НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра інформатики та програмного забезпечення

(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з _____”Обробка надвеликих масивів даних”

_____

(назва дисципліни)

на тему: Класифікація дорожніх знаків за допомогою Apache Spark

_____

Студента__1___ курсу _ІП-41мн____ групи
Спеціальності 121_Інженерія програмного забезпечення
Пашковського Є.С.

(прізвище та ініціали)

Керівник: доцент каф. ІПІ, к.т.н.,
Олійник Ю.О.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка _____
Кількість балів: _____Оцінка: ECTS __

Члени комісії

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2024 рік

Національний технічний університет України “КПІ імені Ігоря Сікорського”

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна

«Оброблення надвеликих масивів даних»

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 1 Група ІП-41мн Семестр 1

ЗАВДАННЯ

на курсову роботу студента

Пашковського Євгенія Сергійовича

(прізвище, ім’я, по батькові)

1. Тема роботи Класифікація дорожніх знаків за допомогою Apache Spark

2. Строк здачі студентом закінченої роботи 22.12.2024

3. Вихідні дані до роботи Датасет зображень дорожніх знаків з метаданими

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Постановка задачі
Розробка процесів підготовки даних, ETL процесів, проектування структури БД
Опис методів обробки даних
Дослідження ефективності методів обробки даних

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Дата видачі завдання 12.11.2024

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	12.11.2024	
2.	Визначення основних задач курсової роботи	21.11.2024	
3.	Пошук та вивчення літератури з питань курсової роботи	21.11.2024	
4.	Розробка процесу попередньої обробки даних, ETL процесів, моделі БД	27.11.2024	
5.	Розробка методів обробки даних	05.12.2024	
6.	Дослідження ефективності методів обробки даних	12.12.2024	
7.	Підготовка пояснювальної записки	18.12.2024	
8.	Здача курсової роботи на перевірку	19.12.2024	
9.	Захист курсової роботи	23.12.2024	

Студент Пашковський Є.С.

(підпис) (прізвище, ім'я, по батькові)

Керівник Олійник Ю. О.

(підпис) (прізвище, ім'я, по батькові)

"___" _____ 2024 р.

АНОТАЦІЯ

Пояснювальна записка курсового проєкту складається з чотирьох розділів, містить 4 таблиць, 5 рисунків та 6 джерел – загалом 36 сторінок.

Курсовий проєкт присвячений розв’язанню задачі класифікації дорожніх знаків.

Мета: покращення точності та швидкості обробки великих масивів даних у контексті класифікації зображень дорожніх знаків за допомогою технології Apache Spark.

У розділі “Розробка процесів підготовки даних, ETL-процесів, проєктування структури бази даних” розглянуто процеси підготовки даних для класифікації, включаючи нормалізацію, обрізання областей інтересу, перетворення зображень до потрібного формату, а також спроєктовано структуру бази даних для збереження метаданих і результатів.

У розділі “Опис методів обробки даних” проаналізовано стандартні методи MLlib, такі як логістична регресія та рандомний ліс, а також методи глибокого навчання з використанням TensorFlow. Наведено опис їхніх алгоритмів роботи, переваги, недоліки та особливості застосування.

У розділі “Дослідження ефективності методів обробки даних” проведено порівняльний аналіз обраних методів за критеріями точності, швидкості навчання та класифікації. Показано, що глибинні нейронні мережі на основі TensorFlow досягли найкращих результатів у задачах класифікації зображень, зокрема завдяки використанню GPU для прискорення обчислень.

У розділі “Загальні висновки” узагальнено результати дослідження, наведено рекомендації щодо вибору методів обробки даних залежно від доступних ресурсів та задач.

Ключові слова: класифікація зображень, дорожні знаки, Apache Spark, TensorFlow, MLlib, глибинне навчання.

ЗМІСТ

ВСТУП

Сучасний світ характеризується безпрецедентним зростанням обсягів даних, зокрема цифрових зображень. В умовах цифровізації різних сфер життя, такі масиви даних є важливим джерелом інформації для багатьох галузей, включаючи медицину, автомобільну промисловість, маркетинг, безпеку та розробку штучного інтелекту. Однією з ключових задач, що виникають при роботі з такими даними, є класифікація зображень, яка дозволяє автоматично визначати категорії або класи для конкретних візуальних об’єктів. Особливий інтерес становить класифікація дорожніх знаків, що є важливою складовою систем автономного керування транспортними засобами.

Впровадження глибоких нейронних мереж, таких як згорткові нейронні мережі (CNN), відкрило нові можливості для точного і швидкого аналізу великих обсягів візуальної інформації. Для ефективної роботи з надвеликими масивами даних використовуються технології розподіленого обчислення, зокрема Apache Spark, який дозволяє масштабувати процеси обробки та аналізу даних.

У рамках даної курсової роботи розглядається задача класифікації зображень дорожніх знаків в умовах надвеликих масивів даних, аналізуються існуючі методи та підходи. Основна увага приділяється використанню сучасних алгоритмів машинного навчання, зокрема глибоких нейронних мереж, у поєднанні з платформою Apache Spark для ефективної обробки великих обсягів даних.

Актуальність роботи обумовлена зростаючими вимогами до точності та швидкості обробки інформації в системах автономного водіння.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Предметна область

Розпізнавання та класифікація дорожніх знаків є однією з ключових задач у сфері комп’ютерного зору. Це завдання відіграє вирішальну роль у створенні автономних транспортних засобів, забезпечуючи безпеку руху, правильне інтерпретування сигналів дорожньої інфраструктури та своєчасне реагування на зміну ситуації на дорозі. Дорожні знаки передають важливу інформацію, що регулює рух, попереджає про можливі небезпеки або надає вказівки водіям. Їх класифікація повинна бути швидкою та точною, аби відповідати вимогам реального часу [1].

Складність задачі класифікації дорожніх знаків зумовлена кількома факторами [2]:

- різноманітність умов зйомки (зміна освітлення, погода, якість камери та її позиціювання, що можуть суттєво вплинути на вигляд знаків);
- досить велика кількість класів (дорожні знаки відрізняються за формою, кольором, текстом і символами, що утворює складну багатокласову проблему);
- шум та перешкоди (знаки можуть бути частково закритими іншими об’єктами, пошкодженими або відображеними у вигляді відбитків);
- реальні часові обмеження (системи повинні працювати в режимі реального часу, обробляючи велику кількість кадрів із відеопотоків).

Для ефективного вирішення цієї задачі у великих масивах даних необхідно використовувати сучасні технології, такі як платформи розподілених обчислень. Apache Spark забезпечує можливість масштабованої обробки великих обсягів даних завдяки паралельним обчисленням та гнучкому інструментарію для обробки, аналізу та машинного навчання. Apache Spark

дозволяє ефективно класифікувати дорожні знаки, враховуючи особливості великих даних [3].

У наступних розділах буде детально описано постановку задачі, вибір методів для вирішення, а також розглянуто експериментальну частину роботи.

1.2 Задача

Метою даної роботи є покращення точності та швидкості обробки великих масивів даних у контексті класифікації зображень дорожніх знаків за допомогою технології Apache Spark.

