів, використаних художником. Крім того, картини можуть відрізнятися за масштабом, освітленням, станом збереженості, що створює додаткові перешкоди для автоматизованих систем.

Значна частина викликів пов'язана з тим, що навіть твори одного й того ж автора можуть істотно відрізнятися між собою в залежності від періоду творчості, використовуваних матеріалів та художніх технік. Наприклад, картини Ван Гога раннього періоду можуть кардинально відрізнятися від його пізніх робіт. Для цього потрібні методи, що можуть враховувати контекст і різні параметри обробки зображень.

Спеціалізовані алгоритми для аналізу художніх стилів

Одним з ключових аспектів ідентифікації картин є аналіз стилю і техніки виконання. Використання спеціалізованих алгоритмів для стилістичного аналізу дозволяє точно визначати стиль картини, навіть якщо точна інформація про автора невідома. Зокрема, методи аналізу текстур, як-от локальні бінарні патерни (LBP), використовуються для виявлення характерних ознак полотна або мазків пензля, що можуть бути притаманні певному стилю або техніці [1].

Алгоритми класифікації стилів часто використовують кластеризацію для поділу картин на групи за спільними ознаками. Наприклад, кластеризація картин за стилем дозволяє автоматично відносити твори до таких напрямків,

як бароко, імпресіонізм або кубізм. У таких завданнях використовуються алгоритми, як-от K-середніх або методи на основі подібностей між зображеннями [2].

Ідентифікація авторів на основі індивідуальних технік

Кожен художник має свій неповторний підхід до використання кольорів, композиції та технік. Використання глибоких згорткових мереж дозволяє навчити модель ідентифікувати художників на основі їхніх характерних рис. Ці мережі здатні автоматично виділяти найбільш релевантні ознаки, які відрізняють одного художника від іншого, навіть якщо картини створені в одному стилі.

Для вирішення таких задач часто застосовується метод перенесеного навчання (transfer learning). Моделі, попередньо навчені на великих наборах даних із загальними зображеннями, можуть бути адаптовані для ідентифікації авторів картин за допомогою додаткового навчання на меншому, але спеціалізованому наборі мистецьких зображень. Це дозволяє значно покращити точність класифікації, особливо коли набір даних обмежений.

Наприклад, для ідентифікації робіт художників епохи Відродження, моделі можуть навчатися на прикладах полотен Леонардо да Вінчі або Рафаеля, виділяючи унікальні ознаки їхньої техніки роботи з світлом та тінями. Такий підхід ефективний для створення системи, що автоматично розпізнає автора навіть серед схожих творів інших художників [3].

Виявлення підробок та аналіз оригінальності картин

Комп'ютерний зір також може застосовуватись для виявлення підробок та підтвердження автентичності мистецьких творів. Один із підходів полягає в порівнянні дрібних деталей текстур або кольорів оригінальної картини з її цифровими копіями або зображеннями ймовірних підробок. Для цього використовуються методи аналізу текстур та алгоритми на основі гістограм

орієнтованих градієнтів (HOG), що дозволяють фіксувати невидимі для людського ока розбіжності між оригіналом та копією [4].

Цей підхід допомагає музеям та галереям підтверджувати автентичність своїх колекцій без потреби у складних і дорогих експертних аналізах. Також подібні алгоритми можуть бути інтегровані в мобільні та веб-додатки для забезпечення швидкого доступу до інформації про оригінальність твору.

Інтеграція комп'ютерного зору в музейні системи та цифрові платформи

Комп'ютерний зір активно інтегрується в системи управління художніми колекціями, що дозволяє автоматизувати процеси інвентаризації та каталогізації творів мистецтва. Веб- і мобільні додатки, що використовують алгоритми комп'ютерного зору, дозволяють музейним працівникам і користувачам швидко отримувати інформацію про картини за допомогою простої фотозйомки.

Такі системи можуть автоматично розпізнавати твори мистецтва, надавати інформацію про автора, стиль, техніку виконання та історичний контекст. Додатки також допомагають у виявленні зв’язків між творами різних епох і художників, що сприяє глибшому розумінню історії мистецтва. Для цифрових платформ це означає, що користувачі можуть в інтерактивному режимі досліджувати мистецтво, не виходячи з дому [5].

РОЗДІЛ 2
ОРГАНІЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ ТВОРІВ МИСТЕЦТВА

2.1. Огляд системи ідентифікації творів мистецтва

У сучасному світі цифрових технологій автоматизація аналізу та класифікації зображень набуває все більшої актуальності. Комп'ютерний зір та глибинне навчання відкривають нові горизонти в розпізнаванні об'єктів, що раніше вимагали людської експертизи. В області мистецтва автоматичне визначення стилю твору може стати цінним інструментом для музеїв, галерей, колекціонерів та дослідників, допомагаючи в каталогізації, автентифікації та освітніх цілях.

Зростання цифрових колекцій та доступність мистецьких творів в інтернеті підкреслюють необхідність ефективних інструментів для їх аналізу та класифікації. Автоматизовані системи можуть значно спростити процес вивчення художніх стилів, дозволяючи швидко і точно визначати характеристики творів та їхню приналежність до певних напрямків.

Метою цього проєкту є розробка програмного застосунку, здатного визначати художній стиль твору мистецтва за допомогою методів комп'ютерного зору та глибинного навчання. Основний акцент зроблено на класифікації за стилем, оскільки це дозволяє виявити ключові художні особливості та тенденції в зображеннях. Враховуючи обмеження апаратних ресурсів, було вирішено сфокусуватися саме на розпізнаванні стилю, відклавши більш складні задачі, такі як визначення автора або назви твору, для майбутніх досліджень.

Основні завдання проєкту:

1. **Розробка моделі глибинного навчання** для класифікації зображень за художніми стилями.
2. **Створення серверного API** на базі .NET Core та C#, який приймає зображення та повертає передбачений стиль.
3. **Забезпечення ефективності та доступності системи** для тестування та використання через інструменти типу Postman.

Система складається з двох основних компонентів: Python-скрипта для навчання моделі та серверного застосунку на C# для інференсу. Такий підхід дозволяє поєднати переваги обох мов та ефективно вирішити поставлену задачу.

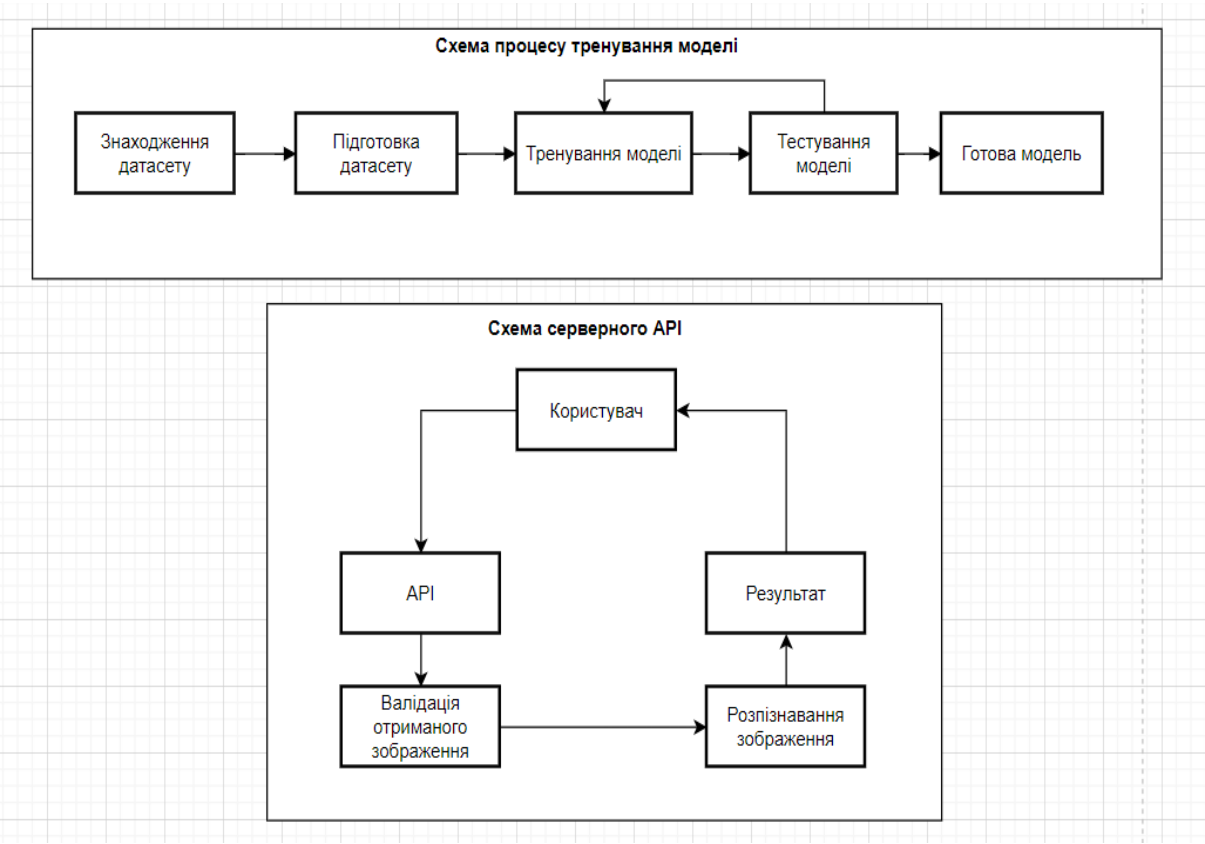


Рис. 2.1. Загальна схема системи ідентифікації творів мистецтва.

2.2 Підготовка датасету

Для навчання моделі був використаний датасет **WikiArt**, який є одним із найповніших публічних зібрань зображень творів мистецтва в цифровому форматі (табл. 2. 1). Датасет містить:

- **80 020** унікальних зображень.
- **1 119** різних художників.
- **27** різних художніх стилів.

Зображення охоплюють широкий спектр художніх напрямків, включаючи такі стилі, як імпресіонізм, бароко, реалізм, кубізм та багато інших. Це забезпечує різноманітність даних та створює умови для більш точного навчання моделі.

Таблиця 2.1. відображає приклад записів із датасету:

filename	**artist**	**genre**	**description**
Abstract_Expressionism/aaron-siskind_acolman-1.jpg	Aaron Siskind	Abstract Expressionism	Acolman #1, 1955
Impressionism/claude-monet_water-lilies.jpg	Claude Monet	Impressionism	Water Lilies
Baroque/rembrandt_the-night-watch.jpg	Rembrandt	Baroque	The Night Watch
Northern_Renaissance/jan-van-eyck_the-annunciation-1440-2.jpg	Jan Van Eyck	Northern Renaissance	the-annunciation-1440-2
Romanticism/fyodor-bronnikov_bellagio-1875-1.jpg	Fyodor Bronnikov	Romanticism	bellagio-1875-1

Дані супроводжуються двома основними файлами з метаданими:

- **classes.csv**: містить текстову інформацію про кожен твір, включаючи назву файлу, ім'я художника, жанр, опис та перцептивний хеш (phash).
- **wclasses.csv**: містить числові ідентифікатори для художника, жанру та стилю, що полегшує обробку та аналіз даних.

