

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розпізнавання дорожніх знаків за допомогою
згорткової нейронної мережі»

Виконав студент групи ІПЗм-21-1
МУКОВОЗ Віктор Сергійович

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
 ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
 ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
 КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
 зав. кафедри інженерії
 програмного забезпечення
 Тетяна ВАКАЛЮК
 «18» вересня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Здобувач вищої освіти: МУКОВОЗ Віктор Сергійович

Керівник роботи: ВАКАЛЮК Тетяна Анатоліївна

Тема роботи: «Розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі»,

затверджена Наказом закладу вищої освіти від «18» вересня 2023 р. № 479/с

Вихідні дані для роботи: предметом дослідження є теоретичні та практичні аспекти розробки та застосування згорткових нейронних мереж для розпізнавання дорожніх знаків. Об'єктом дослідження є розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	22.09.2023р.	22.09.2023р.
2	Вакалюк Т.А.	22.09.2023р.	22.09.2023р.
3	Вакалюк Т.А.	22.09.2023р.	22.09.2023р.

РЕФЕРАТ

Завданням кваліфікаційної роботи є проведення наукового дослідження та обґрунтуванні вибору моделей, методів та технологій для розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Кваліфікаційна магістерська робота на тему «Розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі» складається з програмного комплексу, який включає в себе програму навчання моделі на наборі даних та додатку ідентифікації та класифікації дорожніх знаків, а також пояснювальної записки. Пояснювальна записка до кваліфікаційної магістерської роботи складається зі вступу, трьох розділів, висновків та списку використаної літератури. В роботі наведено 42 рисунки та 1 таблицю. Загальний обсяг роботи –74 сторінок.

В першому розділі кваліфікаційної роботи проведено аналіз проблематики та методів вирішення задачі, а також здійснено постановку завдання.

В другому розділі було розглянуто практичну реалізацію алгоритмів навчання нейронних мереж.

В третьому розділі було розглянуто програмну реалізацію додатку розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Висновок містить підсумки проведеного дослідження.

Ключові слова: CNN, НЕЙРОННА МЕРЕЖА, МОДЕЛЬ, НАБІР ДАНИХ, ДОРОЖНІ ЗНАКИ, ШТУЧНИЙ ІНТЕЛЕКТ.

ABSTRACT

The task of the qualification work is to conduct a scientific study and justify the choice of models, methods and technologies for road sign recognition using a convolutional neural network.

The master's thesis on "Road Sign Recognition Using Convolutional Neural Network" consists of a software package that includes a model training program on a dataset and an application for identifying and classifying road signs, as well as an explanatory note. The explanatory note to the master's thesis consists of an introduction, three chapters, conclusions, and a list of references. The work contains 42 figures and 1 table. The total volume of the work is 74 pages.

The first chapter of the qualification work analyzes the problems and methods of solving the problem, as well as formulates the task.

The second section deals with the practical implementation of neural network training algorithms.

The third chapter discussed the software implementation of the road sign recognition application using a convolutional neural network.

The conclusion contains the results of the conducted research.

Keywords: CNN, NEURAL NETWORK, MODEL, DATASET, ROAD SIGNS, ARTIFICIAL INTELLIGENCE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	5
ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	8
1.1 Аналіз методів розпізнавання зображень	8
1.2 Дорожні знаки України	16
1.3 Аналіз існуючих систем, що застосовуються для розпізнавання дорожніх знаків.....	19
1.4 Постановка задачі.....	25
1.5 Вибір засобів реалізації програмного забезпечення.....	26
Висновки до розділу 1	29
РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ НАВЧАННЯ НЕЙРОННИХ МОДЕЛЕЙ	30
2.1 Опис алгоритму розпізнавання знаків за допомогою CNN	30
2.2 Аналіз вхідних даних для проведення навчання	31
2.3 Тренування моделі для класифікації дорожніх знаків	36
2.4 Результати проведеного навчання.....	38
Висновки до розділу 2	45
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ	46
3.1 Розробка додатку розпізнавання дорожніх знаків	46
3.2 Розробка інтерфейсу додатку.....	50
3.3 Тестовий приклад роботи додатку	52
Висновки до розділу 3	56
ВИСНОВОК.....	57
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	58
ДОДАТКИ.....	61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CNN – Convolutional neural network

ViTs – Vision Transformers

RNN – Recurrent neural network

SLA – Speed Limit Assist

RSI – Road sign information

GTSRB – German Traffic Sign Recognition Benchmark

NMS – Non-Maximum Suppression

GUI – Graphical user interface

ВСТУП

Розпізнавання дорожніх знаків за допомогою нейронної мережі є надзвичайно актуальним у сучасному світі, де технології штучного інтелекту і машинного навчання проникають у всі сфери життя, включаючи автомобільну промисловість і дорожній рух. Ця тема відіграє ключову роль у розвитку безпечних і ефективних систем автономного водіння, які є одним з найбільш обговорюваних напрямків у сфері автомобільних технологій.

Сучасні системи автономного водіння потребують високоточного розпізнавання дорожніх знаків для безпечного управління автомобілем. Нейронні мережі, завдяки своїй здатності ефективно обробляти великі обсяги даних та вчитися на прикладах, є ідеальними для цієї задачі.

Системи, здатні точно ідентифікувати та реагувати на дорожні знаки, можуть значно знизити ризик дорожньо-транспортних пригод. Вони можуть попереджати водіїв про зміни в обмеженнях швидкості, закриті дороги, перехрестя, пішохідні переходи та інші важливі аспекти дорожнього середовища. Це особливо важливо в складних дорожніх умовах, де водії можуть не помітити або ігнорувати дорожні знаки.

Метою роботи є розробка і дослідження системи розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Для досягнення поставленої мети дослідження необхідно виконати наступні завдання:

- вивчити теоретичні основи згорткових нейронних мереж;
- розробити систему розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі;
- навчити систему розпізнавання дорожніх знаків на наборі даних з дорожніх знаків;
- оцінити точність системи розпізнавання дорожніх знаків.

Об'єктом дослідження є розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Предметом дослідження є теоретичні та практичні аспекти розробки та застосування згорткових нейронних мереж для розпізнавання дорожніх знаків.

В кваліфікаційній роботі було використано такі наступні методи дослідження: теорії згорткових нейронних мереж, аналіз даних та обробка зображень, статистичний аналіз.

Наукова новизна результату дослідження у сфері розпізнавання дорожніх знаків за допомогою згорткових нейронних мереж (CNN) включає:

- вдосконалено методологію обробки зображень, яка інтегрує нові алгоритми передобробки та післяобробки даних, що значно підвищує ефективність системи у різних умовах;
- розроблено модифіковані підходи до аналізу великих наборів даних, які забезпечують більш точне та швидке розпізнавання дорожніх знаків;
- реалізовано нові алгоритми навчання та оптимізації, що дозволяють знизити обчислювальну складність та підвищити швидкість роботи системи без втрати точності.

Матеріали кваліфікаційної роботи доповідалися на XVI Всеукраїнській науково практичній WEB конференції аспірантів, студентів та молодих вчених: «Комп'ютерні інтелектуальні системи та мережі – 2023 (KICM-2023)» 21-23 березня 2023 р. [1].

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Аналіз методів розпізнавання зображень

Розпізнавання зображень є однією з найбільш складних і важливих задач в області штучного інтелекту і комп'ютерного зору. Ця область має великий потенціал застосування в різних сферах, включаючи медицину, автономні транспортні засоби, безпеку та багато інших.

Розвиток розпізнавання зображень почався в 1950-ті роки з експериментів у галузі штучного інтелекту та машинного навчання[2]. В цей період, основну увагу приділяли алгоритмам, здатним ідентифікувати прості форми та символи. Наприклад, одним із перших проектів у цій області була система, розроблена Френком Розенблаттом у 1957 році, відома як "Перцептрон". Ця система могла розпізнавати прості образи та була одним із перших прикладів нейронних мереж.

Протягом 1980-1990-х років сфера розпізнавання зображень значно розширилася завдяки зростанню обчислювальної потужності та розвитку алгоритмів. У цей період було розроблено багато основних алгоритмів та технік, що використовуються сьогодні, включаючи методи обробки зображень, такі як фільтрація, згладжування та виявлення країв. Також з'явилися перші комерційні системи оптичного розпізнавання символів (OCR), що використовувалися для автоматичного введення даних з документів.

Найбільший стрибок у розвитку розпізнавання зображень відбувся з появою глибоких нейронних мереж у 2000-их роках. Цей період характеризується використанням конволюційних нейронних мереж (CNN), які значно покращили здатність комп'ютерних систем розпізнавати складні зображення. Одним із ключових моментів стало введення ImageNet Large Scale Visual Recognition Challenge (ILSVRC) у 2010 році, де команди з усього світу змагалися у точності розпізнавання та класифікації зображень.

Згорткові нейронні мережі (CNN). CNN - це клас глибоких нейронних мереж, спеціально розроблених для вирішення завдань обробки та розпізнавання

зображень. Вони відіграли ключову роль у розвитку комп'ютерного зору, дозволивши комп'ютерам розуміти та інтерпретувати візуальні дані з надзвичайною точністю[3]. CNN зробили революцію в різних сферах - від автономного водіння до медичної візуалізації, що зробило їх наріжним каменем сучасного штучного інтелекту.

Однією з визначальних характеристик CNN є їхня здатність автоматично вивчати ієрархічні ознаки з необроблених значень пікселів. Це має вирішальне значення для виявлення значущих патернів на різних рівнях абстракції. CNN досягають цього за допомогою серії шарів, кожен з яких слугує певній меті:

1. Згорткові шари: Ці шари застосовують згорткові фільтри до вхідних зображень, дозволяючи мережі виявляти такі особливості, як краї, кути і текстури[4]. Згортання передбачає поелементне множення фільтра на локальні ділянки зображення з подальшим підсумовуванням. Кілька фільтрів вивчають різні особливості.

2. Об'єднання шарів: Об'єднання шарів зменшує вибірку карт об'єктів, зменшуючи просторові розміри та обчислювальну складність. Наприклад, максимальне об'єднання вибирає максимальне значення з групи сусідніх пікселів, виділяючи найбільш характерні ознаки.

3. Повністю з'єднані шари: У фінальних шарах карти об'єктів згладжуються і з'єднуються з повністю зв'язаними шарами, які виконують завдання класифікації або регресії. Ці шари роблять прогнози на основі вивчених ознак.

Структуру згорткової нейронної мережі зображено на рисунку 1.1.

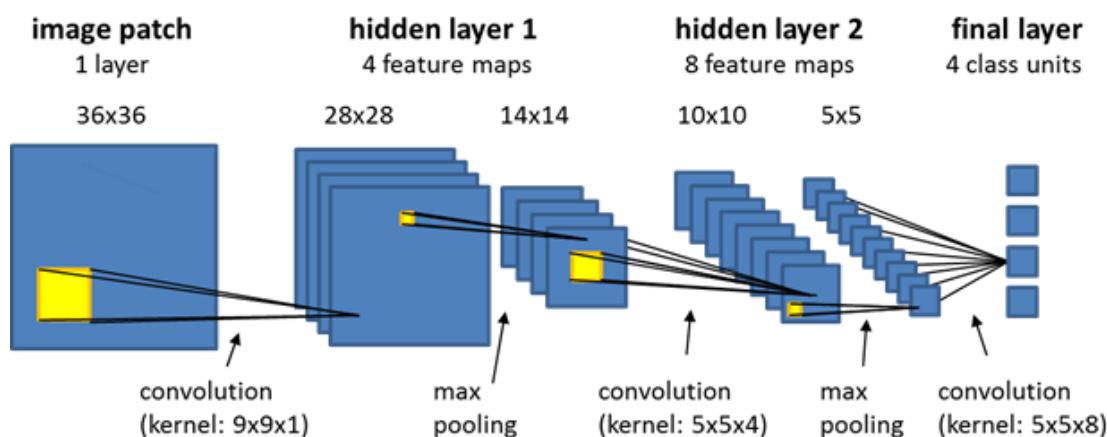


Рисунок 1.1 - Структура згорткової нейронної мережі

Основною концепцією згорткових шарів є спільне використання параметрів, що значно зменшує кількість параметрів, які можна вивчити. До всього зображення застосовується один фільтр, який навчається виявляти певну особливість, незалежно від її розташування[5]. Ця властивість робить CNN інваріантними до перекладу і здатними розпізнавати патерни незалежно від їхнього розташування на зображенні.

CNN навчаються за допомогою зворотного поширення - процесу, який передбачає ітеративне налаштування параметрів мережі для мінімізації різниці між прогнозованими та фактичними результатами. Градієнти функції втрат по відношенню до параметрів обчислюються і використовуються для оновлення ваг за допомогою алгоритмів оптимізації, таких як стохастичний градієнтний спуск (SGD).

Щоб покращити узагальнення та запобігти надмірному пристосуванню, часто використовують методи доповнення даних. Вони передбачають застосування таких перетворень, як обертання, обрізання та перевертання, щоб штучно збільшити навчальний набір даних. Методи регуляризації, такі як відсіювання, які випадковим чином деактивують нейрони під час навчання, також сприяють запобіганню надмірному навчанню.

Хоча CNN чудово справляються з різними завданнями, пов'язаними із зображеннями, вона має певні обмеження. Для ефективного навчання потрібна значна кількість мічених даних, а більш складні архітектури можуть вимагати значних обчислювальних затрат. Дослідники шукають шляхи вирішення цих проблем, включаючи оптимізацію архітектури, ефективне навчання на основі даних і методи інтерпретації моделей.

Згорткові нейронні мережі зробили революцію в розумінні зображень, дозволивши комп'ютерам автоматично навчатися і виокремлювати особливості зображень. Їхня ієрархічна архітектура і здатність вловлювати складні патерни проклали шлях до значного прогресу в комп'ютерному зорі.

Трансферне навчання. Трансферне навчання - це потужна техніка в машинному навчанні та глибокому навчанні, яка передбачає застосування знань,

отриманих під час виконання одного завдання, для покращення продуктивності при виконанні пов'язаного, але іншого завдання. Він ґрунтується на ідеї, що особливості та уявлення, засвоєні моделлю при виконанні одного завдання, можуть бути корисними для вирішення іншого завдання. Трансферне навчання набуло величезної популярності завдяки своїй здатності прискорювати навчання моделей, покращувати узагальнення та вирішувати проблему дефіциту даних.

Типи трансферного навчання:

1. Індуктивне трансферне навчання: Це найпоширеніша форма навчання на основі перенесення. Воно передбачає навчання моделі на вихідній області, а потім застосування її до цільової області. Вихідна і цільова області мають певну базову схожість або структуру, що дозволяє засвоєним знанням бути корисними в новому контексті.

2. Трансдуктивне трансферне навчання: У цьому підході цільова область має певні марковані дані, а модель має на меті узагальнити ці обмежені марковані дані, щоб зробити прогнози щодо немаркованих даних з тієї ж самої області.

3. Неконтрольоване навчання з перенесенням: Це передбачає попереднє навчання моделі на великому наборі даних без міток, що дозволяє їй ефективно вивчати корисні функції. Ці попередньо навчені функції потім можна точно налаштувати на меншому наборі даних з мітками для цільової задачі.

Переваги трансферного навчання:

1. Прискорює навчання: Ініціалізуючи модель заздалегідь підготовленими вагами, ви даєте їй фору. Це особливо цінно при роботі з обмеженими обчислювальними ресурсами та часом.

2. Покращує узагальнення: Навчання з перенесенням допомагає моделям краще узагальнювати нові та небачені дані. Знання, отримані з вихідної задачі, діють як форма регуляризації, роблячи модель менш схильною до надмірного пристосування до цільової задачі.

3. Вирішує проблему нестачі даних: Навчання з перенесенням є дуже корисним, коли цільова задача має обмежену кількість мічених даних. Попередньо навчена модель привносить цінні знання з вихідної задачі і допомагає навчатися на менших наборах даних.

Стратегії трансферного навчання:

1. Виділення ознак: У цьому підході нижні шари (екстрактори ознак) попередньо навченої моделі заморожуються, і лише верхні шари перенавчаються для цільової задачі. Попередньо навчена модель діє як екстрактор ознак, фіксуючи загальні ознаки з вихідної області, які можуть бути корисними для цільової задачі.

2. Точне налаштування: Точне налаштування передбачає зміну ваг шарів попередньо навченої моделі, щоб адаптувати їх до цільової задачі. Це часто виконується на верхніх шарах моделі, а нижні шари залишаються фіксованими. Точне налаштування дозволяє моделі вивчати специфічні особливості задачі, зберігаючи знання, отримані з вихідної задачі.

3. Адаптація домену: Коли вихідна та цільова області мають різні розподіли, застосовуються методи адаптації доменів для вирівнювання розподілів ознак між двома областями, що робить перенесення більш ефективним.

Трансферне навчання змінило ландшафт машинного навчання, дозволивши моделям використовувати знання з одного завдання для підвищення ефективності іншого. Воно відіграє ключову роль у вирішенні проблеми нестачі даних, прискоренні навчання та покращенні узагальнення.

Зоровий трансформер (ViTs). ViTs - це новаторський підхід, який застосовує архітектуру трансформаторів до задач розпізнавання зображень. Спочатку розроблені для обробки природної мови, трансформатори продемонстрували чудову адаптивність, розглядаючи складні зображення як послідовності токенів[6]. Трансформатори швидко завоювали популярність у галузі комп'ютерного зору завдяки своїй здатності фіксувати довгострокові залежності, полегшувати крос-модальне навчання та досягати

конкурентоспроможної продуктивності при виконанні різноманітних завдань, пов'язаних із зображеннями.

