Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

СТВОРЕННЯ ТА ОПТИМІЗАЦІЯ МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ НА ЗОБРАЖЕННЯХ

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп'ютерні науки»
Назарчук Богдан Григорович
Науковий керівник:
к. ф.-м. н., доц. Мороз І. П.

Рівне – 2024

Зміст

ВСТУП

Проблема автоматичного розпізнавання дорожніх знаків на зображеннях є досить перспективною і актуальною. Важливість проблеми підсилюється швидким розвитком технологій у сфері машинного навчання та комп'ютерного зору на основі використання нейронних мереж.

Більшість досліджень у цій галузі спрямовані на вдосконалення алгоритмів розпізнавання, покращення якості даних для навчання моделей, на оптимізацію архітектури нейронних мереж для отримання кращих результатів у реальному часі тощо. Впровадження та дослідження таких концепцій, як механізми аугментації даних (техніка штучного збільшення навчального набору шляхом створення модифікованих копій набору даних з використанням існуючих даних) для розширення навчальної вибірки, оптимізація глибоких архітектур для підвищення ефективності (продуктивності, точності) відповідних програмних систем, а також пошук оптимальних рішень для роботи в реальному часі є ключовими задачами в даній області.

З врахуванням доступності великої кількості відкритих даних з дорожніми знаками, розвиток моделей для їх розпізнавання та класифікації продовжується, надаючи перспективи для подальших інновацій та вдосконалень у цій сфері.

Актуальність роботи. Створення та оптимізація моделей нейронних мереж для автоматичного розпізнавання дорожніх знаків на зображеннях відіграє велику роль у покращенні систем безпеки на дорогах та автоматизації транспортних систем. Актуальність цієї роботи базується на кількох ключових аспектах.

По-перше, безпека дорожнього руху та запобігання аваріям є однією з найважливіших проблем у сучасному світі. Розпізнавання дорожніх знаків за допомогою нейронних мереж може використовуватися для попередження водіїв про обмеження швидкості, небезпечні дорожні умови, а також інші важливі аспекти безпеки на дорогах. Це допомагає знизити ризик дорожніх аварій та підвищує загальний рівень безпеки для усіх учасників руху.

По-друге, розвиток технологій у сфері машинного навчання, зокрема глибоке навчання, дозволяє створювати більш точні та швидкі моделі для розпізнавання знаків на зображеннях. Це відкриває можливості для реалізації систем штучного інтелекту (ШІ), які здатні розпізнавати різноманітні типи дорожніх знаків з високою точністю.

По-третє, автоматичне розпізнавання дорожніх знаків може полегшити життя водіям, забезпечуючи їх інформацією щодо поточних обмежень, дозволяючи уникнути штрафів та забезпечуючи зручність та комфорт при подорожах.

По-четверте, впровадження таких систем в автомобільну промисловість та інфраструктуру дорожнього руху може мати значний вплив на ефективність транспортної системи та міського планування.

Згідно з цими аспектами, важливо розвивати та вдосконалювати моделі нейронних мереж для розпізнавання дорожніх знаків на зображеннях, оскільки це відкриває нові можливості для забезпечення безпеки на дорогах, покращення транспортної системи та створення більш комфортних умов для учасників дорожнього руху.

Завдяки використанню глибокого навчання, моделі можуть навчатися розпізнавати та класифікувати різні типи дорожніх знаків з високою точністю та швидкістю.

Метою роботи є вивчення технологій розробки штучних нейронних мереж для вирішення практичних задач розпізнавання та ідентифікації.

Об’єкт дослідження – процеси розпізнавання та ідентифікації під час дорожнього руху в реальному часі.

Предметом дослідження є технології проектування, розробки, навчання, оптимізації та використання штучних нейронних мереж для розпізнавання дорожніх знаків.

Методи дослідження. Застосовано загальнонаукові підходи до пошуку і аналізу інформації за тематикою дослідження.

Для досягнення мети були поставлені наступні **завдання дослідження:**

- використання глибокого навчання;

- аугментація даних;
- оптимізація моделі;
- валідація та оцінка результатів;
- технології передавання навчання (Transfer Learning);
- експерименти з архітектурою моделі.
- реалізація додатку для розпізнавання дорожніх знаків на фото чи відео з використанням розробленого алгоритму.
- тестування та оцінка ефективності розробленого додатку на справжніх дорожніх ситуаціях.

У процесі виконання дослідження зібрано та аналізовано великий обсяг даних, які використовувалися для створення анотованого датасету. На основі цього датасету була розроблена та навчена нейронна мережа для розпізнавання дорожніх знаків на зображеннях. Розроблена модель була налаштована для взаємодії з камерою для практичного використання.

Викладені в дипломній роботі матеріали є основою для розробки та проектування відповідних нейронних мереж. Результати роботи можуть використовуватись у навчальному процесі.