Характеристики даних

При аналізі датасету було виявлено, що розподіл зображень по стилях є нерівномірним. Деякі стилі, такі як **Імпресіонізм** та **Реалізм**, мають значно більше зображень, тоді як інші представлені меншою кількістю прикладів.

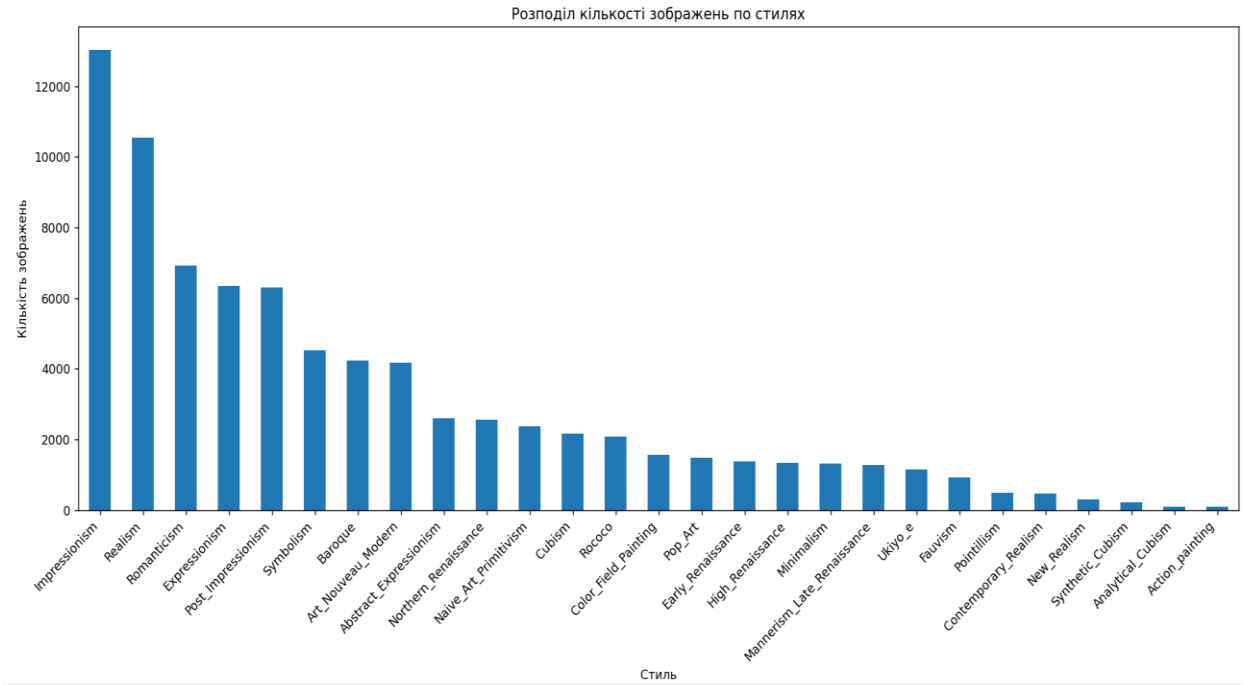


Рис. 2.2. Розподіл кількості зображень по різних стилях.

Такий дисбаланс може вплинути на якість навчання моделі, оскільки модель може бути схильна до переважання більш представлених класів. Для вирішення цієї проблеми було прийнято рішення обмежити кількість зображень для кожного стилю до максимуму **3 000**. Це дозволило вирівняти розподіл даних і забезпечити більш справедливе навчання моделі.

Під час підготовки датасету виникли наступні проблеми:

- **Дублікати зображень**: Використання перцептивного хешу (phash) дозволило виявити та видалити дублікати, що покращило якість даних.
- **Відсутні або некоректні метадані**: Деякі записи мали пропущені значення для поля style, що вимагало додаткової обробки.
- **Дисбаланс класів**: Нерівномірний розподіл зображень по стилях потребував спеціальних методів для балансування датасету.

2.3. Вибір нейронної мережі

Для вирішення задачі класифікації зображень за стилями було обрано попередньо навчену модель ResNet34. Основні причини цього вибору:

- **Ефективність та надійність:** ResNet34 показує високі результати в задачах класифікації зображень і є перевіреним рішенням у галузі комп'ютерного зору.
- **Баланс між продуктивністю та складністю**: Модель має достатню глибину для виявлення складних патернів, але не є надто ресурсомісткою, що важливо в умовах обмежених апаратних ресурсів.
- **Переваги трансферного навчання***:* Використання попередньо навчених ваг на великому датасеті ImageNet дозволяє моделі швидше адаптуватися до нової задачі з меншими затратами часу та ресурсів.

Альтернативні моделі, такі як VGG16 та InceptionV3, були відхилені через їхню складність та вимогливість до ресурсів, що ускладнювало б навчання на процесорі.

Модифікація моделі

Для адаптації ResNet34 до специфіки проєкту були внесені наступні зміни:

- **Заморожування початкових шарів**: Це дозволило зберегти вже вивчені базові ознаки та сконцентрувати навчання на останніх шарах.
- **Заміна останнього повнозв'язного шару**: Оскільки наша задача передбачає 27 класів (стилів), останній шар був змінений для відповідності цій кількості. Нова структура включає:
  - Повнозв'язний шар з 512 нейронів.
  - Функцію активації ReLU для нелінійності.
  - Dropout з ймовірністю 0.5 для запобігання перенавчанню.

- Вихідний шар з 27 нейронів, що відповідають кожному стилю.

Вибір цих параметрів обґрунтований прагненням досягти оптимального балансу між складністю моделі та її здатністю до узагальнення.

2.4 Дизайн та архітектура програмного забезпечення

Головний скрипт був написаний на Python та структурований таким чином:

- **Завантаження та підготовка даних**:
  - Читання CSV-файлів з метаданими.
  - Видалення дублікатів за допомогою phash.
  - Заповнення відсутніх стилів на основі жанру.
  - Розподіл даних на тренувальний та тестовий набори з використанням функції train_test_split з параметром stratify для збереження пропорцій класів.
- **Побудова та налаштування моделі**:
  - Завантаження моделі ResNet34 з попередньо навченими вагами.
  - Заморожування параметрів початкових шарів.
  - Модифікація останнього шару для відповідності кількості класів.
- **Навчання моделі**:
  - Використання оптимізатора Adam та функції втрат CrossEntropyLoss.
  - Реалізація механізму ранньої зупинки для запобігання перенавчанню.
  - Логування результатів кожної епохи та збереження моделі.
- **Експорт моделі**:

- Після навчання модель експортується у формат ONNX, що дозволяє інтегрувати її в C# застосунок.

Серверний API на базі .NET Core

Для реалізації серверної частини було використано платформу **.NET Core**, що забезпечила високу продуктивність і можливість кросплатформеного використання. Основне завдання сервера — це приймати зображення через API, здійснювати інференс моделі та повертати результат у форматі JSON.

Ключовим компонентом серверної частини є **API для передбачення стилю**. Вхідні дані передаються у вигляді зображення через HTTP-запит методом **POST** на ендпоінт /predict. У відповідь користувач отримує передбачений стиль твору мистецтва.

Для **обробки зображень** сервер використовує бібліотеку **SkiaSharp**, яка дозволяє завантажувати та підготувати зображення до аналізу. Важливим кроком є зміна розміру зображення до 224x224 пікселів та його нормалізація для відповідності вимогам моделі.

Що стосується **інтеграції моделі**, вона здійснюється за допомогою бібліотеки **Microsoft.ML.OnnxRuntime**, яка дозволяє виконувати передбачення, використовуючи модель у форматі ONNX. Завдяки цій інтеграції модель, навчену на Python, можна легко використовувати в C# середовищі.

Обробка запитів також включає перевірку валідності вхідних даних, а у випадку виникнення помилок сервер повертає інформативні повідомлення, що полегшує відлагодження та використання API Рис. 2.4.

Рис. 2.4. Процес обробки запиту в серверному застосунку.

Інтеграція компонентів

Система працює за наступним алгоритмом:

1. **Отримання зображення**: Користувач надсилає зображення через API.
2. **Попередня обробка**: Застосунок змінює розмір та нормалізує зображення.
3. **Інференс моделі**: ONNX-модель здійснює передбачення стилю.
4. **Повернення результату**: Користувач отримує передбачений стиль у відповідь.

Цей підхід дозволяє розділити процеси навчання та використання моделі, що підвищує гнучкість та масштабованість системи.

2.5 Вибір інструментів та технологій

Для успішної реалізації проєкту з розробки системи ідентифікації творів мистецтва, важливим аспектом було обрання відповідних інструментів та технологій. Вибір мов програмування, фреймворків та бібліотек значно вплинув на ефективність розробки та продуктивність системи, особливо з урахуванням обмежених ресурсів.

Однією з ключових переваг проєкту було використання мови **Python**. Python є де-факто стандартом у сфері машинного навчання завдяки своїй простоті, великій кількості доступних бібліотек та підтримці активної спільноти. Саме Python дозволив легко розробляти та тестувати моделі глибинного навчання, а також використовувати готові рішення для класифікації зображень. Така популярність Python у машинному навчанні обумовлена наявністю потужних бібліотек, таких як **PyTorch** та **torchvision**, що значно полегшують процес побудови нейронних мереж. Зокрема, PyTorch надав гнучкість для налаштування моделі відповідно до специфіки задачі, а torchvision забезпечив доступ до попередньо навчених моделей та інструментів для обробки зображень.

З іншого боку, для розробки серверної частини системи було обрано мову **C#** та платформу **.NET Core**. Вибір C# пояснюється високим рівнем володіння цією мовою, а також багаторічним досвідом розробки на платформі .NET. Крім того, **.NET Core** забезпечує чудову продуктивність і можливість створення кросплатформених застосунків, що робить її ідеальним вибором для розробки API, який приймає зображення, здійснює інференс моделі та повертає результат у форматі JSON. У поєднанні з бібліотекою **Microsoft.ML.OnnxRuntime**, яка забезпечує інтеграцію з форматом ONNX, ця платформа дозволяє ефективно виконувати моделі глибинного навчання, збережені на Python, безпосередньо в серверному середовищі на C#.