Традиційно згорткові нейронні мережі (CNN) були основною архітектурою для аналізу зображень, використовуючи їхні ієрархічні можливості вилучення ознак. Однак ViTs пропонують зміну парадигми, розглядаючи зображення як послідовності патчів або токенів, таким чином використовуючи механізм самоуваги, притаманний трансформаторам[7]. Такий підхід дозволяє ViTs моделювати взаємозв'язки між патчами, фіксувати глобальний контекст і полегшувати обмін інформацією по всьому зображенню.

Структуру зорового трансформера зображено на рисунку 1.2.

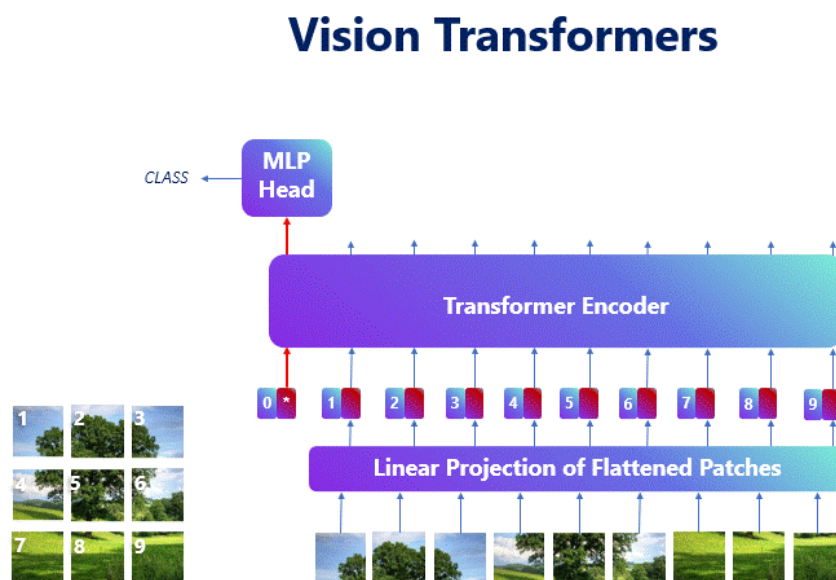


Рисунок 1.2 – Структура зорового трансформера

Ключові компоненти трансформаторів бачення:

1. Вбудовування клаптиків: Зображення розбивається на фрагменти фіксованого розміру, що не перетинаються, які потім лінійно вбудовуються в безперервні вектори. Ці вбудовані ділянки слугують вхідними даними для трансформатора[8].
2. Позиційне вбудовування: Оскільки трансформаторам бракує вродженого просторового розуміння, позиційні вбудовування додаються до вбудовувань патчів, щоб передати просторову інформацію кожного токена.

3. Кодер трансформатора: Серцем Vision Transformer є кодер-трансформер. Він складається з декількох шарів, кожен з яких містить багатоголові механізми самоуваги та нейронні мережі зі зворотним зв'язком.

4. Глобальне об'єднання: Після кодера-трансформера шар глобального усереднення об'єднує представлення токенів в єдиний вектор ознак, який потім використовується для класифікації або інших подальших завдань.

Переваги:

1. Довгострокові залежності: ViTs чудово фіксують довгострокові залежності на зображеннях, що дозволяє їм розуміти складні взаємозв'язки та контекстну інформацію по всьому зображенню.

2. Масштабованість: ViTs можуть обробляти зображення різного розміру без архітектурних модифікацій, що робить їх більш масштабованими порівняно з традиційними CNN.

3. Крос-модальне навчання: Фреймворк-трансформер дозволяє віртуальним терміналам легко інтегруватися з іншими форматами, такими як текст, що сприяє мультимодальному навчанню та передачі знань.

4. Менше архітектурних обмежень: На відміну від CNN, які покладаються на заздалегідь визначені ієрархії вилучення ознак, віртуальні технології потенційно можуть гнучкіше адаптуватися до різних завдань і доменів.

Трансформатори зору являють собою революційне поєднання архітектури трансформаторів і розпізнавання зображень. Їх здатність фіксувати довгострокові залежності, адаптуватися до різних завдань та інтегруватися з іншими модальностями підкреслює їхній потенціал змінити те, як ми розуміємо та аналізуємо зображення.

Рекурентні нейронні мережі (RNN). Рекурентні нейронні мережі - це клас штучних нейронних мереж, спеціально розроблених для обробки послідовних даних шляхом фіксації та моделювання часових залежностей. На відміну від прямих нейронних мереж, які обробляють дані за один прохід, RNN підтримують внутрішній стан, який розвивається в міру того, як вони послідовно отримують нові вхідні дані[9]. Ця унікальна архітектура дозволяє RNN досягати успіху в

задачах, що включають послідовності, таких як обробка природної мови, розпізнавання мови, аналіз часових рядів тощо.

Структуру рекурентної нейронної мережі зображено на рисунку 1.3.

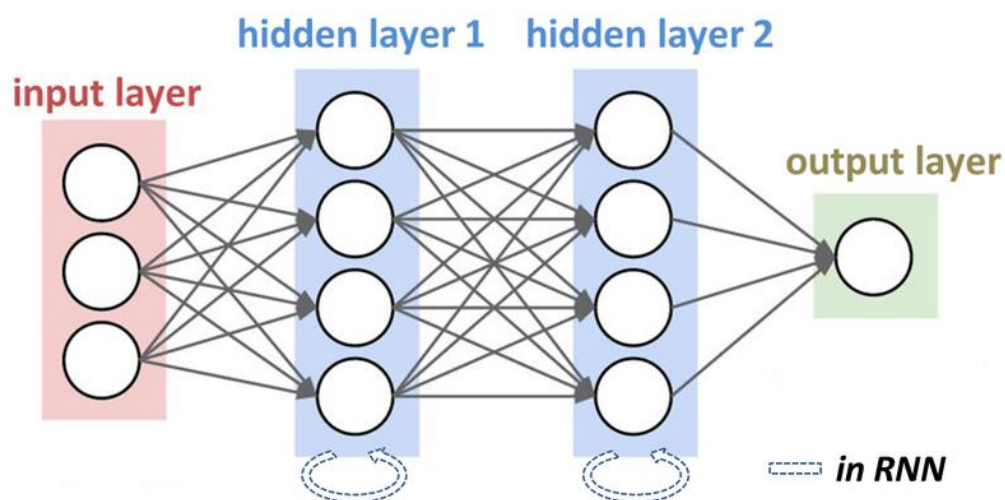


Рисунок 1.3 – Структура рекурентної нейронної мережі

Ключові концепції та архітектура:

1. **Послідовна обробка:** RNN обробляють вхідні послідовності елемент за елементом, що дозволяє їм зберігати пам'ять про минулі вхідні дані[10]. Ця пам'ять полегшує вивчення закономірностей і взаємозв'язків з плином часу.
2. **Приховані стани:** На кожному часовому кроці RNN виробляє вихід і оновлює свій прихований стан, який служить підсумком минулих вхідних даних, отриманих до цього моменту. Цей прихований стан несе інформацію через часові кроки, дозволяючи мережі фіксувати контекст і часові залежності.
3. **Періодичне з'єднання:** Рекурентний зв'язок у RNN дозволяє прихованому стану впливати на вихід на поточному часовому кроці і, в свою чергу, на прихований стан на наступному часовому кроці[11]. Цей механізм дозволяє мережі вивчати послідовні патерни.
4. **Проблема зникаючого градієнта:** Хоча RNN ефективні для моделювання короткострокових залежностей, вони страждають від проблеми зникаючого градієнта, коли градієнти зменшуються в міру того, як вони поширюються

в часі. Це ускладнює вловлювання довгострокових залежностей за допомогою RNN.

Застосування RNN:

1. Обробка природної мови (NLP): NLP широко використовуються в таких задачах, як моделювання мови, машинний переклад, аналіз настроїв і генерація тексту[12].
2. Розпізнавання мови: RNN продемонстрували успіх у розпізнаванні мови, моделюючи часові патерни аудіосигналів.
3. Аналіз часових рядів: RNN є цінним інструментом для прогнозування та аналізу часових рядів даних, таких як ціни на акції, погодні умови та медичні дані.
4. Аналіз відео: RNN можуть аналізувати відеопослідовності для розпізнавання дій, субтитрів до відео та виявлення аномалій.
5. Генерація музики: RNN можуть генерувати музичні послідовності, враховуючи їхню здатність до навчання та генерування послідовних патернів.

Рекурентні нейронні мережі відіграли значну роль у поглибленні нашого розуміння послідовних даних і часових зв'язків. Їх здатність фіксувати закономірності та залежності в часі призвела до застосування в різних сферах, від обробки природної мови до аналізу часових рядів.

1.2 Дорожні знаки України

Дорожній знак – стандартизований графічний малюнок, що встановлюється край дороги для повідомлення визначеної інформації учасникам дорожнього руху, один з засобів регулювання дорожнього руху[13]. Дорожні знаки України регулюються поєднанням норм, встановлених Віденською конвенцією про дорожні знаки й сигнали, Європейським Союзом та Міністерством інфраструктури України. Дорожні знаки в Україні поділяються на 7 груп:

- попереджувальні знаки;
- знаки пріоритету;

- заборонні знаки;
- наказові знаки;
- інформаційно-вказівні знаки;
- знаки сервісу;
- додаткові знаки.

Попереджувальні знаки зазвичай є гостроверхими червоними трикутниками з білим фоном та чорними значками (рис. 1.4). Жовтий фон використовуються для тимчасової небезпеки чи обумовлені дорожніми роботами. Знаки можуть включати додаткові таблички з подробицями небезпеки, до якої відносяться знаки чи містять іншу необхідну інформацію.



Рисунок 1.4 – Попереджувальні знаки

Знаки пріоритету регулюють порядок руху транспортних засобів. Знак головної дороги підказує водіям, що вони мають переважне право на усіх перехрестях попереду на дорозі до кінця дії знаку. Дорожні знаки переваги в русі використовуються там, де дорога занадто вузька, щоб дозволити рух транспортних засобів пліч-о-пліч, а лише по чергово. Приклад знаків пріоритету зображено на рисунку 1.5.



Рисунок 1.5 – Знаки пріоритету

Заборонні знаки мають форму кола з червоною облямівкою. Вони забороняють певні дії на дорозі, наприклад, зупинку, стоянку, обгін або рух у певному напрямку. Приклад заборонних знаків зображено на рисунку 1.6.



Рисунок 1.6 – Заборонні знаки

Наказові знаки інструктують водіїв на дії, які вони повинні прийняти чи підкоритися, або можуть позначати типи транспортних засобів яким дозволяється використовувати дорогу.

Інформаційно-вказівні знаки мають форму прямокутника або квадрата. Вони інформують про напрямки руху, розташування об'єктів і відстані до них.

Знаки сервісу позначають шлях до дорожніх служб, які можуть знадобитися водію. Вони можуть включати стрілки, відстані чи назви служб.

Додаткові знаки додають інформацію до попереджувальних, заборонних, наказових і інформаційно-вказівних знаків. Вони мають форму квадрата або прямокутника.

Розпізнавання дорожніх знаків має велику важливість в сучасному світі дорожнього руху. Водії повинні бути в змозі швидко і точно розпізнавати дорожні знаки, щоб уникнути аварій. Це є ключовим елементом для забезпечення безпеки на дорогах та ефективної організації транспортного потоку.

1.3 Аналіз існуючих систем, що застосовуються для розпізнавання дорожніх знаків

Інформаційна система ідентифікації дорожніх знаків складається з наступних апаратних і програмних компонентів:

1. Оптична камера для зйомки зображення.
2. Апаратний модуль захоплення.
3. Модуль виявлення.
4. Модуль класифікації.
5. База даних.

Спершу, зображення з камери надсилається до системи. Потім використовується певний алгоритм для визначення місцезнаходження дорожнього знаку. Після цього програма розпізнавання здійснює розпізнавання знаку. Структура бази даних може варіюватися залежно від завдань, поставлених перед системою.

Система "Opel Eye" - це інноваційна технологія, яка використовується в автомобілях Opel (Vauxhall в Великобританії) з метою поліпшення безпеки та допомоги водіям під час їзди[14]. Вона включає в себе різноманітні функції та датчики для моніторингу дорожньої обстановки та надання важливої інформації водіям. На рисунку 1.7 зображено приклад роботи Opel Eye.



Рисунок 1.7 – Приклад роботи Opel Eye

Основні функції системи "Opel Eye" можуть включати:

1. Розпізнавання дорожніх знаків. Система може автоматично розпізнавати дорожні знаки, такі як обмеження швидкості, заборони на обгін та інші. Вона може відображати відповідні індикації на панелі приладів.
2. Адаптивний круїз-контроль. Ця функція дозволяє автомобілю автоматично підтримувати безпечну відстань від інших транспортних засобів на дорозі, регулюючи швидкість.
3. Попередження про виїзд за межі полоси руху. Система може виявляти, коли автомобіль виїжджає за межі своєї полоси руху без включеного поворотника, і видає попередження водію.
4. Система контролю сліпих зон. Датчики в системі "Opel Eye" можуть виявляти транспортні засоби, які знаходяться у сліпих зонах, і видають попередження водію, щоб уникнути небезпеки під час зміни полоси.
5. Система автоматичного екстреного гальмування. У деяких моделях система може автоматично активувати гальма для уникнення зіткнення в умовах, коли виявлена небезпека зіткнення.

Speed Limit Assist (SLA) - це система, яка доступна в деяких моделях автомобілів Mercedes-Benz і призначена для сприяння дотриманню обмежень швидкості на дорозі[15]. Ця система допомагає водіям бути усвідомленими щодо дорожніх знаків та вказівок обмеження швидкості, допомагаючи уникнути

непередбачених перевищень швидкості. Приклад роботи системи Speed Limit Assist зображено на рисунку 1.8.



Рисунок 1.8 – Speed Limit Assist

Основні риси Speed Limit Assist можуть включати:

1. Розпізнавання дорожніх знаків. Система може використовувати вбудовані камери та датчики для виявлення дорожніх знаків, що обмежують швидкість. Вона може визначити дійсні обмеження швидкості та відображати їх на панелі приладів або на бортовому комп'ютері.
2. Попередження водія. Якщо виявлено дорожній знак із зміною обмеження швидкості, система може надати відповідне попередження водію, допомагаючи уникнути непередбаченим перевищенням швидкості.
3. Адаптивний круїз-контроль. В деяких моделях SLA може бути інтегрований з адаптивним круїз-контролем, що дозволяє автомобілю автоматично підтримувати безпечну відстань до інших транспортних засобів на дорозі з урахуванням обмежень швидкості.

Road sign information (RSI) – це функція безпеки, яку Volvo пропонує в багатьох своїх автомобілях. Вона використовує камери та датчики для

розпізнавання дорожніх знаків і надає водіям інформацію про дорожні знаки, з якими вони стикаються, в режимі реального часу[16]. Ця система допомагає підвищити рівень обізнаності водія та дотримання ним правил дорожнього руху. Приклад роботи RSI зображено на рисунку 1.9.



Рисунок 1.9 – Приклад роботи RSI

Основні особливості інформації про дорожні знаки в автомобілях Volvo можуть бути такими:

1. Розпізнавання знаків. Система може розпізнавати різні дорожні знаки, такі як обмеження швидкості, знаки заборони в'їзду, знаки зупинки тощо. Вона ідентифікує ці знаки за допомогою візуальних підказок за допомогою бортових камер.
2. Відображення для водія. При виявленні дорожнього знаку відповідна інформація часто відображається на приладовій панелі або на головному дисплеї автомобіля, забезпечуючи водієві легкий доступ до інформації, не відволікаючись від дороги.

3. Попередження про обмеження швидкості. Система може порівняти виявлене обмеження швидкості з поточною швидкістю автомобіля і попередити водія про перевищення встановленого обмеження, допомагаючи водієві дотримуватися безпечної та дозволеної швидкості.

4. Динамічні оновлення. Деякі моделі Volvo з RSI можуть оновлювати інформацію про дорожні знаки в режимі реального часу відповідно до умов, що змінюються, наприклад, змінні обмеження швидкості в зонах будівництва або в шкільних зонах.

5. Пам'ять дорожніх знаків. Деякі моделі Volvo, оснащені системою RSI, можуть зберігати та відображати найсвіжішу інформацію про дорожні знаки на дисплеї водія, допомагаючи водієві залишатися поінформованим, навіть якщо знак більше не видно.

Volkswagen Sign Assist - це технологічна система, що використовується в автомобілях Volkswagen для розпізнавання дорожніх знаків та відображення інформації про них на панелі приладів[17]. Ця система допомагає водіям бути більш усвідомленими щодо дорожнього середовища та дотримуватися дорожніх правил. Приклад роботи Volkswagen Sign Assist зображено на рисунку 1.10.