Для досягнення цієї мети було визначено такі завдання:

- аналіз існуючих методів класифікації дорожніх знаків, їх переваг та недоліків;
- розробка моделей класифікації дорожніх знаків, здатних обробляти зображення дорожніх знаків у потоковому режимі;
- інтеграція розробленої моделі з платформою Apache Spark для забезпечення масштабованості обробки даних;
- проведення експериментів для оцінки точності, швидкості та ефективності запропонованих підходів.

Ця робота спрямована на розв’язання актуальних проблем класифікації дорожніх знаків у реальних умовах, де обсяги даних та вимоги до швидкості обробки є критично важливими.

1.3 Вимоги до програмного забезпечення

Програмне забезпечення має бути представлене у вигляді IPython Notebook з розширенням “.ipynb”. Програмне забезпечення має бути написано мовою Python.

1.3.1 Функціональні вимоги

1.3.1.1 Обробка зображень

1.3.1.1.1 Виконання нормалізації, обрізання, зміни розміру зображень та перетворення у вектор ознак.

1.3.1.1.2 Класифікація дорожніх знаків з використанням моделей машинного навчання.

1.3.1.2 Побудова моделей

1.3.1.2.1 Побудова класифікаційних моделей з використанням бібліотек MLib та TensorFlow.

1.3.1.3 Інтеграція

1.3.1.3.1 Інтеграція обробки даних з платформою Apache Spark.

1.3.1.3.2 Можливість потокової обробки зображень у реальному часі.

1.3.1.4 Візуалізація результатів

1.3.1.4.1 Представлення результатів класифікації у вигляді графіків та таблиць.

1.3.2 Нефункціональні вимоги

1.3.2.1 Масштабованість, обробка великих обсягів даних (тисячі зображень) з можливістю масштабування на кілька вузлів.

1.3.2.2 Продуктивність, забезпечення швидкої обробки даних для відповідності реальним часовим обмеженням.

1.3.2.3 Надійність, підтримка логування та моніторингу для виявлення та усунення помилок у процесі виконання.

1.3.2.4 Сумісність, підтримка бібліотек машинного навчання (наприклад, TensorFlow) та платформ обробки великих даних (Apache Spark)

1.3.2.5 Гнучкість, можливість налаштування параметрів системи, таких як розмір партії даних та кількість вузлів обробки.

Відповідність цим вимогам гарантуватиме ефективність, точність та стабільність роботи системи класифікації дорожніх знаків у рамках надвеликих масивах даних.

1.4 Висновки до розділу

У даному розділі визначено предметну область, сформульовано основну задачу та описано вимоги до програмного забезпечення. Цей розділ створює базу для подальшого проєктування та реалізації рішень, спрямованих на досягнення високої точності класифікації за реальних умов експлуатації. Підхід, описаний у даній роботі, забезпечує інтеграцію сучасних методів машинного навчання з платформою Apache Spark, що дозволяє обробляти великі обсяги даних у масштабованому середовищі. Усі ці аспекти створюють передумови для подальшого дослідження та удосконалення методів аналізу даних та їхнього впровадження у практичні застосування.

2 РОЗРОБКА ПРОЦЕСІВ ПІДГОТОВКИ ДАНИХ, ETL ПРОЦЕСІВ, ПРОЕКТУВАННЯ СТРУКТУРИ БД

2.1 Налаштування середовища

Було розгорнуто локальний кластер Apache Spark для забезпечення масштабованої обробки даних. Кластер працює у середовищі Windows Subsystem for Linux (WSL), що дозволяє використовувати можливості Linux у поєднанні з перевагами операційної системи Windows 11.

Для розробки програмного забезпечення обрано редактор коду Visual Studio Code завдяки його широкій підтримці плагінів, інтеграції з системами контролю версій та зручному інтерфейсу. Програмне забезпечення написано мовою Python, що забезпечує гнучкість та великий вибір бібліотек для обробки даних та машинного навчання. Для виконання коду було встановлено деякі необхідні бібліотеки, серед яких:

- Spark – для обробки великих обсягів даних;
- TensorFlow – для реалізації алгоритмів машинного навчання;
- Pandas – для роботи з табличними даними;
- matplotlib – для візуалізації даних;
- sklearn – для зручної побудови confusion matrix;
- PIL – для обробки зображень.

Характеристики середовища виконання:

- Intel Core i7-12700KF 3.6-5GHz
- Nvidia RTX 3080 10Gb
- 32GB RAM 4800MHz CL40
- ОС Windows 11 with WSL 2.0 Ubuntu 22.04
- SATA3 Disk with 240Gb free

Ці параметри забезпечують високу продуктивність та дозволяють обробляти великі обсяги даних у режимі реального часу. Особливо це стосується відеокарти з підтримкою технології CUDA, за допомогою якої можна відносно швидко тренувати моделі за допомогою TensorFlow.

2.2 Процес підготовки даних

Перед тим як навчати модель, необхідно виконати низку етапів підготовки даних, що включають:

– завантаження метаданих зображень, зокрема, шлях до файлу, координати області інтересу (ROI) та клас зображення (вигляд метаданих показано на рисунку 2.1);

Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
27	26	5	5	22	20	20	./dataset/Train/2...
28	27	5	6	23	22	20	./dataset/Train/2...
29	26	6	5	24	21	20	./dataset/Train/2...
28	27	5	6	23	22	20	./dataset/Train/2...
28	26	5	5	23	21	20	./dataset/Train/2...
31	27	6	5	26	22	20	./dataset/Train/2...
31	28	6	6	26	23	20	./dataset/Train/2...
31	28	6	6	26	23	20	./dataset/Train/2...

Рисунок 2.1 – Метадані зображень

– обрізання області інтересу до потрібної ділянки, що містить дорожній знак, використовуючи координати ROI;
– нормалізація та зміна розміру зображення (було обрано розширення 32x32 пікселі);
– трансформувати зображення до одновимірного масиву ознак.

Обробка зображень здійснюється за допомогою спеціально розроблених UDF (User-Defined Functions), які інтегруються у процес обробки даних на платформі Apache Spark [4]. Це дозволяє виконувати підготовку даних у розподіленому середовищі з мінімальними витратами часу. Приклад обробленого зображення можна побачити на рисунку 2.2.

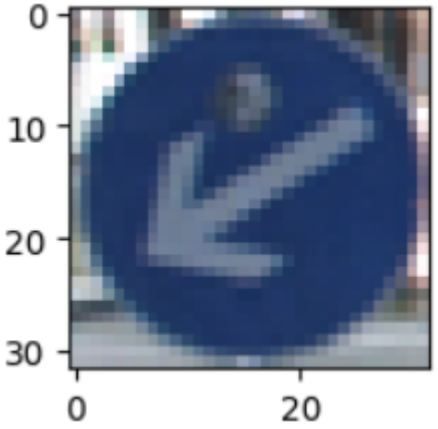

Рисунок 2.2 – Приклад зображення після обробки

Оновлений вигляд датафрейму після обробки показаний на рисунку 2.3.

Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path	features
27	26	5	5	22	20	20	./dataset/Train/2...	[0.2,0.1960784313...
28	27	5	6	23	22	20	./dataset/Train/2...	[0.19215686274509...
29	26	6	5	24	21	20	./dataset/Train/2...	[0.19607843137254...
28	27	5	6	23	22	20	./dataset/Train/2...	[0.19215686274509...
28	26	5	5	23	21	20	./dataset/Train/2...	[0.18823529411764...
31	27	6	5	26	22	20	./dataset/Train/2...	[0.18823529411764...
31	28	6	6	26	23	20	./dataset/Train/2...	[0.18823529411764...
31	28	6	6	26	23	20	./dataset/Train/2...	[0.18431372549019...
31	29	5	6	26	24	20	./dataset/Train/2...	[0.17647058823529...
34	32	6	6	29	26	20	./dataset/Train/2...	[0.18039215686274...
36	33	5	6	31	28	20	./dataset/Train/2...	[0.17254901960784...
37	34	5	6	32	29	20	./dataset/Train/2...	[0.17647058823529...
38	34	5	6	32	29	20	./dataset/Train/2...	[0.17254901960784...
40	34	6	6	34	29	20	./dataset/Train/2...	[0.17647058823529...
39	34	5	5	34	29	20	./dataset/Train/2...	[0.16470588235294...
42	36	6	5	37	31	20	./dataset/Train/2...	[0.17647058823529...
45	39	6	5	40	34	20	./dataset/Train/2...	[0.16470588235294...
47	42	5	5	41	36	20	./dataset/Train/2...	[0.16470588235294...
50	45	5	5	45	40	20	./dataset/Train/2...	[0.15686274509803...
55	49	6	5	49	43	20	./dataset/Train/2...	[0.15686274509803...

only showing top 20 rows

Рисунок 2.3 – Оновлений датафрейм з ознаками

Також було продемонстровано розподіл зображень один відносно одного за допомогою алгоритму зниження розмірності PCA (рисунок 2.4).

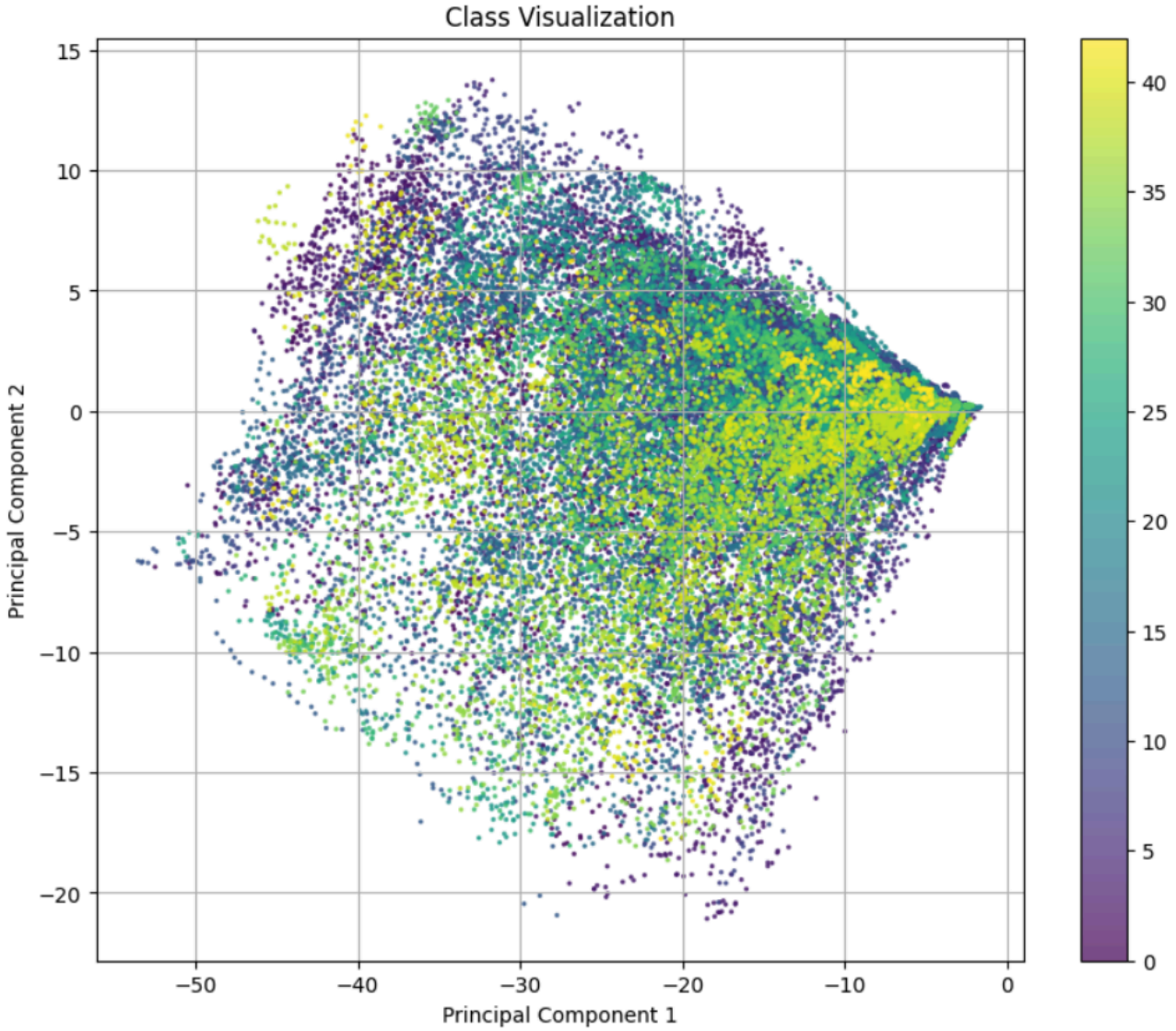

Рисунок 2.4 – Візуалізація класів зображень за допомогою PCA

На рисунку 2.5 продемонстровано кількість елементів кожного класу у тренувальному датасеті.