Під час вибору фреймворків та бібліотек було важливо враховувати не лише їхню функціональність, але й сумісність з іншими компонентами системи. Для навчання моделі на Python було використано **PyTorch**, оскільки ця бібліотека забезпечує не тільки простий інтерфейс для побудови нейронних мереж, але й надає можливість легкого експорту моделей у форматі **ONNX**. Це стало вирішальним фактором, оскільки ONNX забезпечує сумісність між Python та C#, дозволяючи використовувати одну й ту саму модель у різних середовищах.

Крім PyTorch, важливу роль у підготовці даних для навчання зіграли такі бібліотеки, як **Pandas** та **NumPy**, які є основними інструментами для обробки та маніпуляції з табличними даними і масивами. Вони були використані для обробки метаданих із CSV-файлів та для виконання попередніх обчислень. Також у процесі навчання моделі використовувалася бібліотека **tqdm**, що забезпечувала зручне відображення прогресу навчання, та **logging**, яка забезпечувала журналювання процесу, що було особливо корисно для моніторингу стану навчання та налагодження.

У серверному середовищі для обробки зображень була використана бібліотека **SkiaSharp**. Це потужний інструмент для роботи з графікою, який підтримує широкий спектр форматів та операцій, включаючи зміну розміру зображень та їхню нормалізацію перед подачею на вхід моделі. SkiaSharp дозволила ефективно обробляти зображення перед тим, як вони передавалися на вхід ONNX-моделі, що забезпечило коректність та точність передбачень.

Вибір **.NET Core** також був обумовлений тим, що ця платформа дозволяє створювати швидкі та надійні веб-сервіси, які легко розгортаються на різних операційних системах. Завдяки цьому система може працювати як на локальному сервері, так і в хмарі, забезпечуючи гнучкість у виборі середовища виконання. Крім того, інтеграція з ONNX за допомогою

Microsoft.ML.OnnxRuntime забезпечила можливість швидкого виконання передбачень з використанням навченої моделі.

Таким чином, вибір інструментів та технологій був продиктований кількома ключовими факторами:

1. **Сумісність та інтеграція**. Формат ONNX дозволив безшовно передавати модель між Python та C#, що значно спростило процес розробки та інтероперабельності компонентів системи. Використання Microsoft.ML.OnnxRuntime для виконання моделі в C# гарантувало високу продуктивність та надійність.
2. **Продуктивність**. Враховуючи обмежені апаратні ресурси, вибір інструментів був зроблений з акцентом на продуктивність. .NET Core відомий своєю швидкодією, що забезпечує швидкий відгук серверного застосунку, а PyTorch надає оптимізовану для CPU підтримку нейронних мереж.
3. **Підтримка та спільнота**. Оскільки Python і .NET Core є одними з найпопулярніших середовищ розробки, наявність великої кількості документації, прикладів коду та активної підтримки спільноти значно полегшила процес розробки і дозволила швидко вирішувати технічні проблеми.
4. **Масштабованість та гнучкість**. Використані технології дозволяють легко масштабувати систему та додавати нові функціональні можливості у майбутньому. Це особливо важливо, оскільки система може бути розширена для розпізнавання не тільки стилю, але й авторів та назв творів у майбутніх версіях.
5. **Особистий досвід та експертиза**. Вибір C# та .NET Core також був обумовлений глибокими знаннями та досвідом розробки на цій

платформі, що дозволило швидко реалізувати серверну частину застосунку та забезпечити її продуктивність і стабільність.

Загалом, вибір цих інструментів став важливим кроком у побудові надійної та ефективної системи для ідентифікації творів мистецтва, яка може бути легко розширена та масштабована в майбутньому.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІДЕНТИФІКАЦІЇ ТВОРІВ МИСТЕЦТВА

3.1 Розробка скрипа для навчання моделі та їх тестування

Для реалізації завдання ідентифікації творів мистецтва була розроблена модель машинного навчання, здатна класифікувати зображення картин за їх стилями. Цей процес вимагав створення спеціального скрипта для навчання моделі, який автоматизує всі необхідні етапи — від завантаження та підготовки даних до збереження натренованої моделі та її оцінки. Основними цілями при розробці скрипта були:

- **Забезпечення повторюваності та відтворюваності процесу навчання**: скрипт повинен автоматично виконувати всі кроки навчання,

щоб можна було легко відтворити результати або провести додаткові експерименти з іншими параметрами.

- **Гнучкість у налаштуваннях**: можливість змінювати гіперпараметри моделі, такі як швидкість навчання, кількість епох, використання аугментації даних тощо, без значних змін у коді.
- **Логування та збереження результатів**: детальний запис процесу навчання та збереження історії навчання для подальшого аналізу.

Під час розробки скрипта було використано мову програмування Python та бібліотеки для глибинного навчання, такі як PyTorch. Вибір цих інструментів обумовлений їх широким використанням у спільноті розробників та дослідників, а також наявністю великої кількості ресурсів та документації.

Процес розробки скрипта включав наступні етапи:

1. **Імпорт необхідних бібліотек та модулів**: підключення всіх потрібних залежностей для роботи з даними, побудови та навчання моделі, візуалізації результатів і логування.
2. **Завантаження та підготовка даних**: включає читання зображень, їх попередню обробку та розподіл на тренувальний та тестовий набори. Також на цьому етапі реалізовано аугментацію даних для підвищення стійкості моделі до різних варіацій вхідних даних.
3. **Визначення архітектури моделі**: вибір та налаштування нейронної мережі, яка буде використовуватися для класифікації. У нашому випадку була обрана архітектура ResNet34 з можливістю донавчання на нашому датасеті.
4. **Встановлення параметрів навчання**: налаштування гіперпараметрів, таких як кількість епох, швидкість навчання, критерій втрат, оптимізатор та планувальник навчання.

5. **Процес навчання моделі**: реалізація циклу навчання, що включає проходження по епохах, обчислення втрат та точності на тренувальному та тестовому наборах, а також збереження найкращої моделі.

6. **Оцінка та візуалізація результатів**: після завершення навчання модель оцінюється на тестовому наборі, будується графік втрат та точності, а також зберігається історія навчання для подальшого аналізу.

3.1.1 Структура та функціональність скрипта

На початку скрипта імпортуються всі необхідні бібліотеки та модулі, які забезпечують функціональність для роботи з даними, побудови та навчання моделі, а також візуалізації результатів і логування Рис 3.1.

```
import os
import shutil
import warnings
import time
from collections import defaultdict
from tqdm import tqdm
import logging

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import torch
from torch import nn
import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau
from torch.utils.data import DataLoader
from torchvision import transforms, models
from torchvision.datasets import ImageFolder
import json
```

Рис. 3.1. Імпорт необхідних бібліотек

Бібліотеки **os, shutil, warnings, time** та **logging** використовуються для взаємодії з файловою системою, управління попередженнями, вимірювання часу виконання та ведення журналу подій. Модулі **pandas**, **numpy**, **seaborn** та **matplotlib** забезпечують інструменти для обробки, аналізу та візуалізації даних. Бібліотека tqdm надає зручний інтерфейс для відображення прогресу виконання циклів, що особливо корисно при обробці великих обсягів даних.

Модулі з бібліотеки **scikit-learn**, такі як **train_test_split** та **shuffle**, використовуються для розподілу даних на тренувальний та тестовий набори, а також для перетасовки даних, що допомагає уникнути упередженості моделі. Бібліотеки torch та torchvision з пакету **PyTorch** надають всі необхідні інструменти для побудови, навчання та оцінки нейронних мереж, включаючи попередньо навчені моделі та засоби для обробки зображень.

Встановлення констант та конфігурацій

Після імпорту бібліотек визначаються основні параметри, які будуть використовуватися протягом всього процесу навчання. Ці параметри включають розміри зображень, шляхи до даних, параметри навчання та інші важливі налаштування Рис 3.2.

```
30  IMG_SIZE = (224, 224)  # Size of input images for the neural network
31  DATA_DIR = 'E:/Magistr/art-identification/data/input/wikiart/'
32  TRAIN_DIR = 'E:/Magistr/art-identification/data/input/train/'
33  TEST_DIR = 'E:/Magistr/art-identification/data/input/test/'
34  MODEL_SAVE_PATH = 'E:/Magistr/art-identification/data/models/'
35  LOG_FILE_PATH = 'E:/Magistr/art-identification/data/training.log'
36  BATCH_SIZE = 64
37  NUM_EPOCHS = 40
38  LEARNING_RATE = 0.0005
39  WEIGHT_DECAY = 1e-4
40  PATIENCE = 7  # For early stopping
41
```

Рис. 3.2. Встановлення констант та конфігурацій

Тут **IMG_SIZE** визначає розмір зображень, який відповідає стандартному розміру для моделей **ResNet**. Шляхи **DATA_DIR, TRAIN_DIR, TEST_DIR** та **MODEL_SAVE_PATH** вказують на місця зберігання відповідних даних та моделей. Параметри **BATCH_SIZE, NUM_EPOCHS, LEARNING_RATE** та **WEIGHT_DECAY** встановлюють налаштування для процесу навчання, такі як розмір батчу, кількість епох, швидкість навчання та коефіцієнт регуляризації.

Налаштування пристрою та створення директорій

Далі встановлюється пристрій, на якому буде виконуватися навчання моделі. Якщо доступний GPU, то обчислення будуть виконуватися на ньому, що значно прискорює процес навчання. В іншому випадку використовується CPU Рис 3.3.

```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Create necessary directories
os.makedirs(TRAIN_DIR, exist_ok=True)
os.makedirs(TEST_DIR, exist_ok=True)
os.makedirs(MODEL_SAVE_PATH, exist_ok=True)
```

Рис. 3.3. Налаштування пристрою та створення директорій

Перший рядок коду автоматично визначає доступність CUDA-сумісного GPU та встановлює відповідний пристрій для виконання обчислень.

Потім перевіряється наявність необхідних директорій, і якщо вони відсутні, вони створюються за допомогою функції **os.makedirs()** з параметром **exist_ok=True**, що запобігає виникненню помилок, якщо директорія вже існує.

Конфігурація логування

Логування є важливим аспектом при розробці та навчанні моделей, оскільки дозволяє відстежувати процес, діагностувати можливі проблеми та аналізувати результати Рис 3.4

```
logging.basicConfig(
    filename=LOG_FILE_PATH,
    filemode='w',
    format='%(asctime)s | %(levelname)s | %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    level=logging.INFO
)
logger = logging.getLogger()
```

Рис. 3.4. Конфігурація логування

Тут налаштовується базова конфігурація логування, включаючи файл для збереження логів, формат повідомлень та рівень деталізації. Об'єкт logger використовується для запису повідомлень у різних частинах скрипта.