Рисунок 1.10 – Приклад роботи Volkswagen Sign Assist

Основні риси Volkswagen Sign Assist можуть включати:

1. Розпізнавання дорожніх знаків: Система використовує камеру, розташовану зазвичай біля внутрішнього дзеркала заднього виду, для виявлення різних дорожніх знаків, таких як обмеження швидкості, заборони та інші.
2. Відображення інформації: Інформація про розпізнані дорожні знаки відображається на панелі приладів або на бортовому екрані автомобіля. Водії можуть бачити актуальні дані про швидкісні обмеження та інші знаки без потреби звертати увагу на дорожні знаки на обочині.
3. Додаткові попередження: У деяких випадках, коли виявляється перевищення швидкості, система може надати водієві попередження, наприклад, візуальне або звукове сповіщення.
4. Актуальність інформації: Інформація відображається в реальному часі, що дозволяє водіям вчасно реагувати на зміни дорожніх обмежень.

Порівняльна характеристика існуючих систем розпізнавання зображена у таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих систем розпізнавання

Параметр	Opel Eye	Speed Limit Assist	Road sign information	Volkswagen Sign Assist
Заявлена точність розпізнавання	90%	95%	96%	95%
Освітленість знака не менше	50	50	50	50
Розпізнавання знаків обмеження швидкості	Так	Так	Так	Так
Розпізнавання інших заборонних знаків	Так	Ні	Так	Так
Реальна точність розпізнавання	75%	70%	75%	95%

З таблиці 1.1 видно, що окремо впроваджені інформаційні системи для розпізнавання дорожніх знаків зустрічаються досить рідко, більшість систем не є окремо встановленою функцією, а здебільшого входять до складу заводської комплектації автомобіля. Звідси випливає, що їхня комерційна вартість занадто висока. Всі встановлені системи показують невеликий відсоток розпізнавання дорожніх знаків, відсоток розпізнавання падає, якщо дорожній знак знаходиться не в ідеальних умовах (нахил, часткове перекриття, забруднення знаку), оскільки зображення піддаються конструктивних спотворень. Під час тестування надійно і точно розпізнаються тільки чисті знаки надійно і точно розпізнаються тільки чисті знаки з контрастним контуром. З'ясувати, які алгоритми лежать в основі цих систем неможливо, оскільки ця інформація не знаходиться у вільному доступі.

Розробка методу ідентифікації дорожніх знаків, стійких до таких перетворень, дозволить підвищити точність розпізнавання дорожніх знаків, стійких до таких перетворень, дозволить підвищити точність розпізнавання, що є актуальним при:

- нахилених дорожніх знаків;
- маневруванні автомобіля в зоні ідентифікації;
- високій швидкості руху транспортного засобу.

1.4 Постановка задачі

У сучасному світі зростає значення автономних транспортних засобів та систем підтримки водіїв. Однією з ключових технологій у цьому контексті є розпізнавання дорожніх знаків. Головною метою дослідження є розробка та імплементація програмної системи, яка може автоматично розпізнавати дорожні знаки на цифрових зображеннях. Для досягнення цієї мети необхідно вирішити такі завдання:

1. Зібрати та підготувати датасет зображень дорожніх знаків, включаючи різні класи та варіації.

2. Розробити архітектуру згорткової нейронної мережі для розпізнавання дорожніх знаків.
3. Реалізувати обробку зображень, включаючи попередню обробку, аугментацію та нормалізацію даних.
4. Навчити згорткову нейронну мережу на підготовленому датасеті та провести експерименти для встановлення найкращих параметрів моделі.
5. Оцінити ефективність розробленої моделі за допомогою метрик точності, чутливості та специфічності на тестових даних.

Дорожні знаки можуть бути різних розмірів, кольорів та форм. Вони можуть бути розташовані на різних відстанях та під різними кутами до камери спостереження. Система повинна бути здатна розпізнавати ці знаки навіть при варіабельних умовах освітлення та складних фонових об'єктах.

В рамках цього дослідження, буде розроблена згорткова нейронна мережа для розпізнавання дорожніх знаків. Вона буде навчена на великому наборі зображень з різних дорожніх сцен та умов.

Очікується, що розроблена програмна система буде здатна розпізнавати дорожні знаки з високою точністю та ефективністю навіть у реальних умовах дорожнього руху. Успішне виконання цього дослідження може сприяти покращенню систем автоматичного розпізнавання знаків та зробити їх використання більш надійним та ефективним.

1.5 Вибір засобів реалізації програмного забезпечення

Вибір мови програмування є ключовим аспектом при розробці програмного забезпечення, особливо в області машинного навчання та обробки зображень. Вибір мови Python для реалізації програмного забезпечення розпізнавання дорожніх знаків має наступні причини[18]:

1. Простота та Зручність. Python має простий та лаконічний синтаксис, що полегшує процес розробки, читання та розуміння коду. Це робить його відмінним вибором для великої кількості розробників з різним досвідом.

2. Багатофункціональність. Python має багатий екосистему бібліотек для різних завдань, включаючи машинне навчання та обробку зображень. Бібліотеки, такі як TensorFlow, PyTorch, scikit-learn, OpenCV, надають засоби для побудови та навчання нейронних мереж, а також обробки зображень[19].

3. Швидкість Розробки. Дані бібліотеки для машинного навчання та обробки зображень надають готові реалізації багатьох алгоритмів та моделей, що значно спрощує і прискорює процес розробки.

4. Велика кількість Ресурсів. Освітні матеріали, документація, онлайн-курси та підручники для машинного навчання та Python є широко поширеними, що полегшує вивчення та вдосконалення ваших навичок.

5. Кросплатформенність. Python є кросплатформенною мовою, що означає, що програмне забезпечення, написане на ньому, може працювати на різних операційних системах без значних змін.

Вибір середовища розробки є важливим етапом при створенні програмного забезпечення. Одним із популярних та обґрунтованих виборів для реалізації програмного забезпечення для розпізнавання дорожніх знаків є середовище PyCharm[20]. Ось причин, чому PyCharm є чудовим вибором:

1. Інтеграція з Python і бібліотеками глибокого навчання: PyCharm ідеально підходить для роботи з Python, який є провідною мовою для розробки нейронних мереж. Це середовище забезпечує легку інтеграцію з популярними бібліотеками глибокого навчання, такими як TensorFlow, Keras та PyTorch, які часто використовуються для створення CNN.

2. Підтримка Jupyter Notebook: PyCharm підтримує Jupyter Notebooks, що є важливим для розробки та тестування моделей машинного навчання, оскільки вони дозволяють візуалізувати дані, процес навчання та результати моделі в одному місці.

3. Налаштування та профілювання: PyCharm пропонує розширені можливості для налаштування та профілювання коду, що є ключовим для оптимізації продуктивності та ефективності нейронних мереж[21].

4. Інтегроване середовище розробки (IDE): PyCharm надає все необхідне в одному місці: від редактора коду до інструментів для рефакторингу та системи контролю версій, що забезпечує ефективність робочого процесу.

5. Плагіни та інтеграція з іншими інструментами: Можливість інтеграції з різними інструментами та службами, а також наявність широкого спектру плагінів, дозволяє розширити функціонал PyCharm згідно з потребами проекту.

6. Підтримка віртуальних середовищ: PyCharm забезпечує легке управління віртуальними середовищами Python, що є важливим для ізоляції залежностей проекту та забезпечення його портативності.

7. Спільнота та документація: Велика спільнота користувачів та доступність документації забезпечують легкість вирішення потенційних проблем та навчання нових функцій середовища.

Загалом, PyCharm є міцним інструментом, який може ефективно підтримувати розробку складних проектів

Для досягнення найкращих результатів під час навчання нейронної мережі було використано навчальний набір даних German Traffic Sign Recognition Benchmark (GTSRB)[22]. Цей набір складається з близько 51 839 зображень, розміри яких варіюються від 15x15 до 250x250 пікселів. Кожне зображення містить один із 43 видів дорожніх знаків. Вже перед використанням дані було розділено в співвідношенні 1:3 на тестові та тренувальні набори відповідно. Тренувальні дані були впорядковані за класами, які вони представляють, і розміщені у 43 папках.

GTSRB містить численні приклади знаків одного типу, які відрізняються розміром, інтенсивністю кольору, кутом нахилу та поворотом на зображенні. Зображення, представлені у GTSRB, вважаються позитивними прецедентами, оскільки містять зображення дорожнього знака. З іншого боку, негативними прецедентами є ті зображення, які не містять дорожніх знаків. В наборі GTSRB також присутні кадри, що не мають зображень дорожніх знаків, але можуть бути

помилково визнані як такі. Ці кадри використовуються для відмежування позитивних від негативних прецедентів.

Висновки до розділу 1

Даний розділ був присвячений огляду основної проблематики кваліфікаційної роботи. Також було докладно розглянуто технологічні аспекти проекту, зокрема, використання згорткових нейронних мереж (CNN) для обробки зображень. Згорткові нейронні мережі є оптимальним вибором для завдань комп'ютерного зору завдяки їх здатності ефективно виявляти та класифікувати об'єкти на зображеннях. Було обговорено різні архітектури CNN, їх переваги та недоліки, а також особливості їх тренування та оптимізації.

Також в даному розділі було проведено аналіз існуючих систем розпізнавання дорожніх знаків, проведено порівняльну характеристику їх можливостей.

На основі проведеного дослідження було складено постановку завдання та вибрані засоби реалізації програми.

РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ НАВЧАННЯ НЕЙРОННИХ МОДЕЛЕЙ

2.1 Опис алгоритму розпізнавання знаків за допомогою CNN

Розпізнавання дорожніх знаків за допомогою згорткових нейронних мереж (CNN) включає декілька ключових етапів[23]:

1. Попередня обробка даних:

- Збір даних: Збираються зображення дорожніх знаків із різних джерел для тренування мережі.

- Нормалізація: Зображення масштабуються до стандартного розміру і нормалізуються для зменшення варіативності освітлення та кольору.

2. Конфігурація архітектури CNN[24]:

- Вхідний шар: Перший шар CNN приймає попередньо оброблене зображення.

- Згорткові шари (Convolutional layers): Використовуються для виявлення основних характеристик зображень, таких як краї, форми, текстури.

- Пулінг шари (Pooling layers): Зменшують розміри вихідних даних із згорткових шарів, зберігаючи при цьому найважливіші особливості.

- Повнозв'язні шари (Fully connected layers): Використовуються для класифікації знаків на основі виявлених характеристик.

- Вихідний шар: Останній шар використовує функцію softmax для багатокласової класифікації, надаючи ймовірності для кожного класу (типу дорожнього знаку).

3. Тренування мережі:

- Ваги і зміщення: Під час тренування алгоритм коригує ваги і зміщення в мережі для мінімізації помилок.

- Функція втрат: Визначає, наскільки віддалений прогноз мережі від реального результату.

- Оптимізатор: Використовується для оновлення вагів у напрямку зменшення помилок.

4. Валідація і тестування:

- Перехресна перевірка: Мережу перевіряють на окремому наборі даних, щоб забезпечити її здатність до узагальнення.
- Тестування: Остаточне тестування мережі на незнайомих зображеннях для оцінки її ефективності.

5. Розгортання:

- Після тренування і валідації, модель може бути розгорнута в реальних системах, наприклад, в автономних транспортних засобах.

Цей процес використовує здатність згорткових нейронних мереж автоматично та адаптивно вивчати просторові ієрархії ознак із зображень, що робить їх високоефективними для задач розпізнавання зображень, таких як розпізнавання дорожніх знаків.

2.2 Аналіз вхідних даних для проведення навчання

Для розробки нейронної моделі розпізнавання дорожніх знаків можна використати набір даних Traffic Signs Preprocessed на Kaggle [25]. Це колекція зображень дорожніх знаків, призначена для використання в машинному навчанні, зокрема в задачах розпізнавання дорожніх знаків. Він містить попередньо оброблені зображення, що може бути корисним для прискорення процесу навчання та покращення продуктивності моделі. Набір даних призначений для підтримки розробки та тестування алгоритмів у галузі комп'ютерного зору, зокрема для додатків, пов'язаних з виявленням і розпізнаванням дорожніх знаків у різних умовах і середовищах. Вихідними даними для даного набору є німецькі стандарти розпізнавання дорожніх знаків (GTSRB).

Зображення в наборі даних були попередньо оброблені та збережені у дев'яти файлах pickle. Перед попередньою обробкою навчальний набір даних було вирівняно, зробивши приклади в класах однаковими, як показано на рисунку 2.1 та рисунку 2.2. Гістограма 43 класів для навчального набору даних з кількістю прикладів для класифікації дорожніх знаків до і після вирівнювання

шляхом додавання перетворених зображень (яскравість і поворот) з вихідного набору даних. Після вирівнювання навчальний набір збільшився до 86989 прикладів.

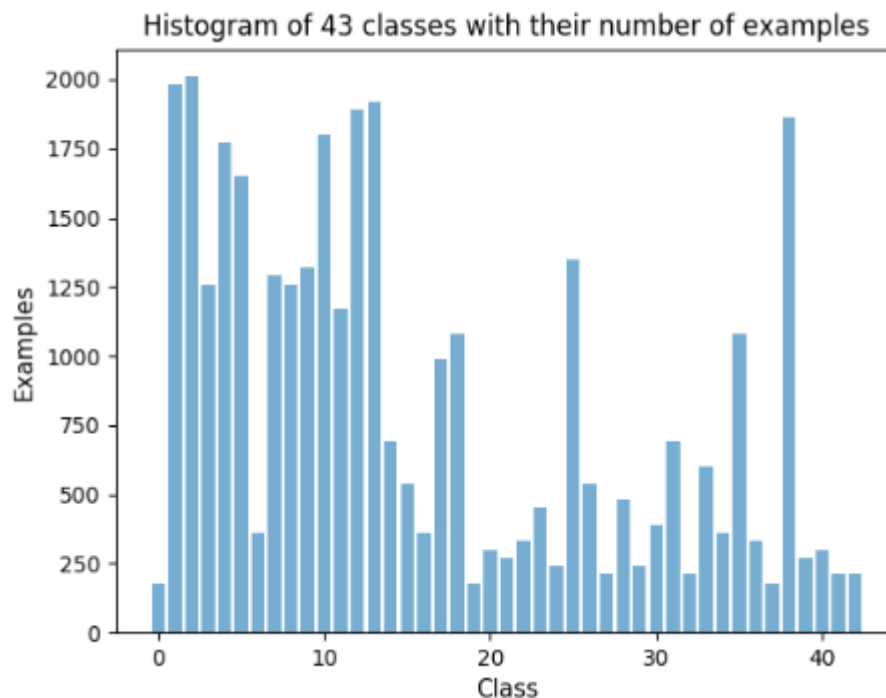


Рисунок 2.1 – Гістограма до вирівнювання

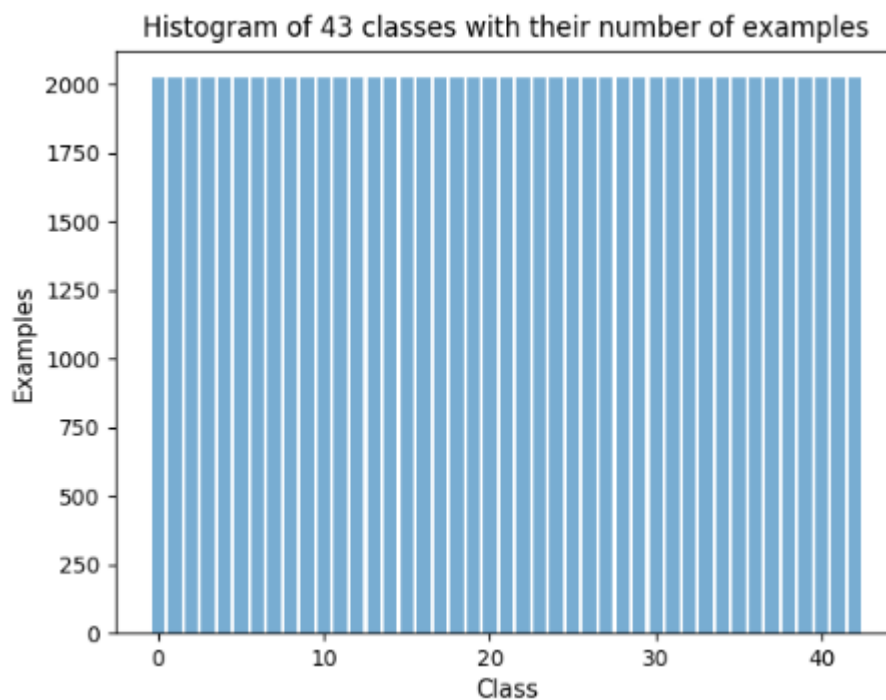


Рисунок 2.2 – Гістограма після вирівнювання

Результати попередньої обробки дев'яти файлів виглядають наступним чином:

- data0.pickle – Shuffling;
- data1.pickle - Shuffling, /255.0 Normalization;
- data2.pickle - Shuffling, /255.0 + Mean Normalization;
- data3.pickle - Shuffling, /255.0 + Mean + STD Normalization;
- data4.pickle - Grayscale, Shuffling;
- data5.pickle - Grayscale, Shuffling, Local Histogram Equalization;
- data6.pickle - Grayscale, Shuffling, Local Histogram Equalization, /255.0 Normalization;
- data7.pickle - Grayscale, Shuffling, Local Histogram Equalization, /255.0 + Mean Normalization;
- data8.pickle - Grayscale, Shuffling, Local Histogram Equalization, /255.0 + Mean + STD Normalization.

Для нейронної моделі розпізнавання дорожніх знаків було обрано файл data2.pickle.

Shuffling (Перемішування): Це означає, що порядок зображень у наборі даних був випадковим. Перемішування є поширеною практикою в машинному навчанні, щоб гарантувати, що модель не засвоїть ненавмисні закономірності, виходячи з порядку даних.