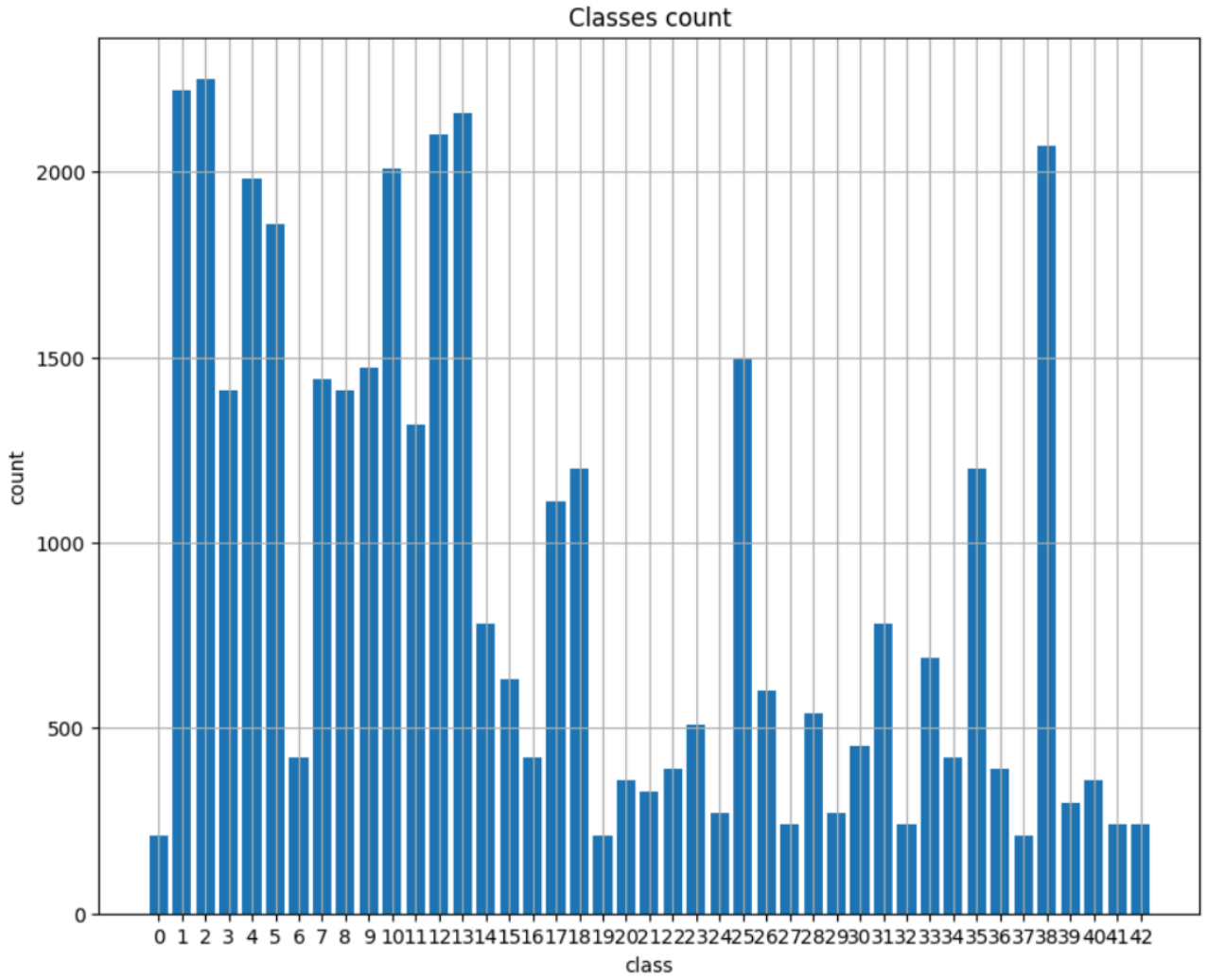

Рисунок 2.5 – Гістограма розподілу даних у датасеті

2.3 Збереження проміжних результатів

Результати підготовки даних, такі як оброблені зображення, вектори ознак та безпосередньо самі моделі, зберігаються у проміжних файлах, що забезпечує можливість їх повторного використання у різних етапах обробки. Цей підхід дозволяє значно скоротити час, необхідний для повторного виконання ресурсозатратних обчислень, особливо при роботі з великими обсягами даних.

Наприклад, оброблені зображення можуть бути використані для тестування декількох моделей класифікації без необхідності повторної підготовки. Додатково, проміжне збереження даних сприяє підвищенню стабільності процесу, оскільки у разі збоїв або змін у алгоритмі можна повернутися до збережених результатів і продовжити обробку без втрати раніше виконаної роботи. Таким чином, зберігання проміжних результатів забезпечує більш гнучкий та надійний підхід до підготовки даних.

2.4 Висновок до розділу

У цьому розділі детально розглянуто процес налаштування середовища розробки та виконання підготовки даних, що є основою для ефективного вирішення задач класифікації. Зокрема, було створено та налаштовано середовище, яке поєднує сучасні технології та апаратні ресурси, забезпечуючи зручність, масштабованість та швидкість обробки. Налаштування кластера Apache Spark дозволяє обробляти великі обсяги даних у розподіленому середовищі, що є критично важливим для задач, пов’язаних із машинним навчанням.

Описано ключові етапи підготовки даних, починаючи із завантаження метаданих, обробки зображень та створення векторів ознак. Використання таких підходів, як UDF у середовищі Spark, дає можливість автоматизувати та прискорити ці процеси, значно скорочуючи час виконання обчислень. Окремо розглянуто важливість проміжного збереження результатів, яке сприяє оптимізації роботи системи та забезпечує гнучкість при зміні умов виконання чи оновленні алгоритмів.

Таким чином, у цьому розділі закладено основу для подальшого навчання моделей на підготовлених даних. Це створює необхідні умови для успішного виконання всього проєкту та досягнення його цілей.

3 ОПИС МЕТОДІВ ОБРОБКИ ДАНИХ

3.1 Стандартні методи Spark MLib

Apache Spark MLlib пропонує широкий набір інструментів для машинного навчання, які забезпечують ефективну обробку даних у розподіленому середовищі. Зокрема, MLlib включає алгоритми класифікації, регресії, кластеризації та зменшення розмірності.

Однією з основних переваг MLlib є його інтеграція з іншими компонентами Spark, такими як Spark SQL та DataFrame API. Це дозволяє легко інтегрувати машинне навчання у ETL-процеси, використовуючи вже підготовлені набори даних. Крім того, MLlib оптимізований для роботи з великими обсягами даних, що робить його ідеальним вибором для класифікації дорожніх знаків в умовах великих датасетів [5].

3.1.1 Логістична регресія (Logistic Regression)

Логістична регресія є одним з найбільш популярних методів класифікації, доступних у MLlib [5]. Цей алгоритм використовує лінійну модель для оцінки ймовірності належності зразка до певного класу. У контексті класифікації дорожніх знаків її застосування має такі особливості:

- простота реалізації та обчислювальна ефективність, що дозволяє швидко створити робочу модель;
- ефективна для задач із лінійно роздільними класами;
- підтримка різноманітних регуляризаційних підходів, що допомагають уникати перенавчання.

Однак логістична регресія має і певні недоліки, зокрема:

- відсутність здатності моделювати нелінійні залежності між ознаками;
- чутливість до масштабування даних, що вимагає ретельної підготовки вхідних ознак.

У цій роботі логістична регресія буде застосовуватись як один із базових алгоритмів для побудови класифікаційної моделі через її простоту та ефективність.

3.1.2 Рандомний ліс (Random Forest)

Random Forest є потужним ансамблевим методом, який створює множину рішень на основі дерев рішень, кожне з яких навчається на різних підмножинах даних [5]. Переваги Random Forest включають:

- висока точність завдяки поєднанню кількох слабких моделей (дерев);
- стійкість до перенавчання через використання механізму випадкової вибірки даних і ознак;
- можливість працювати з нелінійними залежностями між ознаками.

Недоліки Random Forest:

- більш висока обчислювальна складність у порівнянні з логістичною регресією;
- менш інтуїтивне інтерпретування результатів.

У цій роботі Random Forest буде застосовуватись як один із алгоритмів навчання моделі розпізнавання дорожніх знаків.