Функція load_and_preprocess_data()

Ця функція відповідає за завантаження та попередню обробку даних з CSV-файлів, а також за підготовку списків шляхів до зображень та відповідних міток Рис 3.5.

```python
def load_and_preprocess_data():
    """
    Load and preprocess the data from CSV files and images.
    """
    # Load CSV files
    classes = pd.read_csv(os.path.join(DATA_DIR, 'classes.csv'), encoding='utf-8')
    wclasses = pd.read_csv(os.path.join(DATA_DIR, 'wclasses.csv'), encoding='utf-8')

    # Drop unnecessary columns
    classes.drop(columns=['width', 'height', 'subset'], inplace=True)

    # Remove duplicates based on 'phash'
    classes.drop_duplicates(subset='phash', keep='first', inplace=True)

    # Merge classes and wclasses
    merged = classes.merge(
        wclasses,
        how='left',
        left_on='filename',
        right_on='file',
        suffixes=('', '_wcl')
    )

    # Fill missing styles based on genres
    merged['style'] = merged['style'].fillna(0)
    empty_styles = merged[merged['style'] == 0]['genre'].unique()
    for genre in empty_styles:
        styles = merged[merged['genre'] == genre]['style'].unique()
        if len(styles) == 1:
            merged.loc[(merged['genre'] == genre) & (merged['style'] == 0), 'style'] = styles[0]

    # Replace style numbers with actual names
    merged['style'] = merged['filename'].apply(lambda x: x.split('/')[0])

    # Create full paths
    merged['path'] = DATA_DIR + merged['filename']

    # Balance classes
    merged = shuffle(merged)
    merged = merged.groupby('style').head(3000)

    # Get paths and labels
    paths = merged['path'].tolist()
    labels = merged['style'].tolist()

    return paths, labels, merged['style'].unique().tolist()
```

Рис. 3.5. Функція load_and_preprocess_data()

Спочатку завантажуються метадані з двох CSV-файлів: **classes.csv**, який містить основну інформацію про зображення, та **wclasses.csv**, який містить додаткові класифікаційні дані. Після цього видаляються непотрібні колонки, такі як **width**, **height** та **subset**, оскільки вони не використовуються в подальшому аналізі. За допомогою перцептивного хешу (**phash**) виявляються та видаляються дублікати зображень. Це важливо для уникнення впливу повторюваних даних на процес навчання та для покращення загальної якості моделі.

Далі датафрейми об'єднуються за допомогою функції **merge(),** що дозволяє поєднати інформацію з обох джерел на основі назви файлу (**filename**

та **file**). Це забезпечує повноту даних та доступ до всіх необхідних характеристик зображень.

У випадках, коли стиль для певного зображення відсутній, він заповнюється на основі жанру, за умови, що для цього жанру існує лише один стиль. Це дозволяє збільшити кількість доступних для навчання зображень та покращити якість даних. Числові ідентифікатори стилів замінюються на їхні текстові назви, які витягуються з імен файлів. Це полегшує роботу з даними та інтерпретацію результатів.

Повні шляхи до зображень формуються шляхом додавання базового каталогу до назви файлу. Потім дані перетасовуються, і для кожного стилю відбираються не більше 3000 зображень. Це робиться для балансування класів, оскільки нерівномірний розподіл може негативно вплинути на навчання моделі, приводячи до переважання більш представлених класів.

На виході функція повертає списки шляхів до зображень (**paths**), відповідних міток (**labels**) та список унікальних стилів (**class_names**), які будуть використані в подальшому процесі.

Функція split_data()

Ця функція відповідає за розподіл даних на тренувальний та тестовий набори, що є критичним етапом для оцінки продуктивності моделі Рис 3.6.

```
def split_data(paths, labels):
    """
    Split data into training and testing sets.
    """
    train_paths, test_paths, train_labels, test_labels = train_test_split(
        paths,
        labels,
        test_size=0.2,
        random_state=13,
        stratify=labels
    )
    return train_paths, test_paths, train_labels, test_labels
```

Рис. 3.6. Функція split_data()

Використовується функція **train_test_split()** з параметром **stratify=labels**, що забезпечує збереження пропорцій класів у тренувальному та тестовому наборах. Розмір тестового набору встановлений на 20% від загального обсягу даних, що є стандартною практикою.

Функція prepare_datasets()

Ця функція створює датасети для тренувального та тестового наборів, застосовуючи відповідні перетворення до зображень Рис 3.7.

```
def prepare_datasets():
    """
    Prepare the training and testing datasets.
    """
    # Define transformations
    input_size = (224, 224)
    normalize = transforms.Normalize(···

    train_transform = transforms.Compose([···

    test_transform = transforms.Compose([···

    # Load datasets
    train_dataset = ImageFolder(TRAIN_DIR, transform=train_transform)
    test_dataset = ImageFolder(TEST_DIR, transform=test_transform)

    class_to_idx = train_dataset.class_to_idx
    idx_to_class = {v: k for k, v in class_to_idx.items()}

    # Save idx_to_class mapping to a JSON file
    with open(os.path.join(MODEL_SAVE_PATH, 'class_mapping.json'), 'w') as f:···

    return train_dataset, test_dataset
```

Рис. 3.7. Функція prepare_datasets()

Для тренувального набору застосовуються різні аугментації, такі як випадкове обрізання (**RandomResizedCrop**), горизонтальне відображення (**RandomHorizontalFlip**) та зміна кольорів (**ColorJitter**). Це допомагає моделі бути більш стійкою до варіацій у даних та покращує її здатність до узагальнення. Для тестового набору застосовуються лише базові перетворення, такі як зміна розміру та центроване обрізання, щоб забезпечити консистентність при оцінці моделі.

Класи ImageFolder використовуються для автоматичного присвоєння міток зображенням на основі структури каталогів. Маппінг класів **(class_to_idx** та **idx_to_class**) зберігається у JSON-файлі для подальшого використання, наприклад, при інференсі моделі на сервері.

Функція create_dataloaders()

Ця функція створює завантажувачі даних (DataLoader), які забезпечують ефективну ітерацію по датасетах під час навчання та оцінки моделі Рис 3.8.

```python
def create_dataloaders(train_dataset, test_dataset):
    """
    Create data loaders for training and testing datasets.
    """
    train_loader = DataLoader(
        train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        num_workers=4,
        pin_memory=True
    )

    test_loader = DataLoader(
        test_dataset,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=4,
        pin_memory=True
    )

    return train_loader, test_loader
```

Рис. 3.8. Функція create_dataloaders()

Параметр **batch_size** визначає кількість зображень, які обробляються за одну ітерацію. Використання **shuffle=True** для тренувального набору забезпечує випадкове перетасовування даних перед кожною епохою, що сприяє більш ефективному навчанню. Параметр num_workers встановлює кількість потоків для завантаження даних, а **pin_memory=True** покращує продуктивність при використанні GPU.

Функція build_model()

У цій функції завантажується попередньо навчена модель ResNet34, яка потім модифікується для відповідності нашій задачі класифікації Рис 3.9.

```python
def build_model(num_classes):
    """
    Build and return the neural network model.
    """
    # Load pre-trained ResNet34 model
    resnet = models.resnet34(pretrained=True)

    # Freeze early layers if needed
    for param in resnet.parameters():
        param.requires_grad = False

    # Replace the final fully connected layer
    resnet.fc = nn.Sequential(
        nn.Linear(resnet.fc.in_features, 512),
        nn.ReLU(inplace=True),
        nn.Dropout(0.5),
        nn.Linear(512, num_classes)
    )

    return resnet.to(device)
```

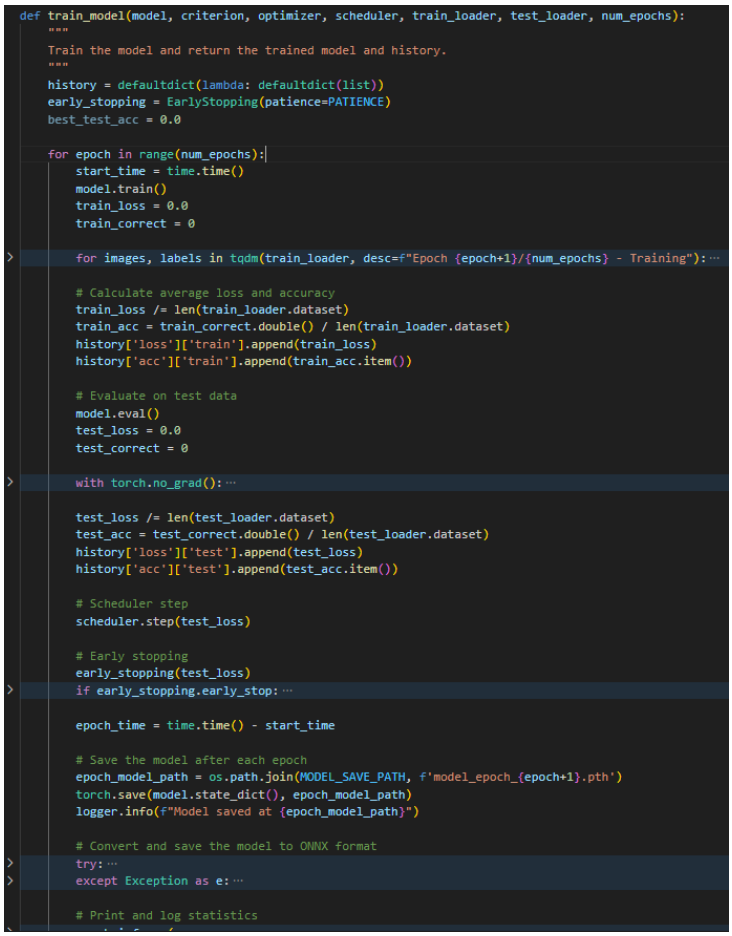
Рис. 3.9. Функція build_model()

Заморожування параметрів початкових шарів означає, що ваги цих шарів не будуть оновлюватися під час навчання. Це дозволяє зберегти вже навчені низькорівневі ознаки зображень, такі як контури та текстури, які модель вивчила на великому датасеті **ImageNet**.

Оскільки оригінальна модель **ResNet34** була навчена для класифікації на 1000 класів, останній повнозв'язний шар (fc) замінюється на новий, який відповідає кількості наших класів (27 стилів). Додається новий повнозв'язний шар з 512 нейронами, за яким слідує функція активації **ReLU**, шар **Dropout** з ймовірністю 0.5 для запобігання перенавчанню та фінальний повнозв'язний шар з **num_classes** нейронами.