Результати досліджень доповідались на звітній викладацько-студентській конференції РДГУ Тези опубліковані у збірнику матеріалів XXIV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів м. Одеса. Опубліковані 1 тези доповіді.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Базові поняття

Метод пошуку об'єктів на зображеннях, базується на використанні нейронних мереж. Ці мережі відтворюють біологічні нейронні мережі людського мозку для обробки інформації. Нейрони людського мозку взаємодіють, використовуючи складну мережу нервових клітин, що з'єднані через синапси. Кожен нейрон отримує інформацію через дендрити й передає її аксоном, який розгалужується на тисячі синапсів. Взаємодія між нейронами відбувається за допомогою серії імпульсів, які тривають лише декілька мілісекунд і передаються з різною частотою, від декількох до сотень герців. Хоча цей процес повільніший, ніж у сучасних комп'ютерах, людський мозок здатний швидше обробляти інформацію, таку як розпізнавання звуків. Вивчення механізмів функціонування окремих нейронів та їх взаємодії має велике значення для розуміння процесів у нервовій системі, включаючи пошук, передачу та обробку інформації.

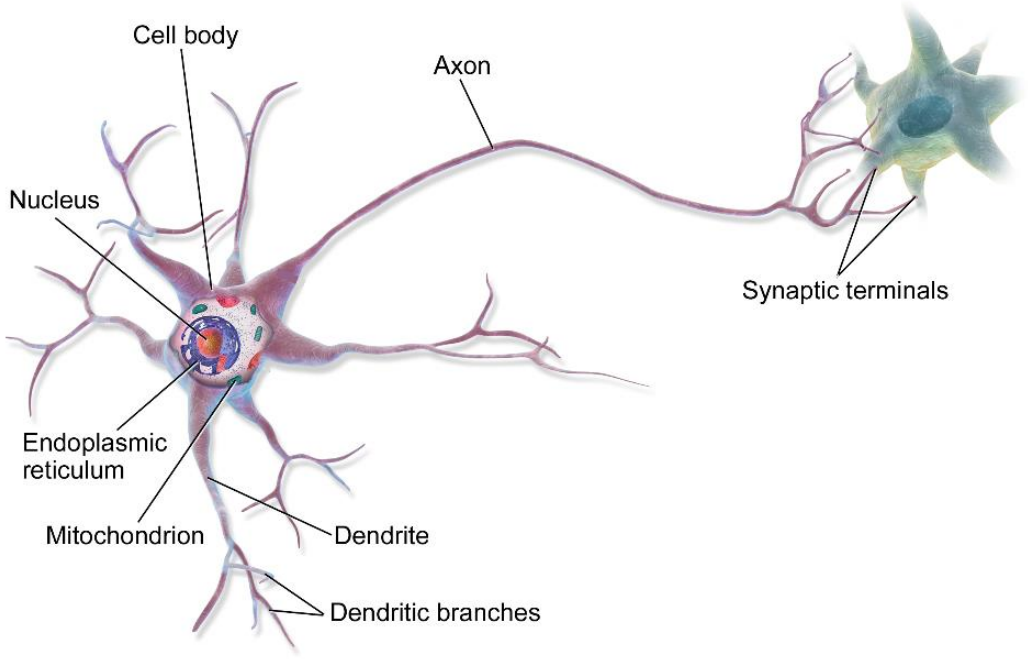

Рис. 1.1. Будова нейрона

Кожен нейрон має тіло, відоме як сома, з ядром і типовим набором органел (рис.1.1. [1]). Від нейрона виходить багато відростків, які грають ключову роль у взаємодії з іншими нервовими клітинами. Існують два основних типи відростків: тонкі, гіллясті дендрити і товстіший аксон, кінець якого розгалужується як гілля. Вхідні сигнали до клітини надходять через синапси, а вихідний сигнал передається аксоном через його численні нервові закінчення, що контактують з іншими нейронами, утворюючи нові синапси. Ці синапси з'єднуються з клітиною виходами інших нейронів і можуть бути розташовані як на дендритах, так і на тілі клітини.

Узагальнюючи, складові біологічного нейрона включають[2]:

Дендрити (Dendrite) - вхідні волокна, що постачають інформацію нейрону.

Аксон (Axon) - довге вихідне волокно, яке передає інформацію до м'язових волокон та інших нейронів (нейрон може мати лише один аксон, іноді він відсутній зовсім).

Синапс (Synapse) - точка контакту між нервовими волокнами, яка передає імпульси від клітини до клітини.

Колатерали - нервові закінчення аксонів, що контактують з іншими нейронами, утворюючи нові синапси на сомі та дендритах.

Нейрони є основними будівельними блоками нейронних мереж. Кожен нейрон моделює роботу біологічного нейрона в людському мозку. У нейронних мережах кожен нейрон має вхідні зв'язки, де вхідні сигнали (або дані) приходять, вони збираються, обробляються шляхом зважування та сумування, і, якщо результат перевищує певний поріг, нейрон видає вихідний сигнал, який потім передається до наступних шарів мережі. Кожен нейрон мережі взаємодіє з іншими нейронами, що дозволяє нейронним мережам навчатися відтворювати та розпізнавати патерни у вхідних даних, виконуючи завдання класифікації, регресії, або розпізнавання об'єктів на зображеннях.