/255.0: Це означає, що значення пікселів зображень, які зазвичай знаходяться в діапазоні від 0 до 255, було поділено на 255. Цей процес масштабує значення пікселів до діапазону від 0 до 1 - крок, відомий як нормалізація. Нормалізація даних до цього діапазону часто робиться для покращення продуктивності нейронних мереж.

Mean Normalization (Нормалізація середнього значення): Це процес, коли від кожного пікселя віднімається середнє значення інтенсивності пікселя[26]. Цей тип нормалізації центрує значення пікселів навколо нуля, що також може бути корисним для навчання нейронних мереж.

Набір даних data2.pickle містить RGB зображення. Форма даних наступна:

- x_train: (86989, 3, 32, 32)
- y_train: (86989,)
- x_validation: (4410, 3, 32, 32)
- y_validation: (4410,)
- x_test: (12630, 3, 32, 32)
- y_test: (12630,)

x_train, x_validation, x_test: Це масиви зображень. Числа показують кількість зображень у кожній підмножині (навчальній, валідаційній, тестовій) та їх розмірність. Наприклад, x_train: (86989, 3, 32, 32) означає, що є 86 989 навчальних зображень, кожне з яких має 3 колірні канали (RGB) і роздільну здатність 32x32 пікселі.

y_train, y_validation, y_test: Ці масиви містять мітки, що відповідають зображенням у наборах даних x. Числа вказують на загальну кількість міток у кожній підмножині, що відповідає кількості зображень. Наприклад, y_train: (86989,) означає, що у навчальному наборі 86 989 міток.

Завантаження набору даних data2.pickle зображено на рисунку 2.3.

```
# Відкриття файлу для читання
with open('input/Traffic Signs Preprocessed/data2.pickle', 'rb') as f:
    data = pickle.load(f, encoding='latin1') # dictionary type

# Підготовка y_train та y_validation для використання в Keras
data['y_train'] = to_categorical(data['y_train'], num_classes=43)
data['y_validation'] = to_categorical(data['y_validation'], num_classes=43)

# Створення каналів у кінці
data['x_train'] = data['x_train'].transpose(0, 2, 3, 1)
data['x_validation'] = data['x_validation'].transpose(0, 2, 3, 1)
data['x_test'] = data['x_test'].transpose(0, 2, 3, 1)

# Відображення завантажених даних з файлу
for i, j in data.items():
    if i == 'labels':
        print(i + ':', len(j))
    else:
        print(i + ':', j.shape)
```

Рисунок 2.3 - Завантаження набору даних data2.pickle

Приклад коду для відображення завантажених даних з файлу зображено на рисунку 2.4.

```
def convert_to_grid(x_input):
    M, H, W, C = x_input.shape
    grid_size = int(np.ceil(np.sqrt(N)))
    grid_height = H * grid_size + 1 * (grid_size - 1)
    grid_width = W * grid_size + 1 * (grid_size - 1)
    grid = np.zeros((grid_height, grid_width, C)) + 255
    next_idx = 0
    y0, y1 = 0, H
    for y in range(grid_size):
        x0, x1 = 0, W
        for x in range(grid_size):
            if next_idx < N:
                img = x_input[next_idx]
                low, high = np.min(img), np.max(img)
                grid[y0:y1, x0:x1] = 255.0 * (img - low) / (high - low)
                next_idx += 1
            x0 += W + 1
            x1 += W + 1
        y0 += H + 1
        y1 += H + 1
    return grid

# Візуалізація прикладів навчальних даних
examples = data['x_train'][:81, :, :, :]
print(examples.shape) # (81, 32, 32, 3)

# Побудова прикладів
fig = plt.figure()
grid = convert_to_grid(examples)
plt.imshow(grid.astype('uint8'), cmap='gray')
plt.axis('off')
plt.gcf().set_size_inches(15, 15)
plt.title('Some examples of training data', fontsize=18)
```

Рисунок 2.4 - Відображення завантажених даних з файлу

Приклад завантажених даних зображено на рисунку 2.5.

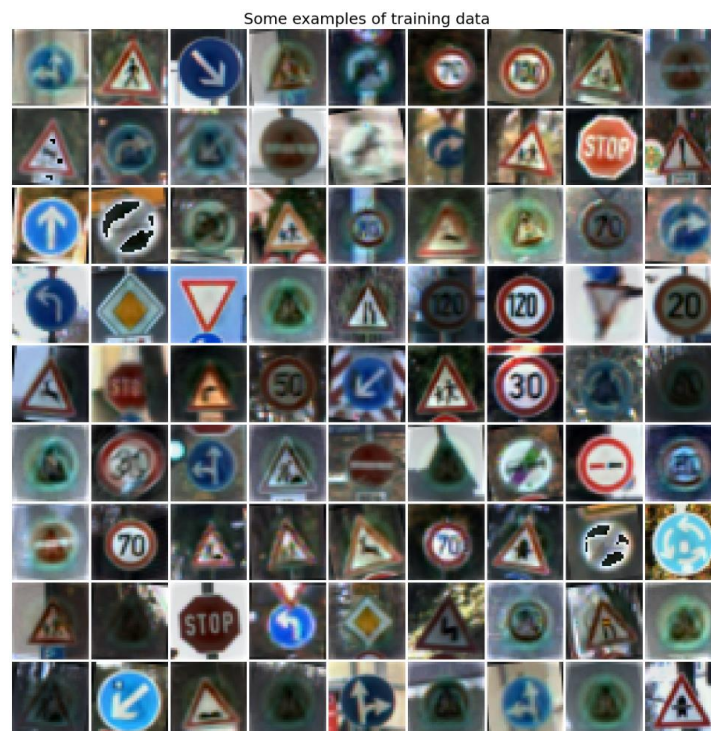


Рисунок 2.5 - Приклад завантажених даних

2.3 Тренування моделі для класифікації дорожніх знаків

Після аналізу набору даних перейдемо до побудови нейронної моделі. Побудова моделі буде відбуватися за допомогою відкритої нейромережної бібліотеки, написаної на мові Python – Keras[27]. Keras містить численні втілення широко вживаних нейромережних будівельних блоків, таких як шари, цільові та передавальні функції, оптимізувальники та безліч інструментів для спрощення роботи із зображеннями та текстом, щоби спрощувати кодування, потрібне для написання глибоко-нейромережного коду[28].

Спочатку будуємо одну модель з фільтрами розміром 3x3 (рис. 2.6). Послідовна модель створюється за допомогою Keras. Вона включає шар згортки (Conv2D), шар максимального об'єднання (MaxPool2D), шар згладжування (Flatten) та два щільні шари[29]. Модель скомпільовано за допомогою оптимізатора Adam і категоричної кросентропії як функції втрат.

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPool2D(pool_size=2))
model.add(Flatten())
model.add(Dense(units=500, activation='relu'))
model.add(Dense(units=43, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Рисунок 2.6 - Модель з фільтрами розміром 3x3

Переоснащення моделі 3x3 невеликою кількістю даних зображено на рисунку 2.7. Визначено планувальник швидкості навчання, який регулює швидкість навчання під час навчання. Модель навчається на підмножині навчальних даних (перші 10 прикладів) з розміром партії 5 протягом певної кількості епох (15). Для моніторингу ефективності також надаються дані валідації. Точність навчання та валідації роздруковується після навчання. Побудова графіка результатів історії для перенавчання на малих даних зображена на рисунку 2.8.

```
annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** (x + epochs))
epochs = 15

h = model.fit(data['x_train'][:10], data['y_train'][:10],
              batch_size=5, epochs=epochs,
              validation_data=(data['x_validation'], data['y_validation']),
              callbacks=[annealer], verbose=1)

print('Epochs={0:d}, training accuracy={1:.5f}, validation accuracy={2:.5f}'.\
      format(*args: epochs, max(h.history['accuracy']), max(h.history['val_accuracy'])))
```

Рисунок 2.7 - Модель з фільтрами розміром 3x3

```
plt.rcParams['figure.figsize'] = (15.0, 5.0) # Встановлення розміру за замовчуванням
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['font.family'] = 'Times New Roman'

fig = plt.figure()
plt.plot(*args: h.history['accuracy'], '-o', linewidth=3.0)
plt.plot(*args: h.history['val_accuracy'], '-o', linewidth=3.0)
plt.title(label: 'Overfitting small data', fontsize=22)
plt.legend(*args: ['train', 'validation'], loc='upper left', fontsize='xx-large')
plt.xlabel(xlabel: 'Epoch', fontsize=20)
plt.ylabel(ylabel: 'Accuracy', fontsize=20)
plt.tick_params(labelsize=18)

plt.show()

fig.savefig('overfitting_small_data.png')
plt.close()
```

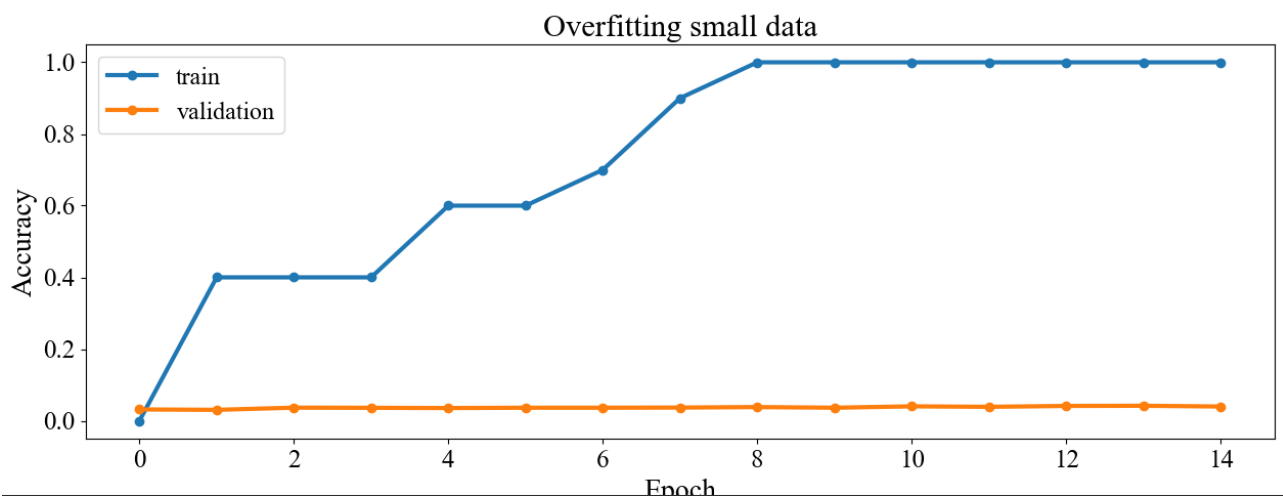


Рисунок 2.8 - Побудова графіка перенавчання на малих даних

Пробуємо будувати різні моделі з різними розмірами фільтрів (рис. 2.9). Той самий процес повторюється з різними розмірами фільтрів у шарі згортки, щоб побачити, як розмір фільтра впливає на продуктивність. Продуктивність кожної моделі оцінюється, а результати візуалізуються і роздруковуються.

```
filters = [3, 5, 9, 13, 15, 19, 23, 25, 31]
model = [0] * len(filters)

for i in range(len(model)):
    model[i] = Sequential()
    model[i].add(Conv2D(filters=32, kernel_size=filters[i], padding='same', activation='relu', input_shape=(32, 32, 3)))
    model[i].add(MaxPool2D(pool_size=2))
    model[i].add(Flatten())
    model[i].add(Dense(units=500, activation='relu'))
    model[i].add(Dense(units=43, activation='softmax'))
    model[i].compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** (x + epochs))
epochs = 5

h = [0] * len(model)

for i in range(len(h)):
    h[i] = model[i].fit(data['x_train'], data['y_train'],
                        batch_size=5, epochs=epochs,
                        validation_data=(data['x_validation'], data['y_validation']),
                        callbacks=[annealer], verbose=0)

print('Model with filters {0:d}x{0:d}, epochs={1:d}, training accuracy={2:.5f}, validation accuracy={3:.5f}'.format(
    args, filters[i], epochs, max(h[i].history['accuracy']), max(h[i].history['val_accuracy'])))
```

Рисунок 2.9 – Побудова різних моделей з різними фільтрами

2.4 Результати проведеного навчання

Побудова графіка результатів порівняння точності зображена на рисунку 2.10. Моделі оцінюються на тестових даних, а точність роздруковується. Час класифікації для кожної моделі вимірюється і друкується. Більші фільтри можуть швидше виокремлювати релевантні ознаки із зображень, що може пояснити їхнє швидке зростання точності навчання. Однак це не завжди ідеально відображається на точності валідації, яка є більш репрезентативною для здатності моделі узагальнювати невидимі дані[30]. Коливання на графіку точності валідації свідчать про те, що може відбуватися деяке перенавчання, особливо для моделей з дуже великими фільтрами. Ці моделі можуть бути

занадто складними і підлаштовуватися під шум у навчальних даних, а не під закономірності, що лежать в їх основі.

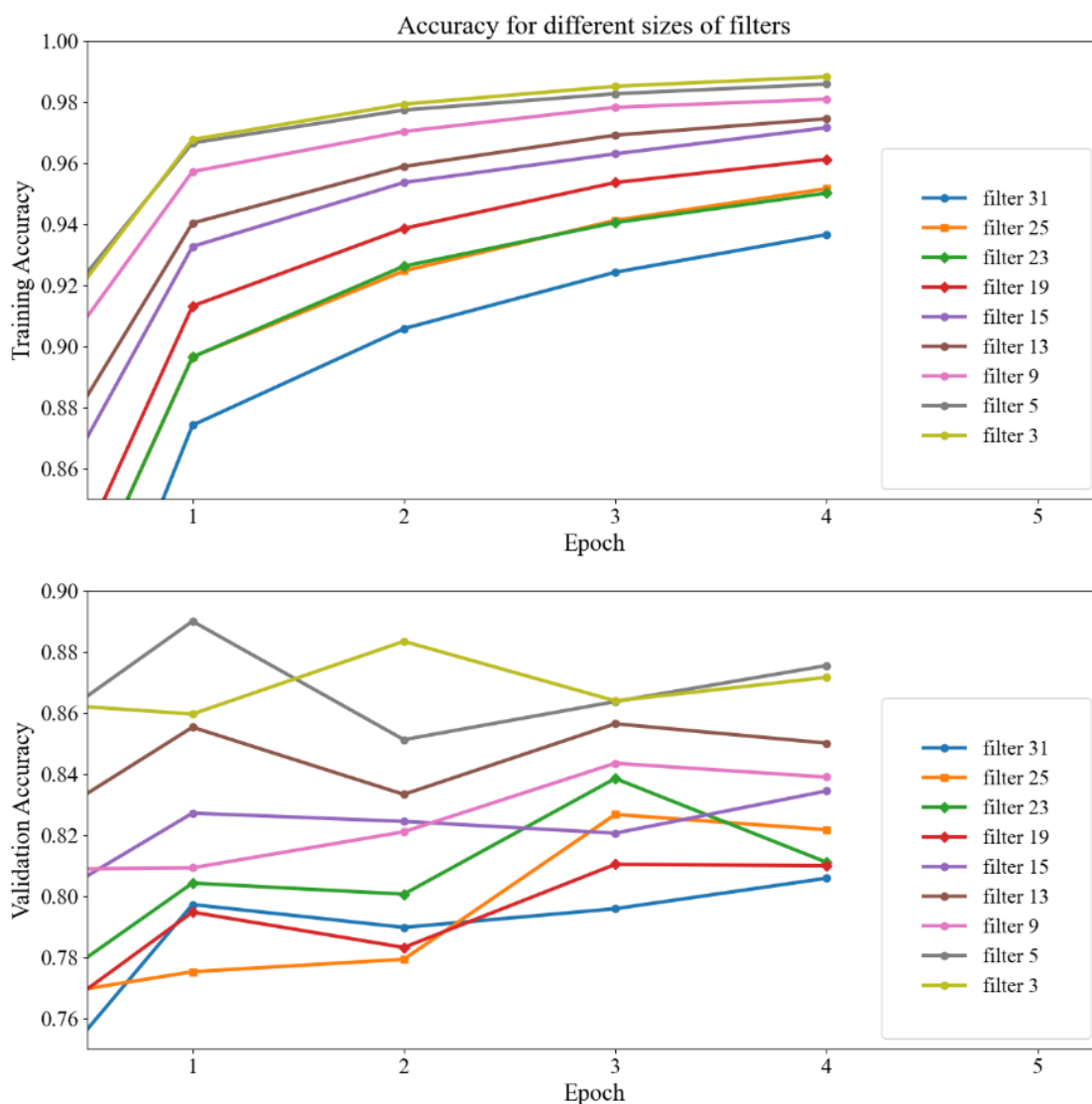


Рисунок 2.10 – Графіки результатів порівняння точності

Вивчені фільтри в кожній моделі візуалізуються, щоб зрозуміти, на яких особливостях фокусується модель. Код для візуалізації фільтрів зображено на рисунку 2.11. Візуалізація навчених фільтрів різних розмірів (від 3x3 до 31x31) з нейронної мережі показують патерни, які згорткові шари навчилися розпізнавати в даних, на яких вони були навчені.

```

for i in range(len(model)):
    w = model[i].get_weights()
    print(w[0].shape)

    # Візуалізація фільтрів
    temp = w[0].transpose(3, 0, 1, 2)
    print(temp.shape)  # (81, 32, 32, 3)

    fig = plt.figure()
    grid = convert_to_grid(temp)
    plt.imshow(grid.astype('uint8'), cmap='gray')
    plt.axis('off')
    plt.gcf().set_size_inches(10, 10)
    name = 'Trained filters ' + str(filters[i]) + 'x' + str(filters[i])
    plt.title(name, fontsize=18)

    plt.show()

    name = 'filters-' + str(filters[i]) + 'x' + str(filters[i]) + '.png'
    fig.savefig(name)
    plt.close()

```

Рисунок 2.11 – Візуалізація фільтрів

На рисунках 2.12-2.20 наведено візуалізацію фільтрів різних розмірів.