3.2 Глибокі нейронні мережі

3.2.1 Tensorflow

TensorFlow – це бібліотека з відкритим кодом, що використовується для розробки та навчання моделей машинного навчання. У цій роботі TensorFlow забезпечує можливості для розробки складніших моделей, таких як згорткові нейронні мережі (CNN), які ідеально підходять для задач класифікації зображень. Також TensorFlow дозволяє створювати обчислювальні графи, які оптимізуються для роботи на різних апаратних архітектурах [6].

Основні переваги TensorFlow:

– підтримка складних моделей, включаючи багатошарові нейронні мережі та згорткові нейронні мережі (CNN);
– широкий вибір інструментів для візуалізації, налагодження та оптимізації моделей;
– можливість використання апаратного прискорення (GPU / TPU) для швидкого навчання великих моделей, підтримка технології NVIDIA Cuda.

До недоліків TensorFlow можна віднести:

– відносно високу складність налаштування та використання для початківців;
– велика кількість обчислювальних ресурсів, необхідних для тренування складних моделей.

TensorFlow буде показано у якості прикладу використання глибокого навчання та згорткових нейронних мереж з метою класифікації дорожніх знаків і порівняння такого підходу зі стандартними методами бібліотеки MLlib.

3.2.2 PyTorch

PyTorch – ще одна потужна бібліотека з відкритим кодом для машинного навчання, яка є популярною завдяки своїй простоті та гнучкості. У рамках задачі класифікації дорожніх знаків PyTorch також надає можливість реалізації складних нейронних мереж.

Переваги:

– зручність у використанні та налагодженні завдяки динамічним графам;
– потужна підтримка спільноти та велика кількість готових рішень;
– інтеграція з іншими бібліотеками та інструментами.

Недоліки:

- відносно більший обсяг коду для реалізації моделей порівняно з TensorFlow;
- не така ефективна оптимізація для великих проєктів, що потребують масштабування.

Хоч PyTorch забезпечує більшу гнучкість та простоту налагодження, що корисно для експериментальних досліджень та прототипування, але TensorFlow краще підходить для складних архітектур із високими вимогами до оптимізації та продуктивності, особливо у випадках масштабних моделей і великих датасетів.

3.3 Висновки до розділу

Розглянуті методи обробки даних пропонують різні підходи до вирішення задачі класифікації. Spark MLlib забезпечує ефективну обробку великих обсягів даних завдяки розподіленому середовищу. Логістична регресія є швидким та інтерпретованим методом, тоді як Random Forest підходить для задач, де потрібна висока точність при роботі з нелінійними даними. TensorFlow, у свою чергу, пропонує можливості для роботи зі складними моделями, проте вимагає значних ресурсів. Оптимальний вибір методу залежить від конкретних вимог задачі та доступних ресурсів.

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МЕТОДІВ ОБРОБКИ ДАНИХ

4.1 Стандартні методи MLlib

4.1.1 Підбір гіперпараметрів

Під час використання стандартних методів MLlib, таких як логістична регресія та рандомний ліс, важливим етапом є підбір гіперпараметрів. Для цього застосовувалася крос-валідація з метою оптимізації параметрів. У якості метрики при підборі гіперпараметрів використовувалась accuracy.

Для логістичної регресії підбрались такі гіперпараметри:

– коефіцієнт регуляризації (regParam);
– tolerance (tol);
– параметр еластичної сітки (elasticNetParam).

У таблиці 4.1 показано оптимальні гіперпараметри логістичної регресії, що були підібрані у рамках емпіричного дослідження.

Таблиця 4.1 – Накращі гіперпараметри логістичної регресії

Параметр	Значення
regParam	0.001
tol	0
elasticNetParam	0.01

При вищевказаних гіперпараметрах та кількості ітерацій 80 точність моделі логістичної регресії склала приблизно 0.86.

Дослідження показало, що у логістичній регресії оптимізація регуляризації дозволяє зменшити перенавчання, що є критичним для великих наборів даних.

Для рандомного лісу підбрались такі гіперпараметри:

– кількість дерев (numTrees);
– максимальна глибина (maxDepth);

– кількість бінів (maxBins).

У таблиці 4.2 показано оптимальні гіперпараметри логістичної регресії, що були підібрані у рамках емпіричного дослідження.

Таблиця 4.2 – Накращі гіперпараметри рандомного лісу

Параметр	Значення
numTrees	100
maxDepth	20
maxBins	12

При вищевказаних гіперпараметрах точність моделі рандомного лісу склала приблизно 0.82.

Результати підбору свідчать про те, що збільшення кількості дерев у рандомному лісі позитивно впливає на точність, але потребує більше ресурсів.

4.1.2 Порівняння швидкості навчання

У ході експериментів було проведено заміри швидкості навчання моделей. Логістична регресія виявилася більш швидкою, що пояснюється її відносно простою математичною моделлю. У той же час, рандомний ліс показав значно більшу тривалість навчання через потребу у побудові та аналізі великої кількості дерев. Порівняння швидкості навчання за найкращих гіперпараметрів показано у таблиці 4.3.

Таблиця 4.3 – Порівняння швидкості навчання методів MlLib

Модель	Час навчання (хв)
Логістична регресія	33
Рандомний ліс	94

4.1.3 Порівняння точності та швидкості класифікації

Точність моделей оцінювалася за допомогою метрик accuracy та f1. Швидкість та точність класифікації була вищою у логістичної регресії, що робить її більш придатною для вирішення задач в умовах класифікації дорожніх знаків.

4.2 Глибинні нейронні мережі

4.2.1 Обгрунтування архітектури нейронної мережі

Для задачі класифікації дорожніх знаків була обрана згорткова нейронна мережа (CNN), яка є стандартом у роботі із зображеннями.

Архітектура нейронної мережі включала:

- Вхідний згортковий шар, у якому використовувались фільтр розміром 3x3 для виявлення локальних патернів у зображенні та активаційна функція ReLU для врахування нелінійностей
- Шар максимального пулінгу (MaxPooling2D) з розміром вікна 2x2 для зменшення розмірності даних і виділення найбільш значущих ознак.
- Додаткові згорткові шари, кількість яких варіювалася залежно від гіперпараметра hp_num_conv_layers. Кожен наступний шар збільшував кількість фільтрів у два рази, що дозволяло враховувати складніші патерни. Після кожного додаткового згорткового шару додавався шар MaxPooling2D.
- Шар глобального усереднення GlobalAveragePooling2D, який перетворював тривимірні дані у вектор фіксованого розміру, що спрощувало подальшу обробку.
- Повнозв’язні шари, які включали до трьох додаткових шарів з кількістю нейронів, що підбиралася автоматично (гіперпараметр hp_units). У якості функції активації використовувалась ReLU для

кожного шару. З метою уникнення перенавчання після кожного шару було розміщено BatchNormalization та Dropout із коефіцієнтом 0.5.

– Вихідний шар, що містив 43 нейрони (відповідає кількості класів) та softmax-функцію активації для формування ймовірнісного розподілу класів.

Використовувались готові шари з бібліотеки Keras. Використання активаційної функції ReLU дозволило врахувати нелінійність, тоді як softmax на останньому шарі забезпечив ймовірнісний розподіл класів. У якості оптимізатора було використано алгоритм Adam. У якості функції похибки було обрано перехресну крос-ентропію.