Біологічний нейрон складається з тіла, дендритів (що приймають сигнали), та аксону (що передає сигнали). У штучних нейронних мережах ця аналогія

відображена у вхідних вагах, активаційній функції та передачі сигналу від одного шару до іншого.

Нейронні мережі включають кілька шарів: вхідний, приховані та вихідний (рис.1.2. [3]). У вхідному шарі нейрони отримують вхідні дані. У прихованих шарах дані обробляються та відфільтровуються через мережу внутрішніх зв'язків. У вихідному шарі відбувається остаточна обробка, яка дає вихідні результати. Наприклад, у задачі розпізнавання зображень, нейронна мережа може мати вхідний шар, що приймає пікселі зображення, приховані шари, які виконують обробку функцій, і вихідний шар, який вказує на ймовірність належності об'єкта до певного класу [1].

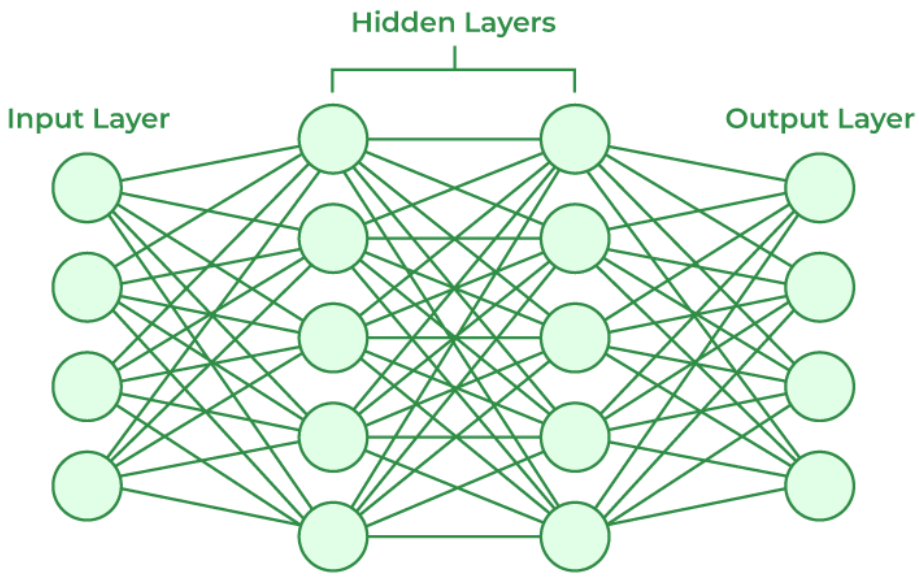

Рис. 1.2. Будова нейронної мережі

Нейронні мережі є лише одним з інструментів, використовуваних в машинному навчанні. Вони можуть використовуватися як складова частина багатьох алгоритмів, які перетворюють складні дані в простори, зрозумілі для комп'ютерів. Застосовуються вони для розв'язання широкого спектру проблем, від розпізнавання зображень та мови до фінансів та медичної діагностики.

1.2. Математична модель нейрона

Математичні моделі нейронів використовуються для апроксимації роботи реальних нейронів у біологічних системах. Одна з найпоширеніших моделей - модель МакКалока-Піттса, запропонована у 1943 році.