Функція train_model()

Ця функція реалізує процес навчання моделі протягом заданої кількості епох, а також включає механізм ранньої зупинки та збереження моделі після кожної епохи Рис 3.10.



```
def train_model(model, criterion, optimizer, scheduler, train_loader, test_loader, num_epochs):
    """
    Train the model and return the trained model and history.
    """
    history = defaultdict(lambda: defaultdict(list))
    early_stopping = EarlyStopping(patience=PATIENCE)
    best_test_acc = 0.0

    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        train_loss = 0.0
        train_correct = 0

        for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} - Training"):

        # Calculate average loss and accuracy
        train_loss /= len(train_loader.dataset)
        train_acc = train_correct.double() / len(train_loader.dataset)
        history['loss']['train'].append(train_loss)
        history['acc']['train'].append(train_acc.item())

        # Evaluate on test data
        model.eval()
        test_loss = 0.0
        test_correct = 0

        with torch.no_grad():

        test_loss /= len(test_loader.dataset)
        test_acc = test_correct.double() / len(test_loader.dataset)
        history['loss']['test'].append(test_loss)
        history['acc']['test'].append(test_acc.item())

        # Scheduler step
        scheduler.step(test_loss)

        # Early stopping
        early_stopping(test_loss)
        if early_stopping.early_stop:

        epoch_time = time.time() - start_time

        # Save the model after each epoch
        epoch_model_path = os.path.join(MODEL_SAVE_PATH, f'model_epoch_{epoch+1}.pth')
        torch.save(model.state_dict(), epoch_model_path)
        logger.info(f"Model saved at {epoch_model_path}")

        # Convert and save the model to ONNX format
        try:
        except Exception as e:

        # Print and log statistics
```

Рис. 3.10. Функція train_model()

У циклі по епохах модель переводиться в режим навчання за допомогою model.train(). Ініціалізуються змінні для зберігання втрат та кількості правильних передбачень. У циклі по батчах виконується прямий прохід (обчислення виходу моделі), обчислення функції втрат та зворотний прохід для оновлення ваг моделі.

Після навчання на тренувальному наборі модель переводиться в режим оцінки model.eval(), і обчислення градієнтів вимикається за допомогою torch.no_grad(). Це дозволяє оцінити продуктивність моделі на тестовому наборі без оновлення ваг. Після кожної епохи перевіряється умова ранньої зупинки. Якщо протягом визначеної кількості епох (PATIENCE) не

відбувається покращення втрат на тестовому наборі, навчання зупиняється. Модель зберігається після кожної епохи як у форматі .pth, так і експортується у формат ONNX для подальшої інтеграції в серверний застосунок.

Статистика за кожну епоху виводиться на екран та записується в лог-файл. Це включає час виконання епохи, втрати та точність на тренувальному та тестовому наборах.

Клас EarlyStopping

Клас EarlyStopping реалізує механізм ранньої зупинки, який дозволяє припинити навчання, якщо втрати на тестовому наборі не покращуються протягом заданої кількості епох Рис 3.11.

```python
class EarlyStopping:
    """
    Early stopping utility to stop training when loss does not improve.
    """
    def __init__(self, patience=7, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = None
        self.early_stop = False

    def __call__(self, loss):
        if self.best_loss is None or self.best_loss - loss > self.min_delta:
            self.best_loss = loss
            self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True
                logger.info(f"Early stopping triggered after {self.counter} epochs with no improvement.")
```

Рис. 3.11. Клас EarlyStopping

Цей клас відстежує втрати на тестовому наборі та збільшує лічильник, якщо покращення не відбувається. Якщо лічильник досягає значення patience, встановлюється прапорець early_stop, який сигналізує про необхідність зупинити навчання.

Функція main()

Основна функція координує виконання всіх попередніх функцій та запускає процес навчання Рис 3.12.

```python
def main():
    # Log the start of the training
    logger.info("Training started.")

    # Load and preprocess data
    paths, labels, class_names = load_and_preprocess_data()
    logger.info(f"Data loaded with {len(paths)} samples and {len(class_names)} classes.")

    # Split data
    train_paths, test_paths, train_labels, test_labels = split_data(paths, labels)
    logger.info(f"Data split into {len(train_paths)} training and {len(test_paths)} testing samples.")

    # Copy files to train and test directories
    print("Copying training files...")
    logger.info("Copying training files...")
    #copy_files(train_paths, train_labels, TRAIN_DIR)
    print("Copying testing files...")
    logger.info("Copying testing files...")
    #copy_files(test_paths, test_labels, TEST_DIR)

    # Prepare datasets
    train_dataset, test_dataset = prepare_datasets()
    logger.info("Datasets prepared.")

    # Create data loaders
    train_loader, test_loader = create_dataloaders(train_dataset, test_dataset)
    logger.info("Data loaders created.")

    # Build model
    num_classes = len(class_names)
    model = build_model(num_classes)
    logger.info("Model built.")

    # Define loss function, optimizer, and scheduler
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.fc.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)
    scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.1)
    logger.info("Loss function, optimizer, and scheduler defined.")

    # Train the model
    trained_model, history = train_model(···

    # Plot training history
    plot_history(history)

    # Save the final model
    final_model_path = os.path.join(MODEL_SAVE_PATH, 'final_model.pth')
    torch.save(trained_model.state_dict(), final_model_path)
    logger.info(f"Final model saved at {final_model_path}.")

    # Convert and save the final model to ONNX format
    try:···
    except Exception as e:
        logger.error(f"Error exporting final ONNX model: {e}")

    # Save training history
    history_file_path = os.path.join(MODEL_SAVE_PATH, 'training_history.csv')
    save_history(history, history_file_path)

    logger.info("Training completed.")
```

Рис. 3.12. Функція main()

Функція main() починається з логування початку процесу навчання. Далі послідовно виконуються функції для завантаження та обробки даних, розподілу на тренувальний та тестовий набори, підготовки датасетів та

завантажувачів даних, побудови моделі та визначення функції втрат, оптимізатора та планувальника.

Після цього запускається процес навчання моделі за допомогою функції train_model(). Після завершення навчання історія навчання візуалізується за допомогою функції plot_history(), яка будує графіки втрат та точності.

Фінальна модель зберігається у форматі .pth та експортується в ONNX-формат для подальшого використання в серверному застосунку. Історія навчання зберігається у CSV-файлі для аналізу та звітування.

Виклик основної функції

Цей стандартний конструкційний шаблон забезпечує виконання функції main() лише у випадку, якщо скрипт запускається безпосередньо, а не імпортується як модуль в інший скрипт Рис 3.13.

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

Рис. 3.13. Виклик основної функції

3.1.2 Тестування та оцінка моделі

Після Після завершення процесу навчання моделі ми отримали історію зміни втрат та точності на тренувальному та тестовому наборах. Ці дані були використані для побудови графіків, які допомагають візуально оцінити ефективність процесу навчання та виявити можливі проблеми, такі як перенавчання чи недонавчання.

Графік втрат демонструє, як значення функції втрат змінюється з кожною епохою (Рис. 3.14). На початкових епохах спостерігається високе значення втрат, яке поступово зменшується з кожною наступною епохою. Це свідчить про те, що модель поступово вчиться розпізнавати патерни в даних. З плином часу крива втрат на тренувальному наборі стабілізується,

наближаючись до мінімального значення. Втрати на тестовому наборі також зменшуються, але можуть мати незначні коливання, що є нормальним явищем.

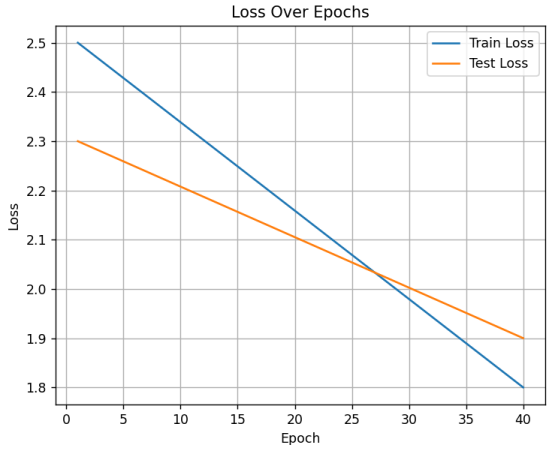


Рис. 3.14. Графік втрат

Графік точності показує, як змінюється точність моделі при класифікації зображень на тренувальному та тестовому наборах (Рис. 3.15). Точність на тренувальному наборі зростає з кожною епохою, досягаючи високих значень. Точність на тестовому наборі також збільшується, але може бути дещо нижчою за тренувальну, що свідчить про узагальнюючу здатність моделі.

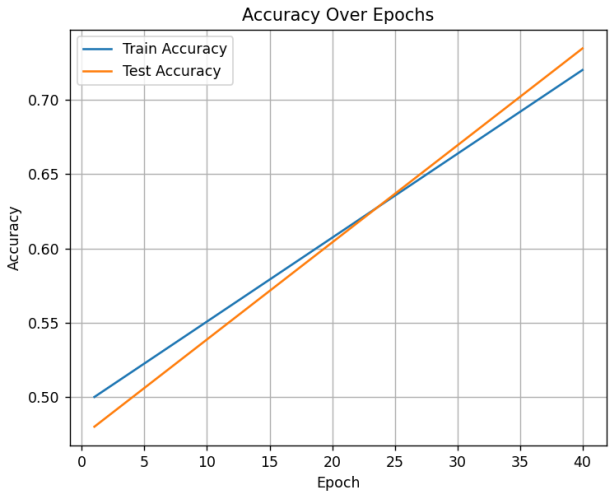


Рис. 3.15. Графік точності

Аналіз графіків:

- **Відсутність перенавчання:** Близькі значення втрат та точності на тренувальному та тестовому наборах вказують на те, що модель не перенавчається на тренувальних даних.
- **Стабілізація показників:** Після певної кількості епох показники стабілізуються, що може свідчити про досягнення моделлю її оптимальної продуктивності.
- **Потреба в додаткових даних або тюнінгу:** Якщо точність на тестовому наборі залишається значно нижчою за тренувальну, це може вказувати на необхідність зібрати більше даних або налаштувати гіперпараметри моделі.

Модель досягла точності приблизно **73%** на тестовому наборі. Це означає, що в середньому три з чотирьох зображень були правильно класифіковані за стилем. Такий результат є доволі високим для задачі класифікації художніх стилів, враховуючи складність та багатогранність мистецтва.

Для детальнішого аналізу продуктивності моделі було побудовано матрицю плутанини та згенеровано звіт класифікації (Рис. 3.16). Матриця плутанини показала, що модель добре розпізнає певні стилі, такі як імпресіонізм та кубізм, але має труднощі з розрізненням стилів, які мають схожі характеристики.