4.2.2 Підбір гіперпараметрів

У таблиці 4.4 показано результати підбору гіперпараметрів.

Таблиця 4.4 – Результат підбору гіперпараметрів нейронної мережі

Назва	Коротке пояснення	Діапазон значень при підборі	Крок підбору	Оптимальне значення
hp_filters	Фільтри згорткових шарів	32-128	32	64
hp_units	Кількість нейронів в повнозв’язних шарах	32-512	32	480
hp_learning_rate	Швидкість навчання	1e-3, 1e-4, 1e-5	-	1e-4
hp_num_conv_layers	Кількість додаткових згорткових шарів	0-2	1	2
hp_num_dense_layers	Кількість повнозв’язних шарів	0-3	1	2

4.2.3 Швидкість навчання

Порівняно зі стандартними методами MLlib, нейронна мережа навчалася значно швидше завдяки використанню GPU для обчислень, що зменшило загальний час навчання. Паралельне виконання обчислень на GPU значно скоротило час навчання порівняно з CPU. Час підбору гіперпараметрів - 20 хвилин.

4.2.4 Точність та швидкість класифікації

Нейронна мережа показала найвищу точність (0.97) серед усіх методів, що пояснюється її здатністю враховувати складні просторові залежності у зображеннях. Також за рахунок використання GPU, виконання передбачення класу зображення відбувається швидше, ніж у методів MLlib

4.3 Висновки до розділу

Результати дослідження показали, що кожен з методів має свої переваги та недоліки. Глибинні нейронні мережі демонструють найвищу точність, але потребують більше часу та ресурсів для навчання. У моєму випадку, завдяки використанню потужної GPU, навчання CNN відбувалося швидше, що дозволило отримати високі результати за значно менший час.

ВИСНОВКИ

У ході виконання роботи було розглянуто сучасні підходи до класифікації зображень, зокрема в умовах надвеликих масивів даних. Основна увага приділялася аналізу ефективності методів обробки даних та побудови моделей класифікації. Вибір методів базувався на можливостях їх адаптації до специфіки задачі – класифікації дорожніх знаків, а також на доступних обчислювальних ресурсах.

У першому розділі досліджено предметну область, визначено мету роботи та сформульовано вимоги до програмного забезпечення. Це дозволило чітко окреслити завдання та цілі, а також забезпечити базу для наступних етапів розробки.

У другому розділі описано процеси підготовки даних, що включали обробку зображень, нормалізацію та перетворення даних для їх подальшого використання у моделях машинного навчання. Важливим етапом було налаштування середовища виконання, включаючи використання сучасних інструментів, таких як Apache Spark і TensorFlow, що сприяло ефективній реалізації задачі.

Третій розділ присвячено опису методів обробки даних. Розглянуто класичні методи машинного навчання, такі як логістична регресія та рандомний ліс, а також сучасні глибинні нейронні мережі. Проведено порівняльний аналіз переваг і недоліків кожного методу, зокрема їх точності, швидкості навчання та застосовності до обробки великих масивів даних. Особливу увагу приділено використанню нейронних мереж, які показали найкращі результати у задачах класифікації зображень.

У четвертому розділі проведено дослідження ефективності реалізованих методів. Виконано порівняння моделей за швидкістю навчання, точністю класифікації, а також оптимізацією гіперпараметрів. Результати експериментів підтвердили доцільність використання глибинних нейронних мереж для складних задач класифікації. У цьому контексті використання GPU значно прискорило навчання та обробку даних, що було особливо актуально в умовах роботи з великими обсягами інформації.

Таким чином, проведена робота не лише підтвердила ефективність глибинного навчання у задачах класифікації зображень, але й дозволила визначити оптимальні методи для реалізації подібних задач у реальних умовах. Отримані результати можуть бути використані для подальшого вдосконалення процесів машинного навчання, зокрема в задачах розпізнавання об’єктів та обробки візуальних даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Multimedia Docs. Using Convolutional Neural Networks for Image Recognition. Електронний ресурс: https://www.multimediadocs.com/assets/cadence_emea/documents/using_convolutional_neural_networks_for_image_recognition.pdf.
2. Ching-Hao Lai, Yu Chia-Chen. An Efficient Realtime Traffic Sign Recognition System for Intelligent Vehicles with Smart Phones. IEEE International Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 195–202, 2010.
3. Apache Spark. Unified analytics engine for large-scale data processing. Електронний ресурс: https://spark.apache.org/.
4. An Overview of Traffic Signs Recognition. International Journal of Computer Applications. Електронний ресурс: https://d1wqtxts1xzle7.cloudfront.net/71655563/ijca2017914524-libre.pdf.
5. MLlib: Machine Learning in Apache Spark. Journal of Machine Learning Research, 17: 1–41, 2015. Електронний ресурс: https://www.jmlr.org/papers/volume17/15-237/15-237.pdf.
6. TensorFlow. An end-to-end open-source platform for machine learning. Електронний ресурс: https://www.tensorflow.org/.