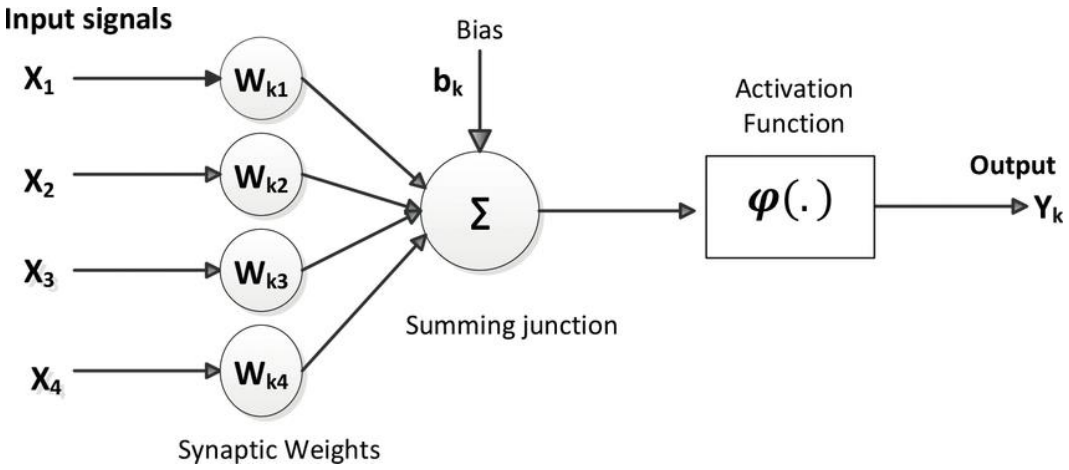

Рис. 1.3. Математична модель нейронів МакКалока-Піттса

Вхідними параметрами моделі нейрона (рис. 1.3. [5]) є скалярні значення входу p з коефіцієнтом ваги w, які формують значення wp, та число 1, яке множиться на значення зсуву b, вони передаються на суматор. Вихід суматора (вхід мережі) u передається функції активації (ФА) f, яка формує вихід нейрона y вигляді скалярного значення y й може бути як лінійною, так і нелінійною. Вихід нейрона обчислюють за формулою y = f (wp + b). Скалярні параметри w і b зазвичай налаштовуються на дані задачі за визначеним правилом навчання так, щоб нейрон, який відображає відношення вхід–вихід, досягнув заданої мети [**Помилка! Джерело посилання не знайдено.**]. Відповідно до біологічного нейрона вага w відповідає довжині синапсу, суматор разом із функцією активації зображує клітинне тіло нейрона, а вихід нейрона y зображує сигнал, який надходить на аксон.

Суматор враховує ваги вхідних сигналів та їх значення і робить лінійну комбінацію цих сигналів. Функція активації відповідає за те, коли нейрон активується та видає вихідний сигнал на основі обчислень, проведених суматором. Одна з типових функцій активації - сигмоїда, яка приймає зважену

суму вхідних сигналів та повертає значення від 0 до 1, що відображає активацію або вимкнення нейрона.

Такі моделі можуть використовуватися в штучних нейронних мережах для моделювання роботи реальних нейронів, розв'язання задач машинного навчання, класифікації даних тощо.

1.3. Типи нейронних мереж. Згорткові нейронні мережі (CNN)

Для повного розуміння ролі згорткових нейронних мереж (CNN) у галузі машинного навчання, варто розглянути різні типи нейронних мереж, кожен з яких володіє унікальними архітектурними особливостями та підходами до вирішення задач обробки даних.

Перцептрон (Perceptron) є найпростішою формою нейронної мережі і складається з одного шару нейронів. Він здатний розв'язувати лише лінійно роздільні задачі. Перцептрон складається з входів, вагових коефіцієнтів, функції активації та виходу.

Багатошаровий перцептрон (Multilayer Perceptron, MLP) MLP складається з кількох шарів нейронів: вхідного, одного або більше прихованих шарів і вихідного шару (рис. 1.4. [**Помилка! Джерело посилання не знайдено.**]). Кожен нейрон у шарі пов'язаний з усіма нейронами наступного шару, утворюючи повнозв'язну мережу. Використовується функція активації (наприклад, ReLU або сигмоїдна функція) для обробки даних. MLP може розв'язувати нелінійні задачі класифікації та регресії.

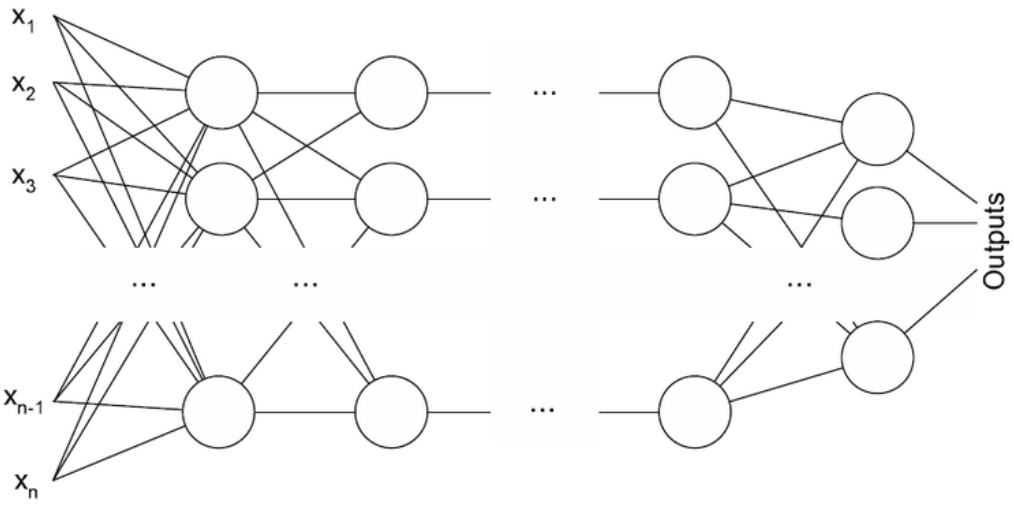

Рис. 1.4. Схема багатошарового перцептрона

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) є типом нейронних мереж, які відрізняються здатністю зберігати та обробляти інформацію у часі завдяки наявності петель зворотного зв'язку (рис. 1.5. [7]). Ці мережі ідеально підходять для роботи з послідовностями даних, таких як тексти, аудіо, відео або часові ряди, оскільки вони можуть враховувати контекст попередніх елементів послідовності при обробці поточного елемента.