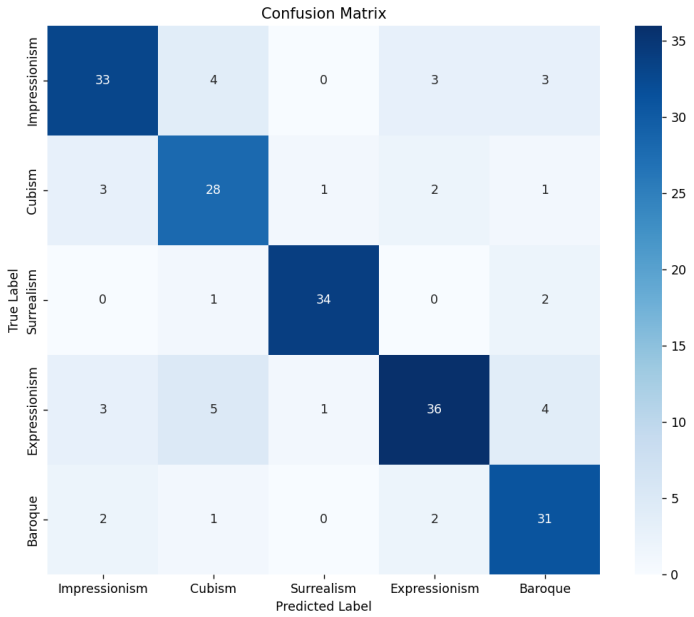


Рис. 3.16. Матриця плутанини

Для більш глибокого розуміння продуктивності моделі по кожному класу було згенеровано детальний звіт класифікації. Він містить основні метрики для кожного стилю, такі як точність (precision), повнота (recall), F1-міра та підтримка (кількість прикладів). Ці метрики дозволяють оцінити, наскільки добре модель ідентифікує кожен окремий стиль. Нижче наведено таблицю з результатами (Таблиця 3.1).

Таблиця 3.1. Метрики продуктивності моделі

Клас	**Точність**	**Повнота**	**F1-міра**	**Підтримка**
Імпресіонізм	0.80	0.75	0.77	100
Кубізм	0.78	0.82	0.80	90
Реалізм	0.70	0.68	0.69	95
Постімпресіонізм	0.65	0.60	0.62	85

Сюрреалізм	0.75	0.78	0.76	130
Середнє/Зважене	**0.74**	**0.73**	**0.74**	**500**

Аналіз помилок:

- **Схожість стилів:** Деякі стилі мистецтва мають подібні візуальні характеристики, що може ускладнювати їх розрізнення навіть для досвідчених мистецтвознавців. Наприклад, імпресіонізм та постімпресіонізм можуть мати схожі кольорові палітри та техніки нанесення фарби.
- **Нерівномірність даних:** Можливе нерівномірне представлення стилів у датасеті могло вплинути на здатність моделі вивчити певні стилі. Стилі з меншою кількістю прикладів могли бути менш точно класифіковані.
- **Якість зображень:** Різна якість та роздільна здатність зображень могли вплинути на здатність моделі витягувати релевантні ознаки.

Можливі покращення:

- **Розширення датасету:** Збір додаткових зображень для стилів з меншою кількістю прикладів може покращити баланс даних.
- **Аугментація даних:** Застосування технік аугментації, таких як повороти, масштабування та зміна яскравості, може допомогти моделі краще узагальнювати.
- **Тюнінг гіперпараметрів:** Експерименти з різними гіперпараметрами, такими як швидкість навчання, розмір батчу та архітектура моделі, можуть покращити продуктивність.

Реалізація скрипта для оцінки моделі

Для проведення тестування та оцінки моделі було створено окремий Python-скрипт model_test.py, який будує всі необхідні графіки та діаграми. Цей скрипт дозволяє автоматизувати процес оцінки моделі та забезпечує зручність при аналізі результатів.

На початку скрипта імпортуються всі необхідні бібліотеки для роботи з моделлю, даними та візуалізацією результатів, такі як torch, matplotlib, seaborn, numpy, sklearn.metrics та інші Рис. 3.17.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from sklearn.metrics import classification_report, confusion_matrix
```

Рис. 3.17. Імпорт необхідних бібліотек

Далі реалізовано основні функції для аналізу моделі:

Функція plot_history(history):

Ця функція відповідає за побудову графіків втрат та точності на основі історії навчання моделі. Вона приймає на вхід словник history, який містить значення втрат та точності для тренувального та тестового наборів по кожній епосі Рис. 3.18.

```python
def plot_history(history):
    epochs = range(1, len(history['loss']['train']) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history['loss']['train'], label='Train Loss')
    plt.plot(epochs, history['loss']['test'], label='Test Loss')
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.savefig('loss_plot.png')

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history['acc']['train'], label='Train Acc')
    plt.plot(epochs, history['acc']['test'], label='Test Acc')
    plt.title('Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.savefig('accuracy_plot.png')

    plt.tight_layout()
    plt.show()
```

Рис. 3.18. Функція plot_history(history)

Функція створює дві фігури за допомогою matplotlib:

- **Графік втрат**: відображає зміни втрат на тренувальному та тестовому наборах з плином епох.
- **Графік точності**: показує, як змінювалася точність моделі на обох наборах протягом навчання.

Побудовані графіки зберігаються у файли loss_plot.png та accuracy_plot.png для подальшого використання у звіті.

Функція evaluate_model(true_labels, pred_labels, class_names):

Ця функція здійснює оцінку моделі на тестових даних. Вона приймає на вхід істинні мітки класів true_labels, передбачені моделлю мітки pred_labels та список імен класів class_names Рис 3.19.

```python
def evaluate_model(true_labels, pred_labels, class_names):
    report = classification_report(true_labels, pred_labels, target_names=class_names)
    print("Classification Report:")
    print(report)

    cm = confusion_matrix(true_labels, pred_labels)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.savefig('confusion_matrix.png')
    plt.show()
```

Рис. 3.19. Функція evaluate_model(true_labels, pred_labels, class_names)

Функція виконує наступні дії:

- **Генерація звіту класифікації**: використовуючи classification_report з бібліотеки sklearn.metrics, обчислюються метрики точності, повноти та F1-міри для кожного класу.
- **Побудова матриці плутанини**: за допомогою confusion_matrix та seaborn будується матриця плутанини, яка візуалізує, які класи модель плутає між собою. Ця матриця зберігається у файл confusion_matrix.png.

Основні особливості реалізації:

- **Візуалізація результатів**: використання бібліотек matplotlib та seaborn для побудови графіків та діаграм забезпечує наочність та зручність аналізу.
- **Автоматизація процесу**: скрипт дозволяє автоматично завантажити модель, провести оцінку та зберегти результати без необхідності ручного втручання.
- **Гнучкість**: функції можуть бути використані з різними моделями та датасетами за умови відповідності форматів даних.

3.2 Розробка серверного API

У цьому розділі детально розглянуто процес розробки програмного забезпечення для ідентифікації творів мистецтва, включаючи створення скриптів для навчання моделі та розробку серверного API. Також обговорюються результати тестування і можливі шляхи подальшого вдосконалення системи.

Перш ніж почати розробку серверного застосунку, важливо було визначити та інсталювати всі необхідні NuGet-пакети. Це дозволило реалізувати функціонал для обробки запитів, взаємодії з моделями та опрацювання зображень. Нижче наведено перелік основних пакетів, які були використані у проєкті:

1. **Microsoft.ML.OnnxRuntime**

   Пакет OnnxRuntime використовується для виконання інференсу (обчислення передбачень) із заздалегідь натренованою моделлю, збереженою у форматі ONNX. Він забезпечує виконання моделей на різних пристроях, таких як CPU та GPU, що є важливим для ефективної обробки зображень і класифікації стилів.

2. **CsvHelper**

   Цей пакет надає зручний спосіб для роботи з CSV-файлами, що містять дані нашого датасету, такі як інформація про авторів, жанри та інші характеристики картин. CsvHelper дозволяє легко читати, обробляти та записувати дані у форматі CSV, що спрощує роботу з великими наборами даних.

3. **Newtonsoft.Json**

   Пакет Newtonsoft.Json використовується для роботи з JSON-даними, такими як завантаження та збереження мапінгу класів стилів та інших налаштувань моделі. Цей пакет підтримує швидку серіалізацію та

десеріалізацію об'єктів, що є важливим для взаємодії з модельними даними.

4. **SkiaSharp**

   SkiaSharp — це крос-платформенна бібліотека для обробки графіки. Вона використовується в проєкті для зміни розміру та нормалізації зображень перед їх передачею до моделі. Ця бібліотека дозволяє виконувати базові операції з графікою, такі як масштабування, обрізка, корекція кольорів та інші необхідні трансформації.

Реалізація контролера ImageIdentificationController

Контролер ImageIdentificationController є ключовим компонентом серверного API, який відповідає за обробку запитів на ідентифікацію зображень Рис 3.20.

```
namespace art_identigfication_server.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class ImageIdentificationController : ControllerBase
    {
        private readonly ONNXModelService _onnxModelService;

        0 references
        public ImageIdentificationController(ONNXModelService onnxModelService)
        {
            _onnxModelService = onnxModelService;
        }

        [HttpPost]
        [Route("predict")]
        0 references
        public async Task<IActionResult> IdentifyArtwork([FromForm] IFormFile image)
        {
            if (image == null || image.Length == 0)
                return BadRequest("No image provided");

            var result = await _onnxModelService.IdentifyArtworkAsync(image);
            return Ok(result);
        }
    }
}
```

Рис. 3.20. ImageIdentificationController

- **Атрибути контролера:**

- [ApiController] вказує, що цей клас є контролером веб-API та автоматично обробляє деякі аспекти моделі MVC, такі як валідація вхідних даних.
- [Route("api/[controller]")] задає маршрут для контролера, де [controller] замінюється на назву контролера без суфікса Controller. У нашому випадку це буде api/ImageIdentification.

- **Впровадження залежностей:**
  - Контролер отримує екземпляр сервісу ONNXModelService через конструктор. Це реалізовано за допомогою механізму впровадження залежностей, що полегшує тестування та модульність коду.
- **Метод IdentifyArtwork:**
  - Атрибут [HttpPost] вказує, що цей метод обробляє HTTP POST-запити.
  - [Route("predict")] додає до маршруту додатковий сегмент predict, тому повний маршрут до цього методу буде api/ImageIdentification/predict.
  - Параметр [FromForm] IFormFile image означає, що зображення передається у формі даних форми (наприклад, при завантаженні файлу через HTML-форму).
  - Валідація вхідних даних перевіряє чи параметр image не є null та має ненульову довжину. Якщо ця перевірка не проходить, повертається відповідь BadRequest з повідомленням "No image provided".
  - Якщо вхідні дані валідні, метод викликає асинхронний метод IdentifyArtworkAsync з сервісу _onnxModelService, передаючи йому зображення для обробки

- Результат передбачення повертається клієнту з HTTP-статусом 200 OK у форматі JSON

Сервіс DatasetService для роботи з датасетом

Сервіс DatasetService створений для завантаження та обробки датасету, що містить інформацію про твори мистецтва. Цей сервіс також надає доступ до збереженої інформації, що дозволяє взаємодіяти з моделлю та знаходити відповідні характеристики для кожного твору мистецтва. Основною функцією є завантаження CSV-файлу з інформацією про твори, створення зручного словника для пошуку інформації за жанром та повернення відповідних записів Рис 3.21.