ДОДАТКИ

Додаток А Текст програмного коду

```
      from pyspark.sql import SparkSession
      import os
      from  pyspark.sql.types  import  ArrayType,  FloatType,  IntegerType,  StringType,
StructType, StructField
      from pyspark.sql.functions import concat, lit
      from pyspark.ml import Pipeline, PipelineModel
      from pyspark.ml.feature import SQLTransformer
      from pyspark.ml.classification import LogisticRegression, RandomForestClassifier
      from pyspark.ml.evaluation import MulticlassClassificationEvaluator
      from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
      from pyspark.sql.functions import udf
      from pyspark.ml.linalg import Vectors, VectorUDT
      import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      from PIL import Image
      from sklearn.metrics import confusion_matrix
      spark:     SparkSession     =     SparkSession.builder.appName("Traffic     Signs
Classification") \
        .master("local[*]") \
        .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
        .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
        .config("spark.rapids.sql.enabled", "true") \
        .config("spark.executor.memory", "12g") \
        .config("spark.driver.memory", "12g") \
        .config("spark.python.worker.memory", "12g") \
        .config("spark.executor.pyspark.memory", "12g") \
        .config("spark.rpc.message.maxSize", "128") \
        .config("spark.executor.memoryOverhead", "2g") \
        .config("spark.executor.heartbeatInterval", "30s") \
        .config("spark.network.timeout", "300s") \
        .config("spark.sql.adaptive.enabled", "true") \
        .config("spark.memory.fraction", "0.6") \
        .config("spark.driver.maxResultSize", "2g") \
        .config("spark.sql.shuffle.partitions", "20") \
        .config("spark.default.parallelism", "20") \
        .config("spark.sql.streaming.checkpointLocation", "./tmp") \
        .getOrCreate()

      spark.sparkContext.setLogLevel("ERROR")

        # .config('spark.driver.extraClassPath', 'rapids-4-spark_2.13-24.10.1.jar') \
        # .config('spark.executor.extraClassPath', 'rapids-4-spark_2.13-24.10.1.jar') \
        # .config("spark.plugins", "com.nvidia.spark.rapids.GpuPlugin") \
        # .config("spark.sql.extensions", "com.nvidia.spark.rapids.SQLExecPlugin") \
        # .config("spark.executor.resource.gpu.amount", "1") \


        #   .master("local[*]") \
        # .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
        # .config("spark.local.dir", "/mnt/e/spark-temp") \
        # .config("spark.eventLog.dir", "/mnt/e/spark-event-logs") \
```

```
        # .master("local[*]") \
        #
               #     .config("spark.executor.extraJavaOptions",     "-XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:tmp/gc.log") \
        # .config("spark.sql.execution.arrow.pyspark.enabled", "false") \

      dataset_path = './dataset'

      if not os.path.exists(dataset_path):
        raise BaseException("Path not found")

      schema = StructType([
          StructField("Width", IntegerType(), True),
          StructField("Height", IntegerType(), True),
          StructField("Roi.X1", IntegerType(), True),
          StructField("Roi.Y1", IntegerType(), True),
          StructField("Roi.X2", IntegerType(), True),
          StructField("Roi.Y2", IntegerType(), True),
          StructField("ClassId", IntegerType(), True),
          StructField("Path", StringType(), True)
      ])

      train_data_df     =     spark.read.csv(os.path.join(dataset_path,     'Train.csv'),
schema=schema, header=True)
      train_data_df = train_data_df.withColumn("Path", concat(lit(dataset_path + "/"),
train_data_df["Path"])).dropna(subset=["ClassId"])
      train_data_df.show()
      class_labels = {
          0: "Speed limit (20km/h)",
          1: "Speed limit (30km/h)",
          2: "Speed limit (50km/h)",
          3: "Speed limit (60km/h)",
          4: "Speed limit (70km/h)",
          5: "Speed limit (80km/h)",
          6: "End of Speed limit (80km/h)",
          7: "Speed limit (100km/h)",
          8: "Speed limit (120km/h)",
          9: "No overtaking",
          10: "No overtaking for trucks",
          11: "Priority at next intersection",
          12: "Priority road",
          13: "Yield",
          14: "Stop",
          15: "No vehicles",
          16: "Vehicles over 3.5 metric tons prohibited",
          17: "No entry",
          18: "General caution",
          19: "Dangerous curve to the left",
          20: "Dangerous curve to the right",
          21: "Double curve",
          22: "Bumpy road",
          23: "Slippery road",
          24: "Road narrows on the right",
          25: "Road work",
          26: "Traffic signals",
          27: "Pedestrians",
```

```
            28: "Children crossing",
            29: "Bicycles crossing",
            30: "Beware of ice/snow",
            31: "Wild animals crossing",
            32: "End of all speed and passing limits",
            33: "Turn right ahead",
            34: "Turn left ahead",
            35: "Ahead only",
            36: "Go straight or right",
            37: "Go straight or left",
            38: "Keep right",
            39: "Keep left",
            40: "Roundabout mandatory",
            41: "End of no overtaking",
            42: "End of no overtaking for trucks"
        }

        def process_image(img_path, roi_x1, roi_y1, roi_x2, roi_y2):
            try:
                img = Image.open(img_path)
                cropped_img = img.crop((roi_x1, roi_y1, roi_x2, roi_y2))
                                resized_img   =   np.array(cropped_img.resize((32,   32),
resample=Image.Resampling.LANCZOS))
                return (resized_img.flatten() / 255.0).tolist()
            except Exception as e:
                print(f"Error processing image {img_path}: {e}")
                return [0.0] * (32 * 32 * 3)

        def image_to_vector(img_features):
            return Vectors.dense(img_features)

        image_to_vector_udf = udf(image_to_vector, VectorUDT())

        spark.udf.register("process_image", process_image, ArrayType(FloatType()))
        spark.udf.register("image_to_vector", image_to_vector, VectorUDT())

        image_transformer          =          SQLTransformer(statement='SELECT          *,
image_to_vector(process_image(Path, `Roi.X1`, `Roi.Y1`, `Roi.X2`, `Roi.Y2`)) features
FROM __THIS__')

        processed_df = image_transformer.transform(train_data_df)

        processed_df.show()

        processed_df.write.mode("overwrite").parquet("processed_df")
        processed_df = spark.read.parquet("processed_df")
        processed_df.show()
        fig, ax = plt.subplots(len(class_labels), figsize=(10, 100))

        for key in class_labels:
          img = processed_df.filter(processed_df.ClassId == key).limit(1).toPandas()

          reshaped_image = img.iloc[0].features.reshape(32, 32, 3)
          ax[key].imshow(reshaped_image)

        plt.show()
        from pyspark.ml.feature import PCA
        import matplotlib.pyplot as plt
```

```
from pyspark.sql.functions import col
import numpy as np

pca = PCA(k=2, inputCol="features", outputCol="pca_features")

pca_model = pca.fit(processed_df)
pca_data = pca_model.transform(processed_df)

pca_points = pca_data.select("pca_features", "ClassId").rdd.map(
    lambda row: (row["pca_features"][0], row["pca_features"][1], row["ClassId"])
).collect()

x = [point[0] for point in pca_points]
y = [point[1] for point in pca_points]
labels = [point[2] for point in pca_points]

plt.figure(figsize=(10, 8))
plt.scatter(x, y, c=labels, alpha=0.7, s=2)
plt.colorbar()
plt.title(f"Class Visualization")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()
count_data = processed_df.groupBy("ClassId").count().toPandas()

plt.figure(figsize=(10, 8))
plt.bar(count_data["ClassId"], count_data["count"])
plt.title(f"Classes count")
plt.xticks(count_data["ClassId"])
plt.xlabel("class")
plt.ylabel("count")
plt.grid(True)
plt.show()
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="ClassId",
predictionCol="prediction", metricName="accuracy")
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="ClassId",
predictionCol="prediction", metricName="f1")
lr = LogisticRegression(featuresCol="features", labelCol="ClassId", maxIter=80)