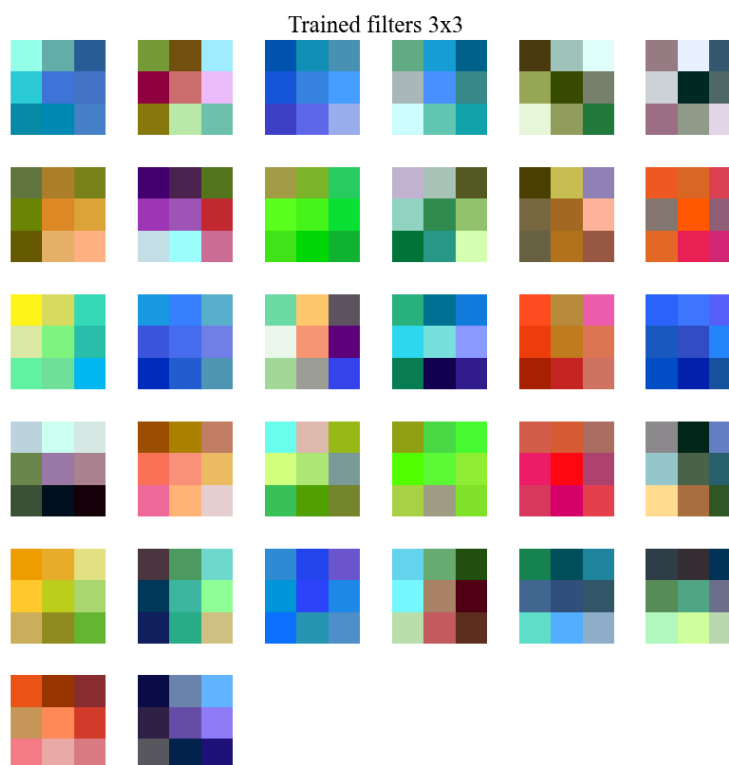


Рисунок 2.12 – Візуалізація фільтру 3x3

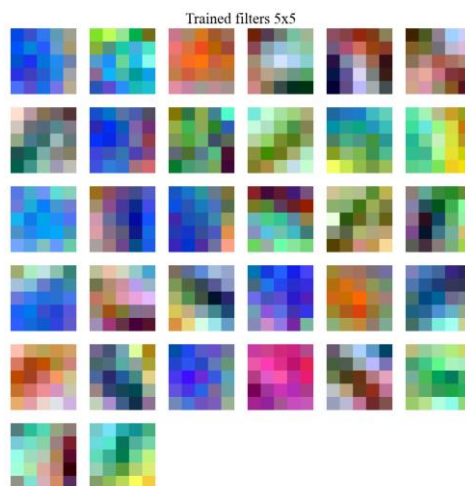


Рисунок 2.13 – Візуалізація фільтру 5x5

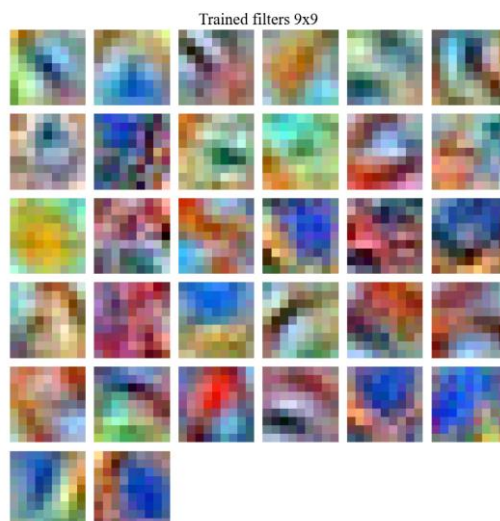


Рисунок 2.14 – Візуалізація фільтру 9x9

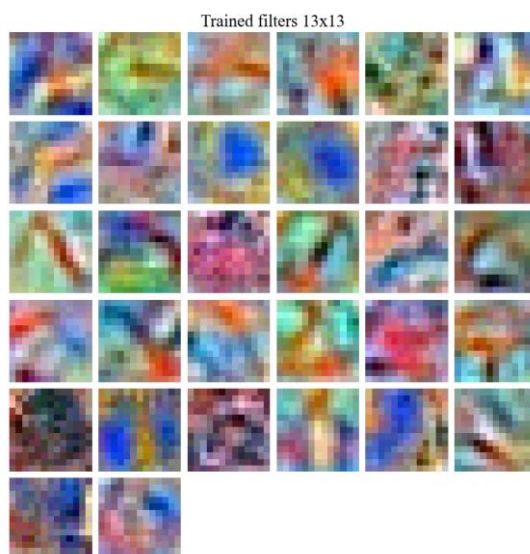


Рисунок 2.15 – Візуалізація фільтру 13x13



Рисунок 2.16 – Візуалізація фільтру 15x15

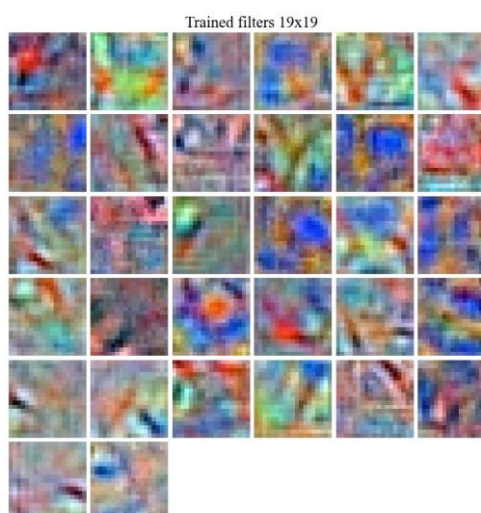


Рисунок 2.17 – Візуалізація фільтру 19x19

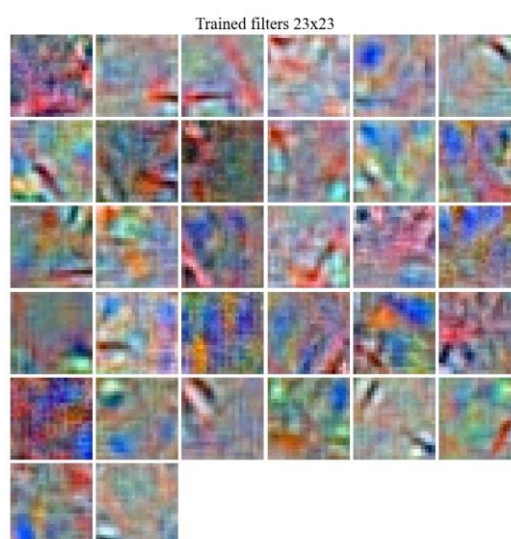


Рисунок 2.18 – Візуалізація фільтру 23x23

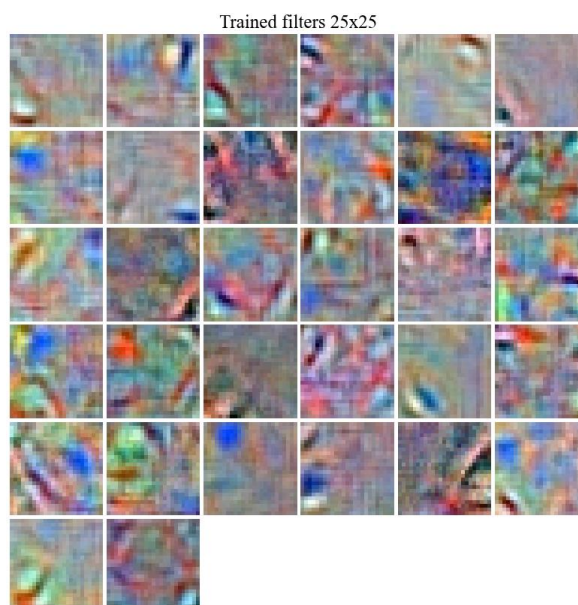


Рисунок 2.19 – Візуалізація фільтру 25x25

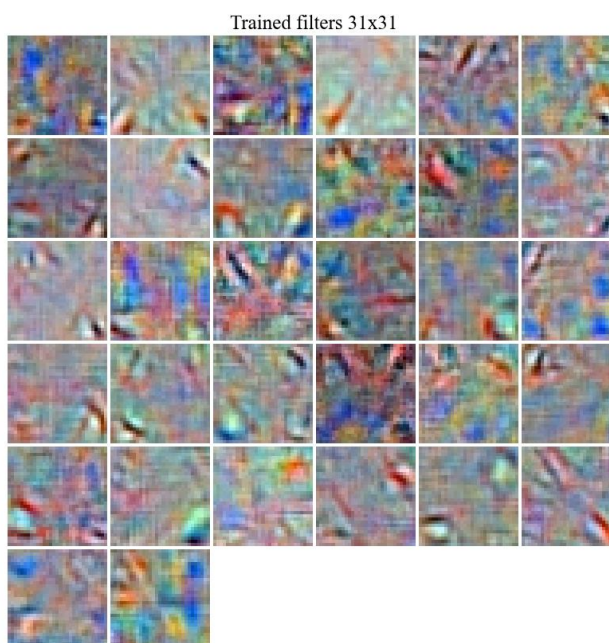


Рисунок 2.20 – Візуалізація фільтру 31x31

Загалом з таких візуалізацій можна спостерігати:

Колірні патерни: Фільтри часто навчаються розпізнавати різні кольорові патерни. Наприклад, деякі фільтри можуть стати чутливими до країв або переходів між кольорами, тоді як інші можуть розпізнавати певні кольорові відтінки, присутні на навчальних зображеннях[31].

Виявлення країв: Менші фільтри, такі як 3x3 і 5x5, часто виконують роль детекторів країв. Вони, як правило, запам'ятовують варіації градієнтів і країв у різних орієнтаціях і можуть бути досить абстрактними.

Текстура і форма: Зі збільшенням розміру фільтра вони здатні захоплювати більш складні патерни, які можуть включати певні текстури або частини об'єктів. Наприклад, фільтри розміром 9x9 або 13x13 можуть почати вловлювати більш специфічні форми або комбінації ребер, які представляють більш складні структури.

Абстрактні об'єкти: Більші фільтри, такі як 19x19 і вище, можуть вловлювати ще більш абстрактні об'єкти, які можуть представляти частини об'єктів, певні текстури або специфічні візерунки, що мають відношення до завдань класифікації. Вони можуть бути не настільки зрозумілими для людського ока, але є важливими для завдання моделі.

Шум і артефакти: Деякі фільтри можуть мати зашумлені або нечіткі шаблони. Це може статися, якщо фільтр не налаштовано належним чином або якщо модель надмірно налаштовано. Це також може бути пов'язано з наявністю шуму в самих навчальних даних.

Надмірність: У фільтрах може бути деяка надмірність, коли кілька фільтрів навчаються виявляти дуже схожі ознаки. Це не обов'язково є неефективним, оскільки може допомогти мережі бути більш стійкою до варіацій вхідних даних.

Чіткість і виразність ознак: Чіткість ознак, представлених кожним фільтром, може дати уявлення про якість процесу навчання. Добре навчені фільтри, як правило, показують чіткі, виразні ознаки, в той час як погано навчені фільтри можуть бути розмитими або нечіткими.

Складність моделі: Про складність моделі іноді можна зробити висновок з візуалізації фільтрів. Більш складні моделі можуть мати фільтри, які захоплюють високорівневі, абстрактні ознаки, тоді як простіші моделі можуть вивчати лише базові ребра та текстури.

Порівняння різних розмірів: Порівняння фільтрів різних розмірів іноді може показати, як збільшення сприйнятливої області (розміру фільтра) впливає на

складність об'єктів, які можна вивчити. Менші фільтри фіксують локальні особливості, тоді як більші фільтри можуть інтегрувати ці локальні особливості в більш глобальні, складні патерни.

Ці візуалізації є корисним діагностичним інструментом для розуміння того, чого навчається модель, а також можуть допомогти у налагодженні та вдосконаленні моделі. Якщо певні фільтри не вивчають корисні властивості, це може вказувати на необхідність коригування процесу навчання або перегляду архітектури моделі.

Висновки до розділу 2

В даному розділі було проведено детальний опис та аналіз процесу навчання нейронної мережі розпізнавання дорожніх знаків. Також проведено аналіз набору даних на якому проводилося навчання моделей. Основна увага в цьому розділі була зосереджена на практичній реалізації згорткової нейронної мережі, яка використовується для обробки та класифікації зображень дорожніх знаків.

Загалом було натреновано 9 моделей з різними фільтрами. Побудовано графіки порівняно точності, які були оцінені на тестових даних. Проведено візуалізацію фільтрів кожної моделі, щоб зрозуміти, на яких особливостях фокусується модель. Результати проведеного навчання показали високу ефективність розроблених моделей.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

3.1 Розробка додатку розпізнавання дорожніх знаків

Після навчання, моделі було збережено до окремих файлів. Також для реалізації виявлення дорожніх знаків на зображенні було використано було використано навчену модель у фреймворку Darknet, яка реалізує це за допомогою бібліотеки OpenCV dnn [32]. Код ініціалізації моделей зображено на рисунку 3.1.

```
# Ініціалізація моделі та інших ресурсів
model = load_model('input/traffic-signs-classification-with-cnn/model-5x5.h5')
with open('input/traffic-signs-preprocessed/mean_image_rgb.pickle', 'rb') as f:
    mean = pickle.load(f, encoding='latin1') # dictionary type

# Читання csv-файлу з назвами міток
# Завантаження двох стовпців [0, 1] у Pandas dataframe
labels = pd.read_csv('input/traffic-signs-preprocessed/label_names.csv')

# Ініціалізація необхідних змінних
path_to_cfg = 'input/traffic-signs-dataset-in-yolo-format/yolov3-ts_test.cfg'
path_to_weights = 'input/trained-traffic-signs-detector-based-on-yolo-v3/yolov3-ts_train_5000.weights'

# Ініціалізація моделі YOLO
network = cv2.dnn.readNetFromDarknet(path_to_cfg, path_to_weights)
network.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
network.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU) # Використання CPU

# Параметри для YOLO
probability_minimum = 0.2
threshold = 0.2

# Генерування кольорів для обмежувальних рамок
colours = np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')
# Отримання назв усіх шарів від мережі
layers_names_all = network.getLayerNames()

# Отримання лише вихідних шарів
layers_names_output = [layers_names_all[i - 1] for i in network.getUnconnectedOutLayers().flatten()]
```

Рисунок 3.1 – Ініціалізація моделей

Для розпізнавання окремого дорожнього знаку розроблена функція process_sign (рис. 3.2).

```
def process_sign(image_path):
    image = Image.open(image_path)
    image = image.resize((32, 32))
    image = np.expand_dims(image, axis=0)
    image = np.array(image)
    pred = model.predict(image)
    pred_class = np.argmax(pred, axis=-1)[0]
    sign = labels.loc[pred_class, 'SignName']
    return sign
```

Рисунок 3.2 – Функція розпізнавання окремого дорожнього знаку

Зображення знаку відкривається та масштабується до розміру 32x32 пікселів, оскільки модель класифікації приймає зображення цього розміру. Зображення перетворюється в масив NumPy, а потім додається додаткова вимірність, щоб воно відповідало формату вхідних даних, який очікує модель. Масив передається в навчену нейронну мережу (яка була завантажена і ініціалізована на початку скрипта) для отримання передбачення. Модель повертає масив передбачень, де кожен елемент масиву відповідає ймовірності того, що зображення належить до конкретного класу дорожнього знаку. Використовуючи `np.argmax`, визначається індекс у масиві передбачень з найвищою ймовірністю, що вказує на клас, до якого, ймовірно, належить знак. Використовуючи отриманий індекс, з датафрейму `labels`, який містить відповідності між індексами та назвами класів, витягується назва класу.