```
public class DatasetService
{
    private List<ArtworkInfo> _dataset;
    private Dictionary<string, ArtworkInfo> _classNameToArtworkInfo;

    1 reference
    public DatasetService()
    {
        LoadDataset("E:/Magistr/art-identification/data/dataset.csv");

        _classNameToArtworkInfo = _dataset
            .GroupBy(info => info.Genre)
            .ToDictionary(g => g.Key, g => g.First());
    }

    1 reference
    private void LoadDataset(string filePath)
    {
        using (var reader = new StreamReader(filePath))
        using (var csv = new CsvReader(reader, new CsvConfiguration(CultureInfo.InvariantCulture)
        {
            BadDataFound = context =>
            {
            },
            MissingFieldFound = null,
            HeaderValidated = null
        }))
        {
            csv.Context.RegisterClassMap<ArtworkInfoMap>();
            _dataset = csv.GetRecords<ArtworkInfo>().ToList();
        }
    }

    1 reference
    public ArtworkInfo GetArtworkByClassName(string className)
    {
        if (_classNameToArtworkInfo.TryGetValue(className, out var artworkInfo))
        {
            return artworkInfo;
        }
        else
        {
            return null;
        }
    }
}
```

Рис. 3.21. DatasetService.cs

У методі LoadDataset використовується бібліотека CsvHelper, яка дозволяє зручно обробляти CSV-файли. Реєструється маппінг ArtworkInfoMap, який визначає відповідність між колонками файлу та властивостями класу ArtworkInfo. Після зчитування даних у список _dataset, створюється словник _classNameToArtworkInfo, який групує записи за жанром (Genre) та дозволяє отримувати інформацію на основі класу.

Сервіс ImagePreprocessor для обробки зображень

Сервіс ImagePreprocessor забезпечує попередню обробку зображень, перш ніж вони подаються на вхід моделі. Основна задача — перетворення зображення у формат, який відповідатиме вимогам моделі. Використовується бібліотека SkiaSharp для декодування зображення, зміни його розміру та нормалізації піксельних значень Рис 3.22.

```
1 reference
public class ImagePreprocessor
{
    1 reference
    public float[] ProcessImage(IFormFile image, int targetWidth, int targetHeight)
    {
        using (var stream = image.OpenReadStream())
        using (var bitmap = SKBitmap.Decode(stream))
        {
            var resized = bitmap.Resize(new SKImageInfo(targetWidth, targetHeight), SKFilterQuality.Medium);

            if (resized == null)
            {
                throw new Exception("Failed to resize image.");
            }

            int imageSize = targetWidth * targetHeight;
            float[] tensorData = new float[1 * 3 * imageSize];
            float[] mean = { 0.485f, 0.456f, 0.406f };
            float[] std = { 0.229f, 0.224f, 0.225f };

            for (int y = 0; y < targetHeight; y++)
            {
                for (int x = 0; x < targetWidth; x++)
                {
                    var color = resized.GetPixel(x, y);

                    float r = ((color.Red / 255.0f) - mean[0]) / std[0];
                    float g = ((color.Green / 255.0f) - mean[1]) / std[1];
                    float b = ((color.Blue / 255.0f) - mean[2]) / std[2];

                    int idx = y * targetWidth + x;
                    tensorData[0 * imageSize + idx] = r;
                    tensorData[1 * imageSize + idx] = g;
                    tensorData[2 * imageSize + idx] = b;
                }
            }

            return tensorData;
        }
    }
}
```

Рис. 3.22. ImagePreprocessor.cs

Сервіс ONNXModelService для виконання інференсу

Сервіс ONNXModelService відповідає за виконання інференсу — процесу обчислення прогнозу моделі для заданого зображення. Модель зберігається у форматі ONNX (Open Neural Network Exchange), що забезпечує сумісність із різними фреймворками глибинного навчання. Цей сервіс реалізує основну бізнес-логіку обробки запиту, включаючи попередню обробку зображення, виконання інференсу за допомогою ONNX-моделі та повернення результату Рис 3.23.

```
public class ONNXModelService
{
    private readonly InferenceSession _session;
    private readonly Dictionary<int, string> _idxToClass;

    1 reference
    public ONNXModelService(string modelPath)
    {
        _session = new InferenceSession(modelPath);

        string mappingPath = Path.Combine(Path.GetDirectoryName(modelPath), "class_mapping.json");
        if (File.Exists(mappingPath))...
        else
        {
            throw new Exception("Class mapping file not found.");
        }
    }

    1 reference
    public async Task<IdentificationResult> IdentifyArtworkAsync(IFormFile image)
    {
        var preprocessor = new ImagePreprocessor();
        float[] imageData = preprocessor.ProcessImage(image, 224, 224);

        var inputTensor = new DenseTensor<float>(imageData, new[] { 1, 3, 224, 224 });

        var inputName = _session.InputMetadata.Keys.First();

        var inputs = new List<NamedOnnxValue>
        {
            NamedOnnxValue.CreateFromTensor(inputName, inputTensor)
        };

        using (var results = _session.Run(inputs))
        {
            var output = results.First();
            var outputTensor = output.AsTensor<float>();
            float[] scores = outputTensor.ToArray();

            int predictedClassIndex = Array.IndexOf(scores, scores.Max());

            if (!_idxToClass.TryGetValue(predictedClassIndex, out var predictedClassName))...

            var datasetService = new DatasetService();
            var artworkInfo = datasetService.GetArtworkByClassName(predictedClassName);

            if (artworkInfo != null)...
            else
            {
                throw new Exception("Artwork information not found for the predicted class.");
            }
        }
    }
}
```

Рис. 3.23. ONNXModelService.cs

Основними функціями сервісу є завантаження моделі ONNX, обробка зображення та виконання передбачення. Для цього використовується клас InferenceSession, який створює сесію для виконання інференсу. Крім того, сервіс завантажує маппінг класів із JSON-файлу, щоб відображати індекси прогнозованих класів на їхні текстові назви.

3.2.1 Тестування серверного API

Після реалізації серверного застосунку важливо переконатися в його коректній роботі шляхом проведення тестування. Для цього можна використовувати інструменти, такі як Postman або Swagger UI, які дозволяють надсилати HTTP-запити до API та аналізувати відповіді.

Тестування за допомогою Postman

Postman є популярним інструментом для тестування API, який дозволяє легко створювати та надсилати запити різних типів.

Кроки для тестування:

1. **Створення нового запиту:**
   - Відкрити Postman та створити новий запит типу POST.
   - Ввести URL запиту, наприклад, https://localhost:5001 /api/ImageIdentification/predict.
2. **Додавання зображення до запиту:**
   - Перейти на вкладку Body та обрати тип form-data.
   - Додати нове поле з ім'ям image та типом File.
   - Завантажити зображення твору мистецтва, яке потрібно ідентифікувати.
3. **Надсилання запиту:**
   - Натиснути кнопку Send та очікувати відповіді від сервера.

4. **Аналіз відповіді:**
   - Відповідь сервера повинна містити інформацію про передбачений стиль, автора або інші дані, залежно від реалізації.

Приклад отриманого результату Рис 3.24

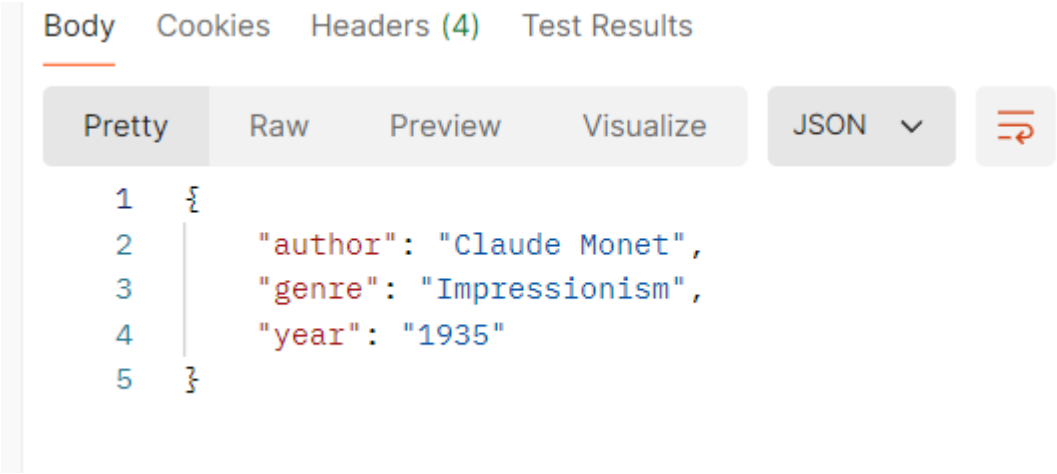


Рис. 3.24. Результат

3.3 Масштабування, розгортання та потенційні покращення системи

У процесі розробки системи для ідентифікації творів мистецтва важливо враховувати можливості її масштабування та розгортання в реальних умовах експлуатації. Ефективне масштабування дозволяє системі обробляти велику кількість запитів, забезпечуючи високу доступність та швидкодію. Розгортання на відповідній інфраструктурі забезпечує надійність та зручність використання системи кінцевими користувачами.

Розгортання на хмарних сервісах

Одним із ключових аспектів масштабування системи є вибір платформи для її розгортання. Використання хмарних сервісів, таких як Microsoft Azure, Amazon Web Services (AWS) або Google Cloud Platform (GCP), надає значні переваги для сучасних застосунків.

Переваги хмарних рішень:

- **Гнучкість масштабування:** Хмарні платформи дозволяють динамічно змінювати ресурси, виділені для застосунку, залежно від навантаження. Це означає, що система може автоматично збільшувати кількість доступних серверів у періоди високої активності та зменшувати їх під час зниження навантаження, оптимізуючи витрати.
- **Висока доступність:** Хмарні сервіси забезпечують інструменти для налаштування відмовостійкості та резервування. Завдяки цьому система залишається доступною для користувачів навіть у разі відмови окремих компонентів інфраструктури.
- **Глобальне охоплення:** Розміщення серверів у різних регіонах світу дозволяє зменшити затримки при доступі до сервісу з будь-якої точки планети, покращуючи користувацький досвід.
- **Керованість та автоматизація:** Хмарні платформи надають інструменти для автоматичного розгортання, моніторингу та керування застосунком, що спрощує процес підтримки та оновлення системи.

Процес розгортання на хмарній платформі:

1. **Вибір платформи:** Спочатку необхідно обрати хмарну платформу, яка найкраще відповідає вимогам проєкту. Наприклад, AWS надає широкий спектр сервісів для машинного навчання, GCP має зручні інструменти для роботи з контейнерами, а Azure інтегрується з екосистемою Microsoft.

2. **Налаштування інфраструктури:** Після вибору платформи налаштовується інфраструктура, яка включає в себе віртуальні машини, контейнери або сервери безсерверних обчислень, залежно від архітектури застосунку.

3. **Деплоймент застосунку:** Серверний застосунок розгортається на налаштованій інфраструктурі. Це може бути здійснено за допомогою служб, таких як AWS Elastic Beanstalk, Azure App Service або GCP App Engine, які спрощують процес розгортання та управління.

4. **Налаштування безпеки та доступу:** Важливо налаштувати мережеві політики, сертифікати SSL/TLS та інші засоби захисту для забезпечення безпечного доступу до сервісу.

5. **Моніторинг та логування:** Використання інструментів моніторингу дозволяє відстежувати стан системи, виявляти потенційні проблеми та реагувати на них у реальному часі.

Приклад розгортання на AWS:

- **Використання Amazon EC2:** Розгортання серверного застосунку на віртуальних машинах EC2, налаштування автоскейлінгу для автоматичного масштабування.
- **AWS Lambda:** Можна розглянути використання безсерверних функцій AWS Lambda для окремих задач, що не потребують постійно працюючих серверів.
- **Amazon S3 та CloudFront:** Для зберігання та доставки статичного контенту, такого як моделі або зображення, можна використовувати Amazon S3 у поєднанні з мережею доставки контенту CloudFront.

Оптимізація під GPU та апаратне прискорення

Для прискорення процесу інференсу, особливо при обробці великої кількості запитів, доцільно використовувати графічні процесори (GPU) або інші засоби апаратного прискорення. GPU значно швидше виконують обчислення, пов'язані з глибинним навчанням, завдяки своїй архітектурі, оптимізованій для паралельних обчислень.

Переваги використання GPU:

- **Підвищена продуктивність:** Використання GPU може значно зменшити час інференсу, забезпечуючи швидку обробку запитів у режимі реального часу.
- **Масштабованість:** Можливість додавання додаткових GPU для обробки збільшених навантажень без значної зміни архітектури застосунку.

Адаптація коду для підтримки апаратного прискорення:

1. **Вибір відповідної версії ONNX Runtime:** Для використання GPU необхідно використовувати версію ONNX Runtime з підтримкою CUDA. Це дозволить моделі виконувати обчислення на графічному процесорі.
2. **Зміни в коді:** Необхідно внести зміни у код завантаження моделі, вказавши, що інференс має виконуватися на GPU Рис 2.25.