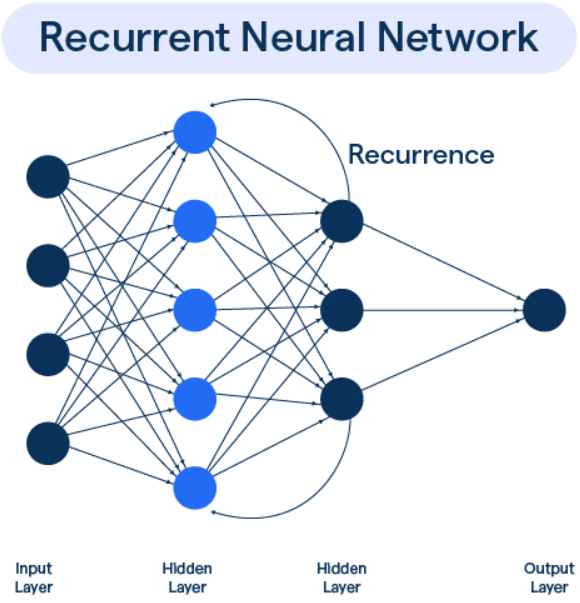

Рис. 1.5. Схема рекурентної нейронної мережі

Основна ідея RNN полягає в тому, що на кожному часовому кроці мережа отримує на вхід не тільки поточний елемент послідовності, але й стан, який зберігає інформацію про попередні кроки. Цей стан, або прихований стан, оновлюється на кожному кроці шляхом застосування функції активації до лінійної комбінації вхідних даних і попереднього стану. Це дозволяє мережі зберігати важливу інформацію про попередні елементи послідовності та використовувати її для прийняття рішень на наступних кроках.

Однак класичні RNN мають певні обмеження, особливо коли йдеться про обробку довготривалих залежностей. Однією з основних проблем є "втрата градієнту" (vanishing gradient problem), коли градієнти, необхідні для навчання мережі, стають надто малими, що призводить до значного зниження ефективності навчання. Це ускладнює завдання навчання RNN з урахуванням інформації з далеких попередніх кроків у послідовності.

Для вирішення цієї проблеми були розроблені вдосконалені архітектури, такі як довго-короткострокова пам'ять **(Long Short-Term Memory, LSTM)** та комірка з регульованим рекурентним блоком **(Gated Recurrent Unit, GRU)** (рис. 1.6. [9])**.** Ці архітектури мають спеціальні механізми для зберігання та управління інформацією протягом тривалих інтервалів часу.

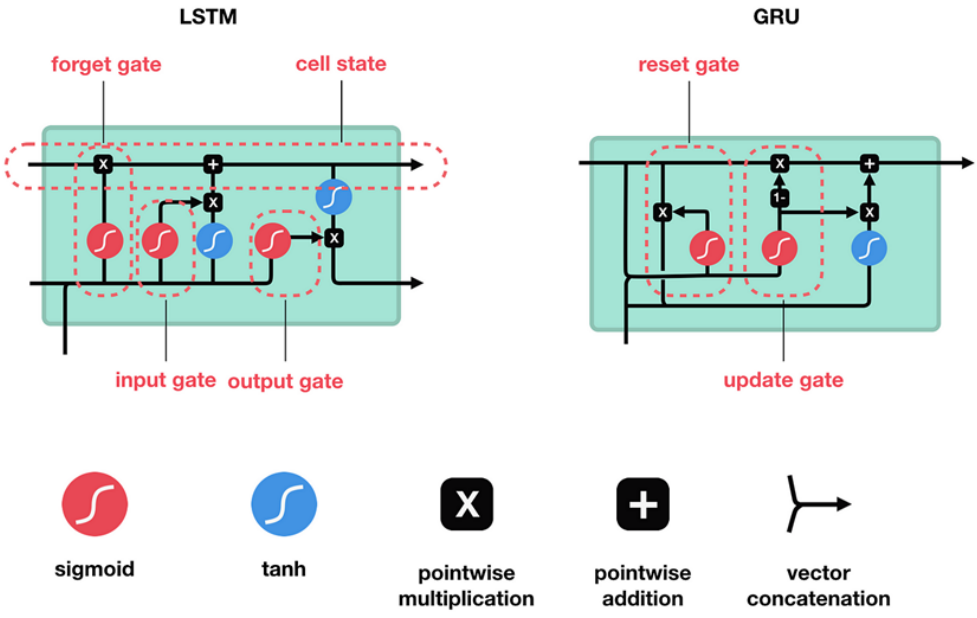

Рис. 1.6. Схема LSTM та GRU

LSTM включають в себе структури, звані "комірками пам'яті", які можуть зберігати інформацію протягом довгих періодів часу. Кожна комірка має три основні компоненти: входову, вихідну та забувальну гейти. Ці гейти контролюють потік інформації всередині комірки, дозволяючи моделі динамічно вирішувати, яку інформацію зберігати, яку оновлювати, а яку видаляти. Це дозволяє LSTM ефективно справлятися з проблемою втрати градієнту та зберігати довготривалі залежності в даних.

GRU є спрощеною версією LSTM, яка також ефективно вирішує проблему втрати градієнту, але має менш складну архітектуру. GRU об'єднує входову та забувальну гейти в один гейт, зменшуючи кількість параметрів і обчислювальних витрат, при цьому зберігаючи здатність моделювати довготривалі залежності[9].