Для обробки зображень з метою виявлення та класифікації дорожніх знаків розроблена функція `process_image` (рис. 3.3). Завантажується зображення за допомогою `cv2.imread(image_path)`, визначаються висота та ширина зображення. Створюються порожні списки для зберігання даних про обмежувальні рамки (`bounding_boxes`), рівні впевненості (`confidences`) та номери класів (`class_numbers`). Зображення перетворюється у "blob" за допомогою `cv2.dnn.blobFromImage`, що є стандартним кроком передбачення в моделях глибокого навчання. Blob передається в модель YOLO[33] (`network.setInput(blob)`), і виконується прогнозування

(network.forward(layers_names_output)). Перебираються кожне виявлення в кожному вихідному шарі, витягуючи інформацію про обмежувальні рамки та рівень впевненості (якщо він вищий за заданий поріг). Використовується cv2.dnn.NMSBoxes для фільтрації набору обмежувальних рамок, видаляючи ті, що перекриваються та залишаючи найкращі за рівнем впевненості. Для кожної обмежувальної рамки, що залишилася після NMS, вирізається відповідна частина зображення. Ця частина перетворюється в blob та передається у модель класифікації дорожніх знаків. На оригінальному зображенні малюються обмежувальні рамки та додаються підписи з назвами та рівнем впевненості для кожного виявленого знака.

```
def process_image(image_path):
    # Завантаження зображення
    image_BGR = cv2.imread(image_path)
    h, w = image_BGR.shape[:2] # Отримання розмірів зображення
    # Списки для зберігання даних про виявлені об'єкти
    bounding_boxes = []
    confidences = []
    class_numbers = []
    # Створення blob із зображення
    blob = cv2.dnn.blobFromImage(image_BGR, 1 / 255.0, size=(416, 416), swapRB=True, crop=False)
    network.setInput(blob) # Встановлення blob як вхідних даних для мережі
    output_from_network = network.forward(layers_names_output)

    # Перебір кожного виявлення
    for result in output_from_network:
        for detected_objects in result:
            scores = detected_objects[5:]
            class_current = np.argmax(scores)
            confidence_current = scores[class_current]
            if confidence_current > probability_minimum:
                box_current = detected_objects[0:4] * np.array([w, h, w, h])
                x_center, y_center, box_width, box_height = box_current
                x_min = int(x_center - (box_width / 2))
                y_min = int(y_center - (box_height / 2))
                bounding_boxes.append([x_min, y_min, int(box_width), int(box_height)])
                confidences.append(float(confidence_current))
                class_numbers.append(class_current)

    results = cv2.dnn.NMSBoxes(bounding_boxes, confidences, probability_minimum, threshold)
    # Перебірка наявності виявлених об'єктів
    if len(results) > 0:
        for i in results.flatten():
            # Координати прямокутника, його ширина та висота
            x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1]
            box_width, box_height = bounding_boxes[i][2], bounding_boxes[i][3]
            # Вирізання фрагменту з дорожнім знаком
            c_ts = image_BGR[y_min:y_min + int(box_height), x_min:x_min + int(box_width), :]
            if c_ts.shape[1] == (0,) or c_ts.shape[1:2] == (0,):
                continue
            else:
                # Обробка фрагменту знака
                blob_ts = cv2.dnn.blobFromImage(c_ts, 1 / 255.0, size=(32, 32), swapRB=True, crop=False)
                blob_ts[0] = blob_ts[0, :, :, :] - mean['mean_image_rgb']
                blob_ts = blob_ts.transpose(0, 2, 3, 1)
                # Класифікація фрагменту
                scores = model.predict(blob_ts)
                prediction = np.argmax(scores)
                # Текст для підпису
                label = labels['signName'][prediction]
                confidence = confidences[i]
                text_box_current = '{}: {:.2f}%'.format('arg: label, confidence * 100)
                # Колір для прямокутника
                colour_box_current = colours[class_numbers[i]].tolist()
                # Малювання прямокутника на оригінальному зображенні
                cv2.rectangle(image_BGR, (x_min, y_min), (x_min + box_width, y_min + box_height), colour_box_current, 6)
                # Додавання тексту до зображення
                cv2.putText(image_BGR, text_box_current, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.9,
                           colour_box_current, thickness=2)

    return image_BGR
```

Рисунок 3.3 – Функція виявлення та класифікації дорожніх знаків

Для обробки відео з метою виявлення та класифікації дорожніх знаків розроблена функція `process_video` (рис. 3.4).

```
def process_video(video_path):
    video = cv2.VideoCapture(video_path)
    writer = None
    h, w = None, None
    plt.rcParams['figure.figsize'] = (3, 3)
    f = 0
    t = 0

    while True:
        ret, frame = video.read()
        if not ret:
            break
        if w is None or h is None:
            h, w = frame.shape[:2]

        blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, size=(416, 416), swapRB=True, crop=False)
        network.setInput(blob)
        start = time.time()
        output_from_network = network.forward(layers_names_output)
        end = time.time()
        f += 1
        t += end - start
        total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
        progress_bar['maximum'] = total_frames

        progress_bar['value'] += 1
        root.update()
        print('Frame number {0} took {1:.5f} seconds'.format(f, end - start))

        bounding_boxes = []
        confidences = []
        class_numbers = []

        for result in output_from_network:
            for detected_objects in result:
                scores = detected_objects[5:]
                class_current = np.argmax(scores)
                confidence_current = scores[class_current]

                if confidence_current > probability_minimum:
                    box_current = detected_objects[0:4] * np.array([w, h, w, h])
                    x_center, y_center, box_width, box_height = box_current
                    x_min = int(x_center - (box_width / 2))
                    y_min = int(y_center - (box_height / 2))
                    bounding_boxes.append([x_min, y_min, int(box_width), int(box_height)])
                    confidences.append(float(confidence_current))
                    class_numbers.append(class_current)

        results = cv2.dnn.NMSBoxes(bounding_boxes, confidences, probability_minimum, threshold)
        if len(results) > 0:
            for i in results.flatten():
                x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1]
                box_width, box_height = bounding_boxes[i][2], bounding_boxes[i][3]

                c_ts = frame[y_min:y_min + int(box_height), x_min:x_min + int(box_width), :]

                if c_ts.shape[1] == (0,) or c_ts.shape[1:2] == (0,):
                    pass
                else:
                    blob_ts = cv2.dnn.blobFromImage(c_ts, 1 / 255.0, size=(32, 32), swapRB=True, crop=False)
                    blob_ts[0] = blob_ts[0, :, :, :] - mean['mean_image_rgb']
                    blob_ts = blob_ts.transpose(0, 2, 3, 1)
                    scores = model.predict(blob_ts)
                    prediction = np.argmax(scores)
                    colour_box_current = colours[class_numbers[i]].tolist()

                    cv2.rectangle(frame, (x_min, y_min),
                                (x_min + box_width, y_min + box_height),
                                colour_box_current, 2)

                    text_box_current = '{:}: {:.4f}'.format('SignName'[prediction],
                                                            confidences[i])

                    cv2.putText(frame, text_box_current, (x_min, y_min - 5),
                                cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.5, colour_box_current, thickness=2)

            if writer is None:
                fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                writer = cv2.VideoWriter('result.mp4', fourcc, 25,
                                        (frame.shape[1], frame.shape[0]), True)

            writer.write(frame)

        progress_bar['value'] = 0
        video.release()
        writer.release()
```

Рисунок 3.4 - Функція виявлення та класифікації дорожніх знаків на відео

Функція `process_video` призначена для обробки відеофайлів з метою виявлення та класифікації дорожніх знаків в кожному кадрі відео. Відео завантажується через `cv2.VideoCapture(video_path)`, що дозволяє читати відеофайл кадр за кадром. Ініціалізуються змінні для записувального об'єкта (`writer`), просторових розмірів кадрів (`h`, `w`) та інші змінні для підрахунку кількості кадрів та загального часу обробки. Виконується цикл, в якому кожен кадр відео читається окремо. Якщо не вдалося отримати кадр (зображення), цикл завершується. Для кожного кадру створюється `blob` (аналогічно до `process_image`), і він передається у модель YOLO для виявлення дорожніх знаків. Аналогічно до `process_image`, перебираються всі виявлення з кожного вихідного шару YOLO, вибираючи обмежувальні рамки та рівень впевненості. Для кожного виявленого знака виконується обробка й класифікація. На кожному обробленому кадрі малюються обмежувальні рамки та додаються підписи. Кадри записуються у вихідний відеофайл.

3.2 Розробка інтерфейсу додатку

Інтерфейс додатку створено за допомогою бібліотеки `tkinter`, яка є стандартною бібліотекою для створення графічного інтерфейсу користувача (GUI) в Python[34]. Приклад коду використання бібліотеки `tkinter` зображено на рисунку 3.5.

```
# Створення основного вікна
root = tk.Tk()
root.title("Розпізнавання дорожніх знаків")
root.geometry('800x600') # Збільшення розмірів вікна
text_label = tk.Label(root, background='#C0C0C0', font=('arial', 15, 'bold'), pady=5)
text_label.pack(side=tk.TOP, fill=tk.X)
# Місце для відображення зображення
img_label = tk.Label(root, bg='grey')
img_label.pack(side=tk.TOP, fill=tk.BOTH, expand=True, pady=10)
# Панель для кнопок
buttons_frame = tk.Frame(root)
buttons_frame.pack(side=tk.TOP, pady=10) # Розташування панелі під область зображення
# Кнопка для завантаження відео
button_video = tk.Button(buttons_frame, text="Upload Video", command=upload_video, font=('arial', 15, 'bold'))
button_video.pack(side=tk.LEFT, padx=10)
# Кнопка для завантаження зображення
button_image = tk.Button(buttons_frame, text="Upload Image", command=upload_image, font=('arial', 15, 'bold'))
button_image.pack(side=tk.LEFT, padx=10)
# Кнопка для завантаження знаку
button_sign = tk.Button(buttons_frame, text="Upload sign", command=upload_sign, font=('arial', 15, 'bold'))
button_sign.pack(side=tk.LEFT, padx=10)
# Шкала прогресу обробки відео
progress_bar = ttk.Progressbar(root, orient='horizontal', mode='determinate', length=800)
progress_bar.pack(side=tk.TOP, pady=5)
# Запуск головного циклу
root.mainloop()
```

Рисунок 3.5 – Використання бібліотеки `tkinter`

На рисунку 3.6 зображено інтерфейс додатку з розпізнавання дорожніх знаків.

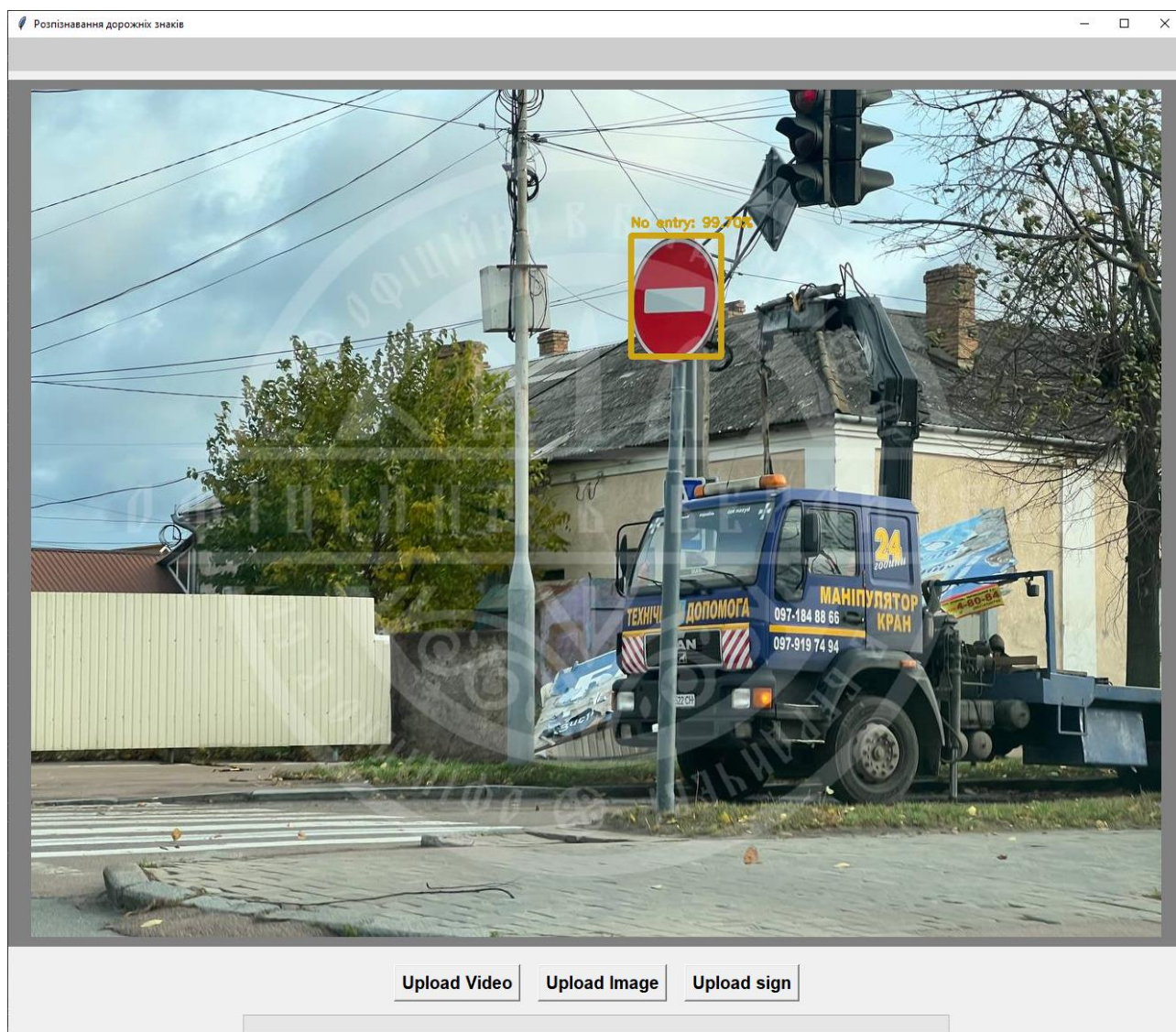


Рисунок 3.6 – Інтерфейс додатку з розпізнавання дорожніх знаків

Інтерфейс додатку має наступні елементи:

- Головне вікно програми.
- Панель зображення. Велика область частини інтерфейсу призначена для відображення зображення дорожнього знака після його завантаження та обробки.
- Кнопки управління. Під зображенням розташовано ряд кнопок: "Upload Video", "Upload Image" і "Upload sign". Ці кнопки використовуються для

завантаження відео, зображень та окремих знаків відповідно, що дозволяє користувачам взаємодіяти з програмою та завантажувати вхідні дані для обробки.

- Підписи та Анотації. Підписи на зображенні, які позначають виявлені об'єкти, зроблені зрозумілими шрифтами та мають контрастні кольори для зручності читання. Текстовий підпис поруч з виявленим знаком містить назву класу знаку та рівень впевненості у відсотках.

- Шкала прогресу обробки відео.

Інтерфейс є досить простим і зосередженим на завданні виявлення та класифікації дорожніх знаків, надаючи користувачеві необхідні інструменти для виконання завдання без зайвих елементів.

3.3 Тестовий приклад роботи додатку

Приклад роботи додатку з розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі покликаний ілюструвати, як програма використовує передові технології комп'ютерного зору та машинного навчання для ідентифікації та класифікації дорожніх знаків з фотографій та відеоматеріалів.

Програма аналізує вхідні зображення, виявляє присутні дорожні знаки, визначає їх типи та відображає результати у зрозумілому форматі. Це не лише демонструє здатність додатку розпізнавати знаки з високою точністю, але й показує, як він може бути застосований у реальному світі для підтримки водіїв, підвищення дорожньої безпеки та сприяння автоматизованому аналізу транспортних потоків.

Після запуску додатку, користувача зустрічає головне вікно програми, яке зображено на рисунку 3.7. На головному вікні додатку зображено область для відображення завантажених зображень та кнопки керування.

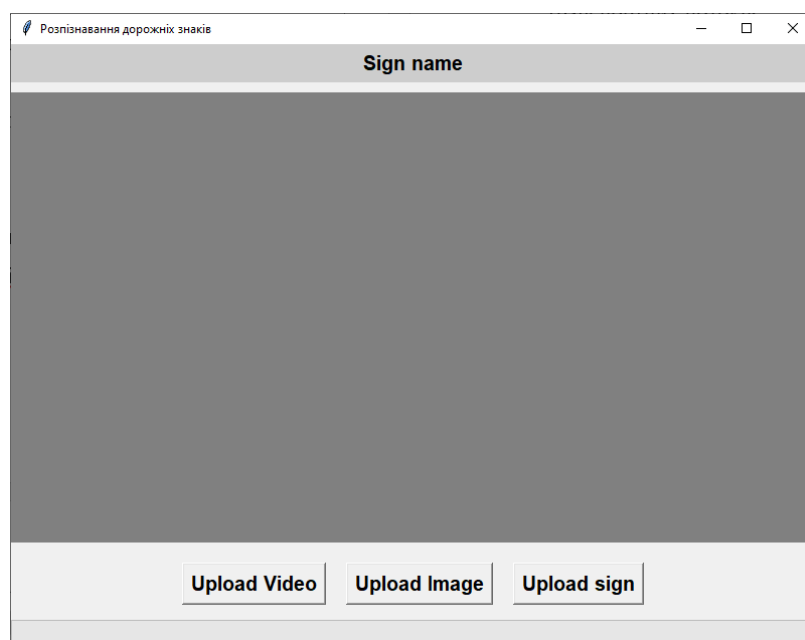


Рисунок 3.7 – Головне вікно програми

Для вибору завантажувальних даних використовується діалогове вікно Windows, яке зображено на рисунку 3.8.