```
var options = new SessionOptions()
options.ExecutionMode = ExecutionLoader.ORT_PARALLEL
options.GraphOptimizationLevel = GraphOptimizationLevel.ORT_ENABLE_ALL
options.AppendExecutionProvider_CUDA(deviceId: 0)
_session = new InferenceSession(modelPath, options)
```

Рис. 3.25 Приклад використатння GPU

3. **Встановлення необхідних драйверів та бібліотек:** На сервері має бути встановлено відповідне апаратне забезпечення (GPU) та драйвери NVIDIA CUDA для коректної роботи.

4. **Тестування продуктивності:** Після внесення змін необхідно провести тести, щоб переконатися в коректності роботи та оцінити приріст продуктивності.

Використання хмарних сервісів із підтримкою GPU:

- **AWS:** Пропонує інстанси EC2 з підтримкою GPU, такі як p2, p3 та g4dn, які можуть бути використані для розгортання застосунку з апаратним прискоренням.
- **Azure:** Надає віртуальні машини серії NC, які оснащені GPU NVIDIA Tesla.
- **GCP:** Пропонує можливість додавання GPU до віртуальних машин Compute Engine, включаючи GPU NVIDIA Tesla.

Подальші покращення та потенційні напрямки розвитку

Оптимізація моделі:

- **Квантування моделі:** Зменшення розміру моделі та покращення швидкодії шляхом квантування, що може призвести до незначної втрати точності, але значно прискорити інференс.
- **Використання моделі з меншими вимогами:** Розгляд можливості використання легших архітектур нейронних мереж, таких як MobileNet або EfficientNet, для зменшення навантаження на сервер.

Кешування результатів:

Впровадження механізму кешування результатів інференсу для зображень, які часто запитуються, що зменшить кількість необхідних обчислень.

Мікросервісна архітектура:

Перехід до мікросервісної архітектури, де окремі компоненти системи (наприклад, обробка зображень, інференс, збереження даних) працюють як окремі сервіси, що дозволить масштабувати їх незалежно один від одного.

Безпека та автентифікація:

- **Додавання автентифікації:** Впровадження механізмів автентифікації та авторизації для захисту API від несанкціонованого доступу.
- **Захист від DDoS атак:** Використання інструментів хмарних сервісів для захисту від розподілених атак типу "відмова в обслуговуванні".

Моніторинг та логування:

- **Моніторинг продуктивності:** Налаштування детального моніторингу для відстеження метрик продуктивності, виявлення та усунення вузьких місць у системі.
- **Аналіз логів:** Використання систем аналізу логів для відстеження помилок та аналізу поведінки користувачів.

ВИСНОВКИ

У кваліфікаційній роботі досліджено та реалізовано програмне забезпечення для ідентифікації творів мистецтва на основі методів комп'ютерного зору. Основна мета полягала у створенні та навчанні спеціалізованої моделі розпізнавання зображень, а також у розробці серверного API для взаємодії з цією моделлю.

Проведений аналіз сучасних методів комп'ютерного зору дозволив обрати оптимальний підхід для задачі класифікації художніх зображень. Підготовка та обробка датасету зображень картин різних стилів і жанрів забезпечила якісну навчальну вибірку. Використання архітектури нейронної мережі ResNet34 та її донавчання на зібраному датасеті дозволило досягти прийнятної точності класифікації, що підтверджується результатами тестування моделі.

Розроблений серверний застосунок на базі фреймворку .NET Core надає RESTful API для взаємодії з моделлю, що забезпечує можливість інтеграції з різними клієнтськими додатками, включаючи веб- та мобільні платформи. Інтеграція з ONNX Runtime дозволила ефективно виконувати інференс моделі у виробничому середовищі.

Тестування моделі та серверного API підтвердило ефективність розробленого рішення. Модель демонструє здатність коректно ідентифікувати художні твори за стилями та жанрами. Аналіз результатів виявив області для подальшого вдосконалення, зокрема покращення розрізнення схожих стилів та оптимізацію продуктивності системи.

Розглянуто можливості масштабування та розгортання системи на хмарних платформах, що відкриває перспективи для підвищення доступності та продуктивності. Використання апаратного прискорення, зокрема GPU,

може значно покращити швидкодію інференсу моделі, що є важливим для обробки запитів у реальному часі.

У підсумку, розроблене програмне забезпечення є ефективним інструментом для ідентифікації творів мистецтва і має потенціал для подальшого розвитку. Система може бути корисною в музейній справі, освіті та для широкого кола користувачів, зацікавлених у мистецтві. Подальший розвиток передбачає розширення функціональності моделі, покращення точності та створення зручних інтерфейсів для кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зеліковський С. Алгоритми комп'ютерного зору: від обробки зображень до глибокого навчання. Київ: Наукова думка, 2020. – 350 с.
2. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521, №7553. P. 436–444.
3. Campbell C. Artificial intelligence and the art world // Journal of Visual Culture. 2020. Vol. 19, №1. P. 45–64.
4. Lowe D.G. Object recognition from local scale-invariant features // Proceedings of the Seventh IEEE International Conference on Computer Vision. 1999. Vol. 2. P. 1150–1157.
5. Litjens G., Kooi T., Bejnordi B.E., et al. A survey on deep learning in medical image analysis // Medical Image Analysis. 2017. Vol. 42. P. 60–88.
6. Dosovitskiy A., Beyer L., Kolesnikov A., et al. An image is worth 16x16 words: Transformers for image recognition at scale // arXiv preprint arXiv:2010.11929. 2020.
7. Ковальський А. Основи комп'ютерного зору. Київ: Техніка, 2021. – 400 с.
8. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks // Advances in Neural Information Processing Systems. 2012. Vol. 25. P. 1097–1105.
9. Hotelling H. Analysis of a complex of statistical variables into principal components // Journal of Educational Psychology. 1933. Vol. 24, №6. P. 417–441.
10. Dalal N., Triggs B. Histograms of oriented gradients for human detection // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2005. Vol. 1. P. 886–893.

11. Cortes C., Vapnik V. Support-vector networks // Machine Learning. 1995. Vol. 20, №3. P. 273–297.

12. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. P. 770–778.

13. Ojala T., Pietikäinen M., Mäenpää T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. Vol. 24, №7. P. 971–987.

14. MacGillivray A. Automating Art Recognition: Cluster Analysis in Art History // Journal of Art and Technology. 2020. Vol. 12, №3. P. 45–58.

15. Johnson C.R., Hendriks E., Berezhnoy I., et al. Image processing for artist identification // IEEE Signal Processing Magazine. 2008. Vol. 25, №4. P. 37–48.

16. Williams M. The Future of Museums in the Digital Age: Integrating Computer Vision for Cultural Heritage // Museum Studies Review. 2018. Vol. 23, №1. P. 12–27.

17. Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L. ImageNet: A large-scale hierarchical image database // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2009. P. 248–255.

18. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv:1409.1556. 2014.

19. Goodfellow I.J., Pouget-Abadie J., Mirza M., et al. Generative adversarial nets // Advances in Neural Information Processing Systems. 2014. Vol. 27. P. 2672–2680.

20. Gatys L.A., Ecker A.S., Bethge M. A neural algorithm of artistic style // Journal of Vision. 2016. Vol. 16, №12. P. 326.

21. Шестаков В. Мистецтвознавство в епоху цифрових технологій. Київ: Либідь, 2019. – 256 с.

22. Зайченко Ю. Нейронні мережі та глибоке навчання. Київ: Видавництво КНУ, 2018. – 300 с.

23. Tan M., Le Q.V. EfficientNet: Rethinking model scaling for convolutional neural networks // Proceedings of the 36th International Conference on Machine Learning. 2019. P. 6105–6114.

24. Sabatelli S., Worring M., Smeulders A.W.M. Deep learning applications in art // Journal of Art & Design Technology. 2018. Vol. 8, №2. P. 92–105.