Таким чином, RNN та їх вдосконалені варіанти, такі як LSTM і GRU, є потужними інструментами для роботи з послідовними даними, дозволяючи враховувати контекст та залежності у часі, що робить їх незамінними для багатьох задач машинного навчання, зокрема обробки природної мови та аналізу часових рядів.

Генеративно-змагальні мережі (Generative Adversarial Networks, GAN) є інноваційною архітектурою в галузі машинного навчання, яка дозволяє створювати нові, реалістичні зразки даних(рис. 1.7. [10]). Вони складаються з двох основних компонентів: генератора і дискримінатора, які працюють у змагальному режимі.

Генератор є нейронною мережею, яка створює підроблені зразки даних, намагаючись зробити їх настільки схожими на справжні, наскільки це можливо. Він приймає на вхід випадковий шум або латентний вектор і перетворює його в дані, що мають ту ж структуру, що й справжні дані (наприклад, зображення, текст або звук). Мета генератора – обдурити дискримінатор, змушуючи його приймати підроблені зразки за справжні.

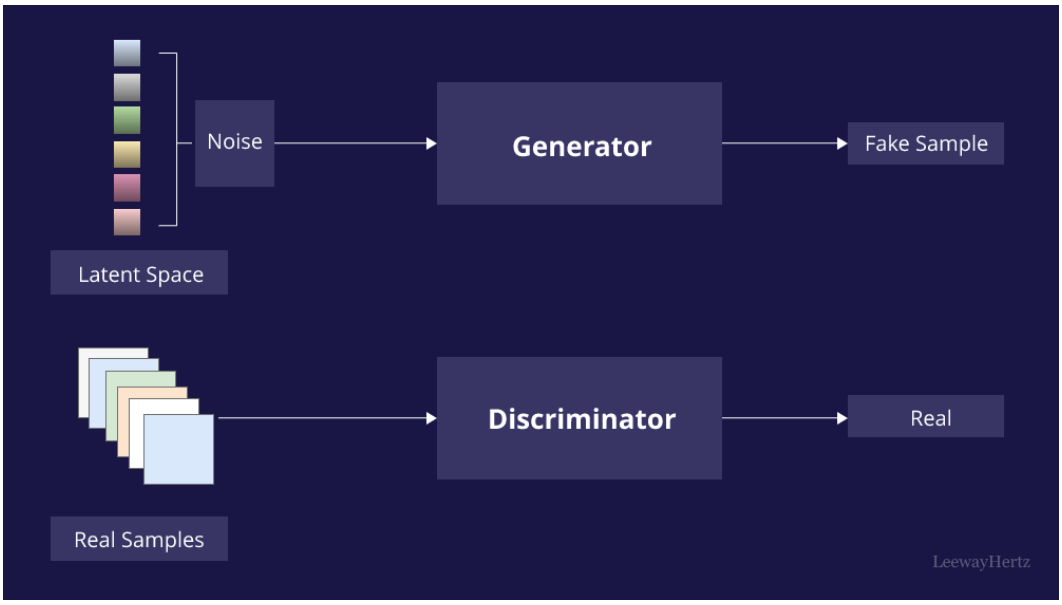

Рис. 1.7. Схема генеративно-змагальної мережі

Дискримінатор, з іншого боку, є нейронною мережею, яка намагається відрізнити справжні зразки даних від підроблених, створених генератором. Він приймає на вхід як справжні зразки, так і підроблені, і виводить ймовірність того, що вхідний зразок є справжнім[10]. Мета дискримінатора – правильно класифікувати всі зразки, надані на вхід, тим самим "навчаючи" генератор покращувати якість підроблених даних.

Глибоке навчання - це галузь машинного навчання, яка використовує нейронні мережі з численними шарами для автоматичного вивчення представлення даних у високорівневому абстрактному рівні. Однією з ключових архітектур для обробки візуальних даних, зокрема для розпізнавання знаків на зображеннях, є згорткові нейронні мережі (Convolutional Neural Networks, CNN).

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) представляють собою модель, спеціально створену для роботи з візуальними даними, зокрема зображеннями. Вони складаються з ряду шарів, які здійснюють обробку даних на різних рівнях абстракції. Основна особливість цих мереж полягає у використанні фільтрів, які застосовуються на різних шарах для виявлення візуальних або просторових характеристик зображення [11].

Фільтри у CNN дозволяють автоматично виділяти основні візуальні або текстурні особливості, такі як краї, форми, текстури тощо. Це досягається шляхом застосування математичних операцій (зокрема згортки) до регіонів зображення. Після цього знаходяться значення, які вказують на наявність певних характеристик у кожному регіоні зображення. Ці виявлені характеристики формують ієрархію репрезентацій, де ранні шари виявляють прості особливості, такі як лінії або кут, тоді як більш високі шари можуть реагувати на більш складні або абстрактні ознаки, наприклад, форми об'єктів. Цей процес використовується для створення моделі, яка може автоматично визначати ознаки та особливості зображення, що у свою чергу може бути використано для класифікації, виявлення об'єктів або розпізнавання сцен у зображеннях.