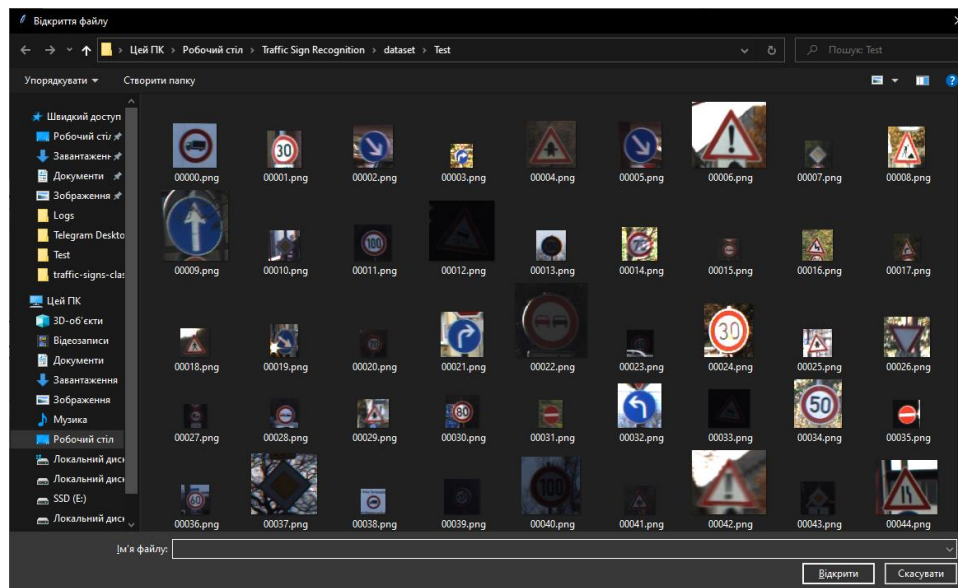


Рисунок 3.8 – Діалогове вікно завантаження файлів

Після завантаження зображення дорожнього знаку, програма проводить класифікацію зображення та підписує розпізнаний знак у верху інтерфейсу (рис. 3.9).

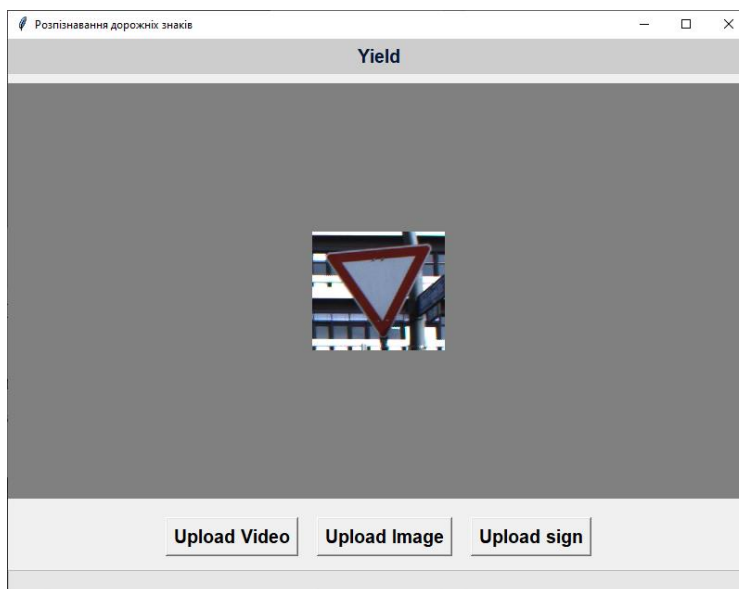


Рисунок 3.9 – Результат розпізнавання дорожнього знаку

Результат роботи додатку з ідентифікації та розпізнавання знаку зображено на рисунку 3.10.



Рисунок 3.10 – Результат ідентифікації та розпізнавання знаку

Для обробки відео, потрібно вибрати потрібний відеофайл за допомогою діалогового вікна, після чого почнеться процес обробки відео по кадрам, де на кожному кадрі буде ідентифіковано та розпізнано знак. Прогрес обробки відео

відображається за допомогою шкали прогресу у нижній частині інтерфейсу (рис. 3.11).

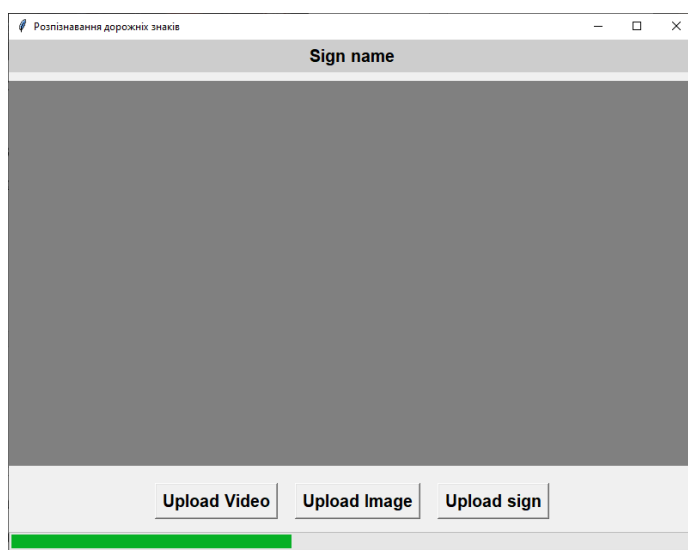


Рисунок 3.11 – Прогрес обробки відео

Після обробки відео, відкривається оброблений відеофайл на якому обводиться і підписується кожен розпізнаний знак. Приклад кадру з обробленого відеофайлу зображено на рисунку 3.12.



Рисунок 3.12 - Приклад кадру з обробленого відеофайлу

У результаті проведеного тестового прикладу роботи додатку було підтверджено його повну працездатність.

Висновки до розділу 3

Даний розділ був зосереджений на практичному застосуванні розробленої системи розпізнавання дорожніх знаків. Важливим аспектом цього розділу є детальний аналіз реалізації програмного додатку, включаючи його інтерфейс, функціонал та способи взаємодії з користувачем.

Основним досягненням розділу є демонстрація повної працездатності додатку, що підтверджується через ряд тестових прикладів. Розглядається процес ідентифікації та класифікації дорожніх знаків з використанням згорткової нейронної мережі, демонструючи ефективність застосування глибокого навчання в задачах комп'ютерного зору.

Загалом було продемонстровано успішне втілення теоретичних знань та досліджень у практичну розробку програмного додатку.

ВИСНОВОК

У кваліфікаційній роботі було успішно реалізовано актуальне завдання розробки програмного додатку для розпізнавання дорожніх знаків за допомогою згорткової нейронної мережі.

Був проведений аналіз існуючих методів розпізнавання зображень за допомогою нейронних мереж, розглянуто переваги і недоліки кожного з них. Це дало змогу впевнитися у тому, що використання згорткових нейронних мереж дозволить створити ефективну модель, яка здатна ідентифікувати та класифікувати дорожні знаки з високою точністю.

Для тренування моделі було обрано набір даних Traffic Signs Preprocessed, що містить колекцію попередньо оброблених зображень із 43 видів дорожніх знаків. Після детального огляду набору, з нього були вилучені потрібні для навчання дані.

Було розроблено та оптимізовано архітектуру згорткової нейронної мережі, яка продемонструвала вражаючу здатність до розпізнавання різноманітних дорожніх знаків. Загалом було натреновано 9 моделей з різними фільтрами, отримано корисні метрики, що демонструють ефективність розроблених моделей. Результати експериментів підтвердили, що модель ефективно виділяє важливі риси зображень знаків та автоматично класифікує їх за стандартними категоріями.

Отримані результати свідчать про високий потенціал згорткових нейронних мереж у задачі розпізнавання дорожніх знаків. Розроблена модель має практичну цінність для застосування в системах допомоги водіям та автономних автомобілях, що сприяє підвищенню безпеки на дорогах.

З огляду на швидкий розвиток галузі глибокого навчання, є очевидною потреба у подальшому вдосконаленні архітектур моделей, методів аугментації даних та вивченні можливостей використання мереж з більшою кількістю шарів для досягнення ще вищої точності та надійності моделі в реальних умовах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Муковоз В.С., Вакалюк Т.А. Використання нейронних мереж для обробки зображень та відео. Тези доповідей XVI Всеукраїнської науково-практичної WEB конференції аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі» 21-23 березня 2023 р., с. 146-147.
2. Бішоп К. Характеристики Розпізнавання образів і машинне навчання / Крістофер Бішоп., 2020. – 480 с. – (Діалектика).
3. Чару Аггарвал. Нейронні мережі і глибоке навчання. Навчальний курс. / Чару Аггарвал., 2020. – 752 с. – (Діалектика).
4. Yamashita R. Convolutional neural networks: an overview and application in radiology / R. Yamashita, M. Nishio, R. Kinh. – 2018.
5. Indolia S. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach / S. Indolia, P. Asopa. – 2018. – С. 10.
6. Frąckiewicz M. Роль трансферного навчання в мовних моделях [Електронний ресурс] / Marcin Frąckiewicz. – 2023. – Режим доступу до ресурсу: <http://surl.li/kenhx>.
7. Ranasinghe K. Intriguing Properties of Vision Transformers / K. Ranasinghe, M. Hayat, M. Yang. – 2021. – С. 13.
8. Kulkarni P. Transformer Models for Language and Vision: Theory and Applications / Prajit Kulkarni., 2023. – 104 с.
9. Хайкін С. Нейронні мережі. / Саймон Хайкін.. – 1104 с. – (Діалектика).
10. Pascanu R. On the difficulty of training recurrent neural networks / R. Pascanu, T. Mikolov, Y. Bengio. – 2013. – С. 9.
11. Salem F. Recurrent Neural Networks: From Simple to Gated Architectures / Fathi Salem., 2022. – 141 с. – (Springer).
12. Graves A. Multi-dimensional Recurrent Neural Networks / A. Graves, S. Fernández, J. Schmidhuber. – 2007.
13. Правила дорожнього руху [Електронний ресурс] – Режим доступу до ресурсу: <https://autoshkola-navihator.com.ua/pravyla-dorozhnoho-rukhu-ukrayina>.

14. Opel Eye [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: <https://preview.thenewsmarket.com/Previews/NCAP/DocumentAssets/188042.pdf>.
15. The driver assistance systems in the EQC point the way to autonomous driving. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.la.mercedes-benz.com/en/passengercars/mercedes-benz-cars/models/eqc/safety/assistance-systems.module.html>.
16. Road sign information [Електронний ресурс] – Режим доступу до ресурсу: <https://www.bosch-mobility.com/en/solutions/assistance-systems/road-sign-information/>.
17. Traffic Sign Recognition [Електронний ресурс] – Режим доступу до ресурсу: <https://www.volkswagen.co.uk/en/technology/driver-assist/sign-assist.html>.
18. Маттес Е. Python Crash Course / Ерік Маттес., 2023. – 552 с. – (No Starch Press).
19. Yarlagadda R. Python Engineering Automation to Advance Artificial Intelligence and Machine Learning Systems / Ravi Yarlagadda. – 2021. – С. 11.
20. Hu Q. DeepGraph: A PyCharm Tool for Visualizing and Understanding Deep Learning Models / Q. Hu, L. Ma. – 2018.
21. PyCharm як найкраща IDE для розробки ПЗ на Python [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/pycharm-tse/>.
22. German Traffic Sign Recognition Benchmark [Електронний ресурс] – Режим доступу до ресурсу: <https://benchmark.ini.rub.de/>.
23. Sewak M. Practical Convolutional Neural Networks: Implement advanced deep learning models using Python / M. Sewak, P. Pujari., 2018. – 218 с. – (Packt Publishing).
24. Venkatesan R. Convolutional Neural Networks in Visual Computing: A Concise Guide (Data-Enabled Engineering) / Ragav Venkatesan., 2017. – 168 с. – (CRC Press)
25. Sichkar V. Traffic Signs Preprocessed [Електронний ресурс] / Valentyn Sichkar. – 2019. – Режим доступу до ресурсу: <https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-preprocessed>.

26. Jun-Mo J. Effectiveness of Normalization Pre-Processing of Big Data to the Machine Learning Performance / J. Jun-Mo. – 2019.
27. Keras: Deep Learning for humans [Электронный ресурс] – Режим доступа до ресурсу: <https://keras.io/>.
28. Chollet F. Deep Learning with Python, Second Edition / Francois Chollet., 2021. – 504 с. – (Manning).
29. Training & evaluation with the built-in methods [Электронный ресурс] – Режим доступа до ресурсу: https://www.tensorflow.org/guide/keras/training_with_built_in_methods.
30. Pullum L. Guidance for the Verification and Validation of Neural Networks (Emerging Technologies) / L. Pullum, B. Taylor., 2007. – 133 с. – (Wiley-IEEE Computer Society Pr).
31. Qin Z. How convolutional neural network see the world - A survey of convolutional neural network visualization methods / Z. Qin, F. Yu, X. Chen. – 2018. – С. 32.
32. OpenCV - Open Computer Vision Library [Электронный ресурс] – Режим доступа до ресурсу: <https://opencv.org/>.
33. YOLO: Real-Time Object Detection [Электронный ресурс] – Режим доступа до ресурсу: <https://pjreddie.com/darknet/yolo/>.
34. tkinter — Интерфейс Python до Tcl/Tk [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.python.org/uk/3/library/tkinter.html>.

ДОДАТКИ

Посилання на набір даних дорожніх знаків «Traffic Signs Preprocessed»:

<https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-preprocessed>

Посилання на набір даних дорожніх знаків у YOLO форматі:

<https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format>

Посилання на треновані ваги для ідентифікації дорожніх знаків на основі YOLOv3:

<https://www.kaggle.com/datasets/kishlaykk/trainedtrafficsignsdetectorbasedonyolov3/data>

Лістинг коду файлу modelTraining.py

```

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import pickle
import matplotlib.pyplot as plt
from timeit import default_timer as timer

from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, AvgPool2D,
BatchNormalization, Reshape
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
all files under the input directory

import os
for dirname, _, filenames in os.walk('input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

print(os.listdir('input'))

# Відкриття файлу для читання
with open('input/traffic-signs-preprocessed/data2.pickle', 'rb') as f:
    data = pickle.load(f, encoding='latin1') # dictionary type

# Підготовка y_train та y_validation для використання в Keras
data['y_train'] = to_categorical(data['y_train'], num_classes=43)
data['y_validation'] = to_categorical(data['y_validation'], num_classes=43)

# Створення каналів у кінці
data['x_train'] = data['x_train'].transpose(0, 2, 3, 1)
data['x_validation'] = data['x_validation'].transpose(0, 2, 3, 1)
data['x_test'] = data['x_test'].transpose(0, 2, 3, 1)

# Відображення завантажених даних з файлу
for i, j in data.items():
    if i == 'labels':
        print(i + ':', len(j))
    else:
        print(i + ':', j.shape)

# Підготовка функції для побудови набору прикладів
# На вхід буде подано 4D тензор і перетворено його на сітку
# Значення будуть масштабуватися до діапазону [0, 255]
def convert_to_grid(x_input):
    N, H, W, C = x_input.shape
    grid_size = int(np.ceil(np.sqrt(N)))
    grid_height = H * grid_size + 1 * (grid_size - 1)
    grid_width = W * grid_size + 1 * (grid_size - 1)
    grid = np.zeros((grid_height, grid_width, C)) + 255
    next_idx = 0
    y0, y1 = 0, H
    for y in range(grid_size):
        x0, x1 = 0, W
        for x in range(grid_size):
            if next_idx < N:

```

```

        img = x_input[next_idx]
        low, high = np.min(img), np.max(img)
        grid[y0:y1, x0:x1] = 255.0 * (img - low) / (high - low)
        next_idx += 1
        x0 += W + 1
        x1 += W + 1
        y0 += H + 1
        y1 += H + 1

    return grid

# Візуалізація прикладів навчальних даних
examples = data['x_train'][:81, :, :, :]
print(examples.shape)  # (81, 32, 32, 3)

# Побудова прикладів
fig = plt.figure()
grid = convert_to_grid(examples)
plt.imshow(grid.astype('uint8'), cmap='gray')
plt.axis('off')
plt.gcf().set_size_inches(15, 15)
plt.title('Some examples of training data', fontsize=18)

plt.show()

fig.savefig('training_examples.png')
plt.close()

model = Sequential()
model.add(Conv2D(32, kernel_size=3, padding='same', activation='relu',
input_shape=(32, 32, 3)))
model.add(MaxPool2D(pool_size=2))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dense(43, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** (x + epochs))
epochs = 15

h = model.fit(data['x_train'][:10], data['y_train'][:10],
            batch_size=5, epochs = epochs,
            validation_data = (data['x_validation'], data['y_validation']),
            callbacks=[annealer], verbose=1)

print('Epochs={0:d}, training accuracy={1:.5f}, validation accuracy={2:.5f}'.\
      format(epochs, max(h.history['accuracy']),
max(h.history['val_accuracy'])))

plt.rcParams['figure.figsize'] = (15.0, 5.0) # Встановлення розміру за замовчу-
ванням
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['font.family'] = 'Times New Roman'

fig = plt.figure()
plt.plot(h.history['accuracy'], '-o', linewidth=3.0)
plt.plot(h.history['val_accuracy'], '-o', linewidth=3.0)
plt.title('Overfitting small data', fontsize=22)
plt.legend(['train', 'validation'], loc='upper left', fontsize='xx-large')
plt.xlabel('Epoch', fontsize=20)
plt.ylabel('Accuracy', fontsize=20)