pipeline = Pipeline(stages=[image_transformer, lr])

paramGrid_lr = ParamGridBuilder() \
  .addGrid(lr.regParam, [1e-4, 1e-3, 0.01]) \
  .addGrid(lr.tol, [0, 0.5]) \
  .addGrid(lr.elasticNetParam, [1e-3, 0.01, 0.1]) \
  .build()


cv = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid_lr,
evaluator=evaluator_accuracy, numFolds=5, seed=1234)

model = cv.fit(train_data_df)

bestModel = model.bestModel
bestModel.write().overwrite().save("best-model-lr")
def show_confusion_matrix(predictions):
    predictions_pd = predictions.toPandas()
```

```
                        conf_matrix    =    confusion_matrix(predictions_pd['ClassId'],
predictions_pd['prediction'])

          fig, ax = plt.subplots(figsize=(30, 8))
          cax = ax.matshow(conf_matrix, cmap="Blues")

          plt.colorbar(cax)

          ax.set_xticks(range(len(class_labels)))
          ax.set_yticks(range(len(class_labels)))
          ax.set_xticklabels(class_labels, fontsize=7)
          ax.set_yticklabels(class_labels, fontsize=7)

          plt.title("Confusion Matrix", pad=20)
          plt.xlabel("Predicted", labelpad=10)
          plt.ylabel("Actual", labelpad=10)

          for (i, j), value in np.ndenumerate(conf_matrix):
                  ax.text(j,  i,  f'{value}',  ha='center',  va='center',  color='black',
fontsize=7)

          plt.tight_layout()
          plt.show()

      bestModel = PipelineModel.load("best-model-lr")
      print("Best lr params:")
      print(f"regParam={bestModel.stages[-1].getRegParam()}
tol={bestModel.stages[-1].getTol()}
elasticNetParam={bestModel.stages[-1].getElasticNetParam()}")

      test_metadata_df    =    spark.read.csv(os.path.join(dataset_path,    'Test.csv'),
schema=schema, header=True)
      test_metadata_df = test_metadata_df.withColumn("Path", concat(lit(dataset_path +
"/"), test_metadata_df["Path"]))
      final_predictions = bestModel.transform(test_metadata_df)
      accuracy = evaluator_accuracy.evaluate(final_predictions)
      f1 = evaluator_f1.evaluate(final_predictions)

      print(f"Final results:")
      print(f"Accuracy: {accuracy}")
      print(f"F1: {f1}")

      show_confusion_matrix(final_predictions)
      rf = RandomForestClassifier(featuresCol="features", labelCol="ClassId")
      pipeline = Pipeline(stages=[image_transformer, rf])

      paramGrid_rf = ParamGridBuilder() \
        .addGrid(rf.numTrees, [50, 100]) \
        .addGrid(rf.maxDepth, [17, 20, 22]) \
        .addGrid(rf.maxBins, [10, 12, 15]) \
        .build()

      cv    =    CrossValidator(estimator=pipeline,    estimatorParamMaps=paramGrid_rf,
evaluator=evaluator_accuracy, numFolds=5, seed=1234)

      model = cv.fit(train_data_df)

      bestModel = model.bestModel
```

```
      bestModel.write().overwrite().save("best-model-rf")
      bestModel = PipelineModel.load("best-model-rf")
      print("Best rf params:")

      print(f"numTrees={bestModel.stages[-1].getNumTrees}
maxDepth={bestModel.stages[-1].getMaxDepth()}
maxBins={bestModel.stages[-1].getMaxBins()}")
      test_metadata_df    =    spark.read.csv(os.path.join(dataset_path,    'Test.csv'),
schema=schema, header=True)
      test_metadata_df = test_metadata_df.withColumn("Path", concat(lit(dataset_path +
"/"), test_metadata_df["Path"]))

      test_metadata_df.show()

      final_predictions = bestModel.transform(test_metadata_df)

      accuracy = evaluator_accuracy.evaluate(final_predictions)
      f1 = evaluator_f1.evaluate(final_predictions)

      print(f"Final results:")
      print(f"Accuracy: {accuracy}")
      print(f"F1: {f1}")

      show_confusion_matrix(final_predictions)
      test_metadata_df = spark.readStream.csv(os.path.join(dataset_path, 'streaming'),
schema=schema, header=True)
      test_metadata_df = test_metadata_df.withColumn("Path", concat(lit(dataset_path +
"/"), test_metadata_df["Path"]))

      predictions_stream = bestModel.transform(test_metadata_df)

      output_stream                                                                    =
predictions_stream.writeStream.outputMode("append").format("console").start()
      output_stream.awaitTermination()

      import pandas as pd
      import numpy as np
      import os
      from PIL import Image
      from tensorflow.keras.models import load_model

      dataset_path = './dataset'

      if not os.path.exists(dataset_path):
        raise BaseException("Path not found")

      train_data = pd.read_csv(os.path.join(dataset_path, "Train.csv"))
      train_data

      test_data = pd.read_csv(os.path.join(dataset_path, "Test.csv"))
      test_data

      def load_image(row):
        try:
          path = os.path.join(dataset_path, row["Path"])
          with Image.open(path) as img:
                cropped_img = img.crop((row['Roi.X1'], row['Roi.Y1'], row['Roi.X2'],
row['Roi.Y2']))
```

```
                            resized_img    =    np.array(cropped_img.resize((32,    32),
resample=Image.Resampling.LANCZOS))
            return resized_img / 255.0
        except Exception as e:
          print(f"Error loading image {row['Path']}: {e}")
          return None

      train_data['image'] = train_data.apply(load_image, axis=1)
      test_data['image'] = test_data.apply(load_image, axis=1)

      import tensorflow as tf
      from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, Conv2D,
MaxPooling2D, GlobalAveragePooling2D
      from tensorflow.keras.utils import to_categorical
      from tensorflow.keras.optimizers import Adam
      import keras_tuner as kt

      num_classes = train_data['ClassId'].nunique()

      x_train = np.stack(train_data['image'].values)
      y_train = to_categorical(train_data['ClassId'].values, num_classes)
      print(f"[Training] Feature shape: {x_train.shape}, Label shape: {y_train.shape}")

      x_test = np.stack(test_data['image'].values)
      y_test = to_categorical(test_data['ClassId'].values, num_classes)
      print(f"[Testing] Feature shape: {x_test.shape}, Label shape: {y_test.shape}")

      def build_model(hp):
        hp_filters = hp.Int('filters', min_value=32, max_value=128, step=32)
        hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-3, 1e-4, 1e-5])
          hp_num_conv_layers  =  hp.Int('hp_num_additional_conv_layers',  min_value=0,
max_value=2)
        hp_num_dense_layers = hp.Int('num_dense_layers', min_value=0, max_value=3)

        model = Sequential()

         model.add(Conv2D(hp_filters, (3, 3), activation='relu', input_shape=(32, 32,
3)))
        model.add(MaxPooling2D((2, 2)))

        i = 2
        for _ in range(hp_num_conv_layers):
          model.add(Conv2D(hp_filters * i, (3, 3), activation='relu'))
          model.add(MaxPooling2D((2, 2)))
          i = i * 2

        model.add(GlobalAveragePooling2D())

        for _ in range(hp_num_dense_layers):
            model.add(Dense(hp_units, activation='relu'))
            model.add(BatchNormalization())
            model.add(Dropout(0.5))

        model.add(Dense(43, activation='softmax'))

        model.compile(optimizer=Adam(learning_rate=hp_learning_rate),
```

```
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])
      return model

    tuner = kt.Hyperband(build_model,
                         objective='val_accuracy',
                         max_epochs=20,
                         factor=3,
                         directory='tmp',
                         project_name='signs-recognition')
    stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
    tuner.search(x_train, y_train, validation_data=(x_test, y_test), epochs=50,
callbacks=[stop_early])

    best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

    print("Best parameters:")
    print(f"filters={best_hps.get('filters')}")
    print(f"units={best_hps.get('units')}")
    print(f"learning_rate={best_hps.get('learning_rate')}")
    print(f"hp_num_additional_conv_layers={best_hps.get('hp_num_additional_conv_layer
s')}")
    print(f"num_dense_layers={best_hps.get('num_dense_layers')}")

    model = tuner.hypermodel.build(best_hps)
    history = model.fit(x_train, y_train, epochs=100, validation_data=(x_test,
y_test), callbacks=[stop_early])

    model.evaluate(x_test, y_test)
    model.save("best_model.keras")
    model = load_model("best_model.keras")
    model.evaluate(x_test, y_test)
```