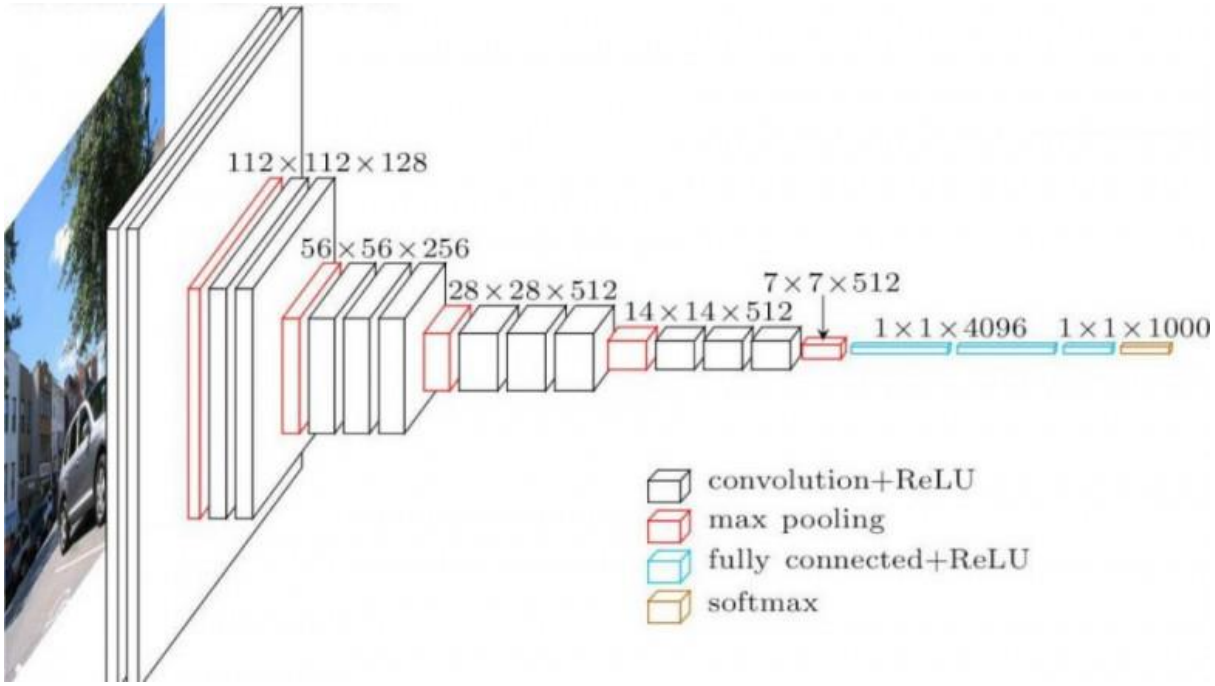

Рис.1.8. Глибоке навчання. Розбиття на шари згорткових нейронних мереж (CNN)

Основні складові частини згорткових нейронних мереж (CNN) (рис. 1.8. [12]):

● Convolution+ReLU: Це перший етап CNN. "Convolution + ReLU" - це комбінація двох операцій, які використовуються в згорткових нейронних мережах (CNN).

1. Convolution (згортка): Це операція, де фільтри або ядра проходять по

вхідному зображенню для виявлення різних візуальних особливостей. Згортка допомагає виділити різні ознаки, такі як краї, форми, або текстури, шляхом множення значень пікселів на фільтр та їх сумування. Це дозволяє створити карти ознак, що виокремлюють ключові аспекти вхідного зображення.

2. ReLU (Rectified Linear Unit): Це функція активації, яка застосовується після згортки. Вона приймає вихідні значення згортки і замінює негативні значення на нуль. Це дозволяє моделі вчитися нелінійностям у вхідних даних та вирішувати проблему втрати градієнту (vanishing gradient problem), полегшуючи швидкість навчання та прискорюючи збіжність моделі.

Ця комбінація Convolution + ReLU є основним блоком у згорткових шарах нейронних мереж та допомагає в них виявляти та передавати важливі ознаки для подальшої обробки та аналізу[13].

● Pooling (Max pooling): Це метод підбору найбільш значущого значення з певного регіону зображення. Операція max pooling розділяє кожну карту ознак на невеликі регіони та вибирає найбільше значення (максимальне) з кожного регіону. Це дозволяє зменшити просторовий обсяг даних шляхом збереження найважливіших ознак та підвищення інваріантності до невеликих трансляцій об'єктів у вхідних даних. Max pooling допомагає скоротити кількість параметрів у мережі, зменшити обсяг обчислень та уникнути перенавчання (overfitting), сприяючи виявленню ключових ознак в зображеннях та забезпечуючи просторову інваріантність.

● Classification (Fully Connected + ReLU): Після згортки та пулінгу, використовуються повністю з'єднані (fully connected) шари. Вони використовуються для того, щоб отримані ознаки і інформація були передані до кінцевого етапу моделі для класифікації або визначення. ReLU також може застосовуватися на цьому етапі для нелінійної обробки даних.

● Softmax: Це останній етап, де використовується функція активації Softmax. Softmax - це функція активації, що використовується для прогнозування ймовірностей різних класів у задачах класифікації. Вона приймає вектор числових

значень і перетворює їх у ймовірності для кожного класу, які в сумі складають 1. Це дозволяє отримати ймовірнісний розподіл між класами, призначеними для класифікації.

1.4. Використання штучних нейронних мереж для розпізнавання об'єктів на зображеннях

Проблема розпізнавання об'єктів на зображенні полягає в розумінні та ідентифікації об'єктів, які знаходяться на зображенні. Це складна задача, оскільки вимагає виявлення та класифікації об'єктів різних розмірів, форм, кольорів та контексту у реальному часі. Наприклад, система розпізнавання облич може потребувати відрізнення людей на зображенні з різних кутів, освітленням або емоціями.

Однією з основних проблем є точність розпізнавання об'єктів у великому наборі даних, оскільки модель може бути недостатньо навчена або не мати достатньої репрезентативності для різних умов. Іншою проблемою є здатність моделі пристосовуватись до нових об'єктів або змін в даних, які вона не бачила під час навчання. Також до викликів входить оптимізація швидкості та ефективності роботи алгоритмів розпізнавання об'єктів, особливо у реальному часі, коли потрібно обробляти великий потік даних, наприклад, в системах моніторингу чи автоматизації.

YOLOv8 — це структура ШІ, яка підтримує численні завдання комп’ютерного зору [14]. Структуру можна використовувати для виявлення, сегментації, класифікації та позиціювання. Кожне з цих завдань має різну мету та варіант використання.

Рис.1.9. Виявлення об’єктів

Виявлення об’єктів — це завдання, яке передбачає визначення розташування та класу об’єктів у зображенні чи відео (рис.1.9. [15]). Результатом детектора об’єктів є набір обмежувальних прямокутників, які охоплюють об’єкти на зображенні, разом із мітками класів і балами достовірності для кожної рамки. Виявлення об’єктів є хорошим вибором, коли вам потрібно ідентифікувати цікаві об’єкти в сцені, але вам не потрібно точно знати, де знаходиться об’єкт або його точну форму.

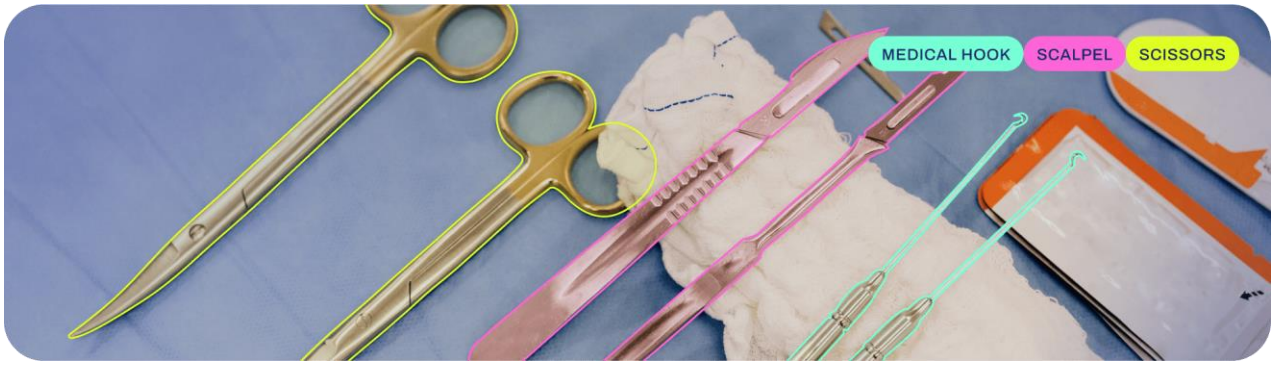

Рис.1.4. Сегментація об’єктів

Сегментація екземплярів йде далі, ніж виявлення об’єктів, і передбачає ідентифікацію окремих об’єктів на зображенні та їх сегментування від решти зображення (рис.1.10. [14]).

Результатом моделі сегментації екземпляра є набір масок або контурів, які окреслюють кожен об’єкт на зображенні разом із мітками класів і балами достовірності для кожного об’єкта. Сегментація екземплярів корисна, коли вам

потрібно знати не лише розташування об’єктів на зображенні, але й їхню точну форму.

Рис.1.11. Класифікація об’єктів

Класифікація зображень є найпростішим із трьох завдань і передбачає класифікацію всього зображення в один із набору попередньо визначених класів(рис.1.11. [14]). Результатом роботи класифікатора зображень є одна мітка класу та оцінка достовірності . Класифікація зображень корисна, коли вам потрібно лише знати, до якого класу належить зображення, і не потрібно знати, де розташовані об’єкти цього класу чи їх точна форма.