```

```

plt.tick_params(labelsize=18)

plt.show()

fig.savefig('overfitting_small_data.png')
plt.close()

filters = [3, 5, 9, 13, 15, 19, 23, 25, 31]
model = [0] * len(filters)

for i in range(len(model)):
    model[i] = Sequential()
    model[i].add(Conv2D(32, kernel_size=filters[i], padding='same',
activation='relu', input_shape=(32, 32, 3)))
    model[i].add(MaxPool2D(pool_size=2))
    model[i].add(Flatten())
    model[i].add(Dense(500, activation='relu'))
    model[i].add(Dense(43, activation='softmax'))
    model[i].compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** (x + epochs))
epochs = 5

h = [0] * len(model)

for i in range(len(h)):
    h[i] = model[i].fit(data['x_train'], data['y_train'],
                        batch_size=5, epochs=epochs,
                        validation_data=(data['x_validation'],
data['y_validation']),
                        callbacks=[annealer], verbose=0)

    print('Model with filters {0:d}x{0:d}, epochs={1:d}, training
accuracy={2:.5f}, validation accuracy={3:.5f}'. \
          format(filters[i], epochs, max(h[i].history['accuracy']),
max(h[i].history['val_accuracy'])))

plt.rcParams['figure.figsize'] = (15.0, 15.0) # Встановлення розміру за замовчу-
ванням
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['font.family'] = 'Times New Roman'

# Побудова історії точності тренувань
fig = plt.figure()
plt.subplot(2, 1, 1)
plt.plot(h[8].history['accuracy'], '-o', linewidth=3.0)
plt.plot(h[7].history['accuracy'], '-s', linewidth=3.0)
plt.plot(h[6].history['accuracy'], '-D', linewidth=3.0)
plt.plot(h[5].history['accuracy'], '-D', linewidth=3.0)
plt.plot(h[4].history['accuracy'], '-o', linewidth=3.0)
plt.plot(h[3].history['accuracy'], '-o', linewidth=3.0)
plt.plot(h[2].history['accuracy'], '-o', linewidth=3.0)
plt.plot(h[1].history['accuracy'], '-o', linewidth=3.0)
plt.plot(h[0].history['accuracy'], '-o', linewidth=3.0)
plt.legend(['filter 31', 'filter 25', 'filter 23', 'filter 19', 'filter 15',
'filter 13', 'filter 9', 'filter 5', 'filter 3'], loc='lower right',
fontsize='xx-large', borderpad=2)
plt.xlabel('Epoch', fontsize=20, fontname='Times New Roman')
plt.ylabel('Training Accuracy', fontsize=20, fontname='Times New Roman')
plt.yscale('linear')
plt.ylim(0.85, 1.0)
plt.xlim(0.5, 5.3)
plt.title('Accuracy for different sizes of filters', fontsize=22)

```

```

plt.tick_params(labelsize=18)

plt.subplot(2, 1, 2)
plt.plot(h[8].history['val_accuracy'], '-o', linewidth=3.0)
plt.plot(h[7].history['val_accuracy'], '-s', linewidth=3.0)
plt.plot(h[6].history['val_accuracy'], '-D', linewidth=3.0)
plt.plot(h[5].history['val_accuracy'], '-D', linewidth=3.0)
plt.plot(h[4].history['val_accuracy'], '-o', linewidth=3.0)
plt.plot(h[3].history['val_accuracy'], '-o', linewidth=3.0)
plt.plot(h[2].history['val_accuracy'], '-o', linewidth=3.0)
plt.plot(h[1].history['val_accuracy'], '-o', linewidth=3.0)
plt.plot(h[0].history['val_accuracy'], '-o', linewidth=3.0)
plt.legend(['filter 31', 'filter 25', 'filter 23', 'filter 19', 'filter 15',
'filter 13', 'filter 9', 'filter 5', 'filter 3'], loc='lower right',
fontsize='xx-large', borderpad=2)
plt.xlabel('Epoch', fontsize=20, fontname='Times New Roman')
plt.ylabel('Validation Accuracy', fontsize=20, fontname='Times New Roman')
plt.yscale('linear')
plt.ylim(0.75, 0.9)
plt.xlim(0.5, 5.3)
plt.tick_params(labelsize=18)

plt.show()

fig.savefig('models_accuracy.png')
plt.close()

# Відображення значень точності для різних фільтрів
for i in range(len(h)):
    print('data2 filter {0:d} training accuracy = {1:.5f}'.\
          format(filters[i], np.max(h[i].history['accuracy'])))

print()

for i in range(len(h)):
    print('data2 filter {0:d} validation accuracy = {1:.5f}'.\
          format(filters[i], np.max(h[i].history['val_accuracy'])))

for i in range(len(model)):
    temp = model[i].predict(data['x_test'])
    temp = np.argmax(temp, axis=1)

    # Порівнюємо передбачений клас з правильним для всіх вхідних зображень
    # обчислюємо середнє значення серед усіх значень наступного numpy масиву
    # Виразом 'testing_accuracy == data['y_test']' створюємо numpy масив зі значеннями True та False
    # Функція np.mean поверне середнє значення елементів масиву
    # За замовчуванням середнє значення береться по згладженому масиву
    temp = np.mean(temp == data['y_test'])

    print('data2 filter {0:d} testing accuracy = {1:.5f}'.format(filters[i],
temp))

# Отримання оцінок з прямого проходження одного вхідного зображення
# Оцінки даються для кожного зображення з 43 числами передбачень для кожного класу
# Одночасне вимірювання часу виконання

for i in range(len(model)):
    start = timer()
    temp = model[i].predict(data['x_test'][:1, :, :, :])
    end = timer()

```

```

    print('data2 filter {0:d} classification time = {1:.5f}'.format(filters[i],
end - start))

for i in range(len(model)):
    w = model[i].get_weights()
    print(w[0].shape)

    # Візуалізація фільтрів
    temp = w[0].transpose(3, 0, 1, 2)
    print(temp.shape)  # (81, 32, 32, 3)

    fig = plt.figure()
    grid = convert_to_grid(temp)
    plt.imshow(grid.astype('uint8'), cmap='gray')
    plt.axis('off')
    plt.gcf().set_size_inches(10, 10)
    name = 'Trained filters ' + str(filters[i]) + 'x' + str(filters[i])
    plt.title(name, fontsize=18)

    plt.show()

    name = 'filters-' + str(filters[i]) + 'x' + str(filters[i]) + '.png'
    fig.savefig(name)
    plt.close()

# Підготовка зображення для прогнозування з тестового набору даних
x_input = data['x_test'][100:101]
print(x_input.shape)
y_input = data['y_test'][100:101]
print(y_input)

plt.rcParams['figure.figsize'] = (2.5, 2.5)
plt.imshow(x_input[0, :, :, :])
plt.axis('off')

plt.show()

# Отримання оцінок з прямого проходження вхідного зображення
scores = model[0].predict(x_input)
print(scores[0].shape)  # (43,)

# Оцінки виводяться для зображення, яке містить 43 числа передбачень для кожного
класу
# Отримання лише одного класу з максимальним значенням
prediction = np.argmax(scores)
print('ClassId:', prediction)

# Визначення функції для отримання текстів для кожного класу - міток
def label_text(file):
    # Визначення функції для отримання текстів для кожного класу - міток
    label_list = []

    # Читання файлу 'csv' та отримання міток зображень
    r = pd.read_csv(file)
    # Перебираємо всі імена
    for name in r['SignName']:
        # Додавання до кожного рядка другого стовпчика з назвою мітки
        label_list.append(name)

    # Повернення результуючого списку з мітками
    return label_list

```

```
labels = label_text('input/traffic-signs-preprocessed/label_names.csv')  
  
print('Label:', labels[prediction])  
  
for i in range(len(model)):  
    name = 'model-' + str(filters[i]) + 'x' + str(filters[i]) + '.h5'  
    model[i].save(name)
```

Лістинг коду файлу app.py

```

import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from PIL import Image, ImageTk
import cv2
import numpy as np
from keras.models import load_model
import pickle
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import time
import matplotlib.pyplot as plt
import threading
from os import startfile

# Ініціалізація моделі та інших ресурсів
model = load_model('input/traffic-signs-classification-with-cnn/model-5x5.h5')
with open('input/traffic-signs-preprocessed/mean_image_rgb.pickle', 'rb') as f:
    mean = pickle.load(f, encoding='latin1') # dictionary type

# Читання csv-файлу з назвами міток
# Завантаження двох стовпців [0, 1] у Pandas dataframe
labels = pd.read_csv('input/traffic-signs-preprocessed/label_names.csv')

# Ініціалізація необхідних змінних
path_to_cfg = 'input/traffic-signs-dataset-in-yolo-format/yolov3_ts_test.cfg'
path_to_weights = 'input/trained-traffic-signs-detector-based-on-yolo-
v3/yolov3_ts_train_5000.weights'

# Ініціалізація моделі YOLO
network = cv2.dnn.readNetFromDarknet(path_to_cfg, path_to_weights)
network.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
network.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU) # Використання CPU

# Параметри для YOLO
probability_minimum = 0.2
threshold = 0.2

# Генерування кольорів для обмежувальних рамок
colours = np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')
# Отримання назв усіх шарів від мережі
layers_names_all = network.getLayerNames()

# Отримання лише вихідних шарів
layers_names_output = [layers_names_all[i - 1] for i in
network.getUnconnectedOutLayers().flatten()]

# Функція для обробки зображення
def process_image(image_path):
    # Завантаження зображення
    image_BGR = cv2.imread(image_path)
    h, w = image_BGR.shape[:2] # Отримання розмірів зображення
    # Списки для зберігання даних про виявлені об'єкти
    bounding_boxes = []
    confidences = []
    class_numbers = []
    # Створення blob із зображення
    blob = cv2.dnn.blobFromImage(image_BGR, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
    network.setInput(blob) # Встановлення blob як вхідних даних для мережі

```

```

output_from_network = network.forward(layers_names_output)

# Перебір кожного виявлення
for result in output_from_network:
    for detected_objects in result:
        scores = detected_objects[5:]
        class_current = np.argmax(scores)
        confidence_current = scores[class_current]
        if confidence_current > probability_minimum:
            box_current = detected_objects[0:4] * np.array([w, h, w, h])
            x_center, y_center, box_width, box_height = box_current
            x_min = int(x_center - (box_width / 2))
            y_min = int(y_center - (box_height / 2))
            bounding_boxes.append([x_min, y_min, int(box_width),
int(box_height)])
            confidences.append(float(confidence_current))
            class_numbers.append(class_current)

    results = cv2.dnn.NMSBoxes(bounding_boxes, confidences, probability_minimum,
threshold)
    # Перевірка наявності виявлених об'єктів
    if len(results) > 0:
        for i in results.flatten():
            # Координати прямокутника, його ширина та висота
            x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1]
            box_width, box_height = bounding_boxes[i][2], bounding_boxes[i][3]
            # Вирізання фрагменту з дорожнім знаком
            c_ts = image_BGR[y_min:y_min + int(box_height), x_min:x_min +
int(box_width), :]
            if c_ts.shape[:1] == (0,) or c_ts.shape[1:2] == (0,):
                continue
            else:
                # Обробка фрагменту знака
                blob_ts = cv2.dnn.blobFromImage(c_ts, 1 / 255.0, (32, 32),
swapRB=True, crop=False)
                blob_ts[0] = blob_ts[0, :, :, :] - mean['mean_image_rgb']
                blob_ts = blob_ts.transpose(0, 2, 3, 1)
                # Класифікація фрагменту
                scores = model.predict(blob_ts)
                prediction = np.argmax(scores)
                # Текст для підпису
                label = labels['SignName'][prediction]
                confidence = confidences[i]
                text_box_current = '{}: {:.2f}%'.format(label, confidence * 100)
                # Колір для прямокутника
                colour_box_current = colours[class_numbers[i]].tolist()
                # Малювання прямокутника на оригінальному зображенні
                cv2.rectangle(image_BGR, (x_min, y_min), (x_min + box_width,
y_min + box_height), colour_box_current, 6)
                # Додавання тексту до зображення
                cv2.putText(image_BGR, text_box_current, (x_min, y_min - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                                colour_box_current, 2)

    return image_BGR

def process_sign(image_path):
    image = Image.open(image_path)
    image = image.resize((32, 32))
    image = np.expand_dims(image, axis=0)
    image = np.array(image)
    pred = model.predict(image)
    pred_class = np.argmax(pred, axis=-1)[0]
    sign = labels.loc[pred_class, 'SignName']
    return sign

```

```

def process_video(video_path):
    video = cv2.VideoCapture(video_path)
    writer = None
    h, w = None, None
    plt.rcParams['figure.figsize'] = (3, 3)
    f = 0
    t = 0

    while True:
        ret, frame = video.read()
        if not ret:
            break
        if w is None or h is None:
            h, w = frame.shape[:2]

        blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
        network.setInput(blob)
        start = time.time()
        output_from_network = network.forward(layers_names_output)
        end = time.time()
        f += 1
        t += end - start
        total_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
        progress_bar['maximum'] = total_frames

        progress_bar['value'] += 1
        root.update()
        print('Frame number {0} took {1:.5f} seconds'.format(f, end - start))

        bounding_boxes = []
        confidences = []
        class_numbers = []

        for result in output_from_network:
            for detected_objects in result:
                scores = detected_objects[5:]
                class_current = np.argmax(scores)
                confidence_current = scores[class_current]

                if confidence_current > probability_minimum:
                    box_current = detected_objects[0:4] * np.array([w, h, w, h])
                    x_center, y_center, box_width, box_height = box_current
                    x_min = int(x_center - (box_width / 2))
                    y_min = int(y_center - (box_height / 2))
                    bounding_boxes.append([x_min, y_min, int(box_width),
int(box_height)])

                    confidences.append(float(confidence_current))
                    class_numbers.append(class_current)

                results = cv2.dnn.NMSBoxes(bounding_boxes, confidences,
probability_minimum, threshold)

                if len(results) > 0:
                    for i in results.flatten():
                        x_min, y_min = bounding_boxes[i][0], bounding_boxes[i][1]
                        box_width, box_height = bounding_boxes[i][2],
bounding_boxes[i][3]

                        c_ts = frame[y_min:y_min + int(box_height), x_min:x_min +
int(box_width), :]

                        if c_ts.shape[:1] == (0,) or c_ts.shape[1:2] == (0,):

```

```

        pass
    else:
        blob_ts = cv2.dnn.blobFromImage(c_ts, 1 / 255.0, size=(32,
32), swapRB=True, crop=False)
        blob_ts[0] = blob_ts[0, :, :, :] - mean['mean_image_rgb']
        blob_ts = blob_ts.transpose(0, 2, 3, 1)
        scores = model.predict(blob_ts)
        prediction = np.argmax(scores)
        colour_box_current = colours[class_numbers[i]].tolist()

        cv2.rectangle(frame, (x_min, y_min),
                        (x_min + box_width, y_min + box_height),
                        colour_box_current, 2)

        text_box_current = '{}:
{: .4f}'.format(labels['SignName'][prediction],
                                                           confidences[i])

        cv2.putText(frame, text_box_current, (x_min, y_min - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    colour_box_current, 2)

    if writer is None:
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')

        writer = cv2.VideoWriter('result.mp4', fourcc, 25,
                                (frame.shape[1], frame.shape[0]), True)

    writer.write(frame)

    progress_bar['value'] = 0
    video.release()
    writer.release()

# Функція для відображення зображення у GUI
def show_image(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    im = Image.fromarray(image)
    img = ImageTk.PhotoImage(image=im)
    img_label.config(image=img)
    img_label.image = img

# Функція для завантаження та обробки зображення
def upload_image():
    text_label.configure(text='Sign name')
    file_path = filedialog.askopenfilename()
    if file_path:
        processed_image = process_image(file_path)
        show_image(processed_image)

# Функція для завантаження відео
def upload_video():
    file_path = filedialog.askopenfilename()
    if file_path:
        process_video(file_path)
        startfile('result.mp4')
    pass

# Функція для завантаження знаку
def upload_sign():
    file_path = filedialog.askopenfilename()
    if file_path:
        sign_name = process_sign(file_path)

```

```

image = cv2.imread(file_path)
show_image(image)
text_label.configure(foreground='#011638', text=sign_name)

# Створення основного вікна
root = tk.Tk()
root.title("Розпізнавання дорожніх знаків")
root.geometry('800x600') # Збільшення розмірів вікна
text_label = tk.Label(root, background='#CDCDCD', font=('arial',15,'bold'),
pady=5)
text_label.pack(side=tk.TOP, fill=tk.X)
text_label.configure(text='Sign name')
# Місце для відображення зображення
img_label = tk.Label(root, bg='grey')
img_label.pack(side=tk.TOP, fill=tk.BOTH, expand=True, pady=10)
# Панель для кнопок
buttons_frame = tk.Frame(root)
buttons_frame.pack(side=tk.TOP, pady=10) # Розташування панелі під областю зо-
браження
# Кнопка для завантаження відео
button_video = tk.Button(buttons_frame, text="Upload Video",
command=upload_video, font=('arial',15,'bold'))
button_video.pack(side=tk.LEFT, padx=10)
# Кнопка для завантаження зображення
button_image = tk.Button(buttons_frame, text="Upload Image",
command=upload_image, font=('arial',15,'bold'))
button_image.pack(side=tk.LEFT, padx=10)
# Кнопка для завантаження знаку
button_sign = tk.Button(buttons_frame, text="Upload sign", command=upload_sign,
font=('arial',15,'bold'))
button_sign.pack(side=tk.LEFT, padx=10)
# Шкала прогресу обробки відео
progress_bar = ttk.Progressbar(root, orient='horizontal', mode='determinate',
length=800)
progress_bar.pack(side=tk.TOP, pady=5)
# Запуск головного циклу
root.mainloop()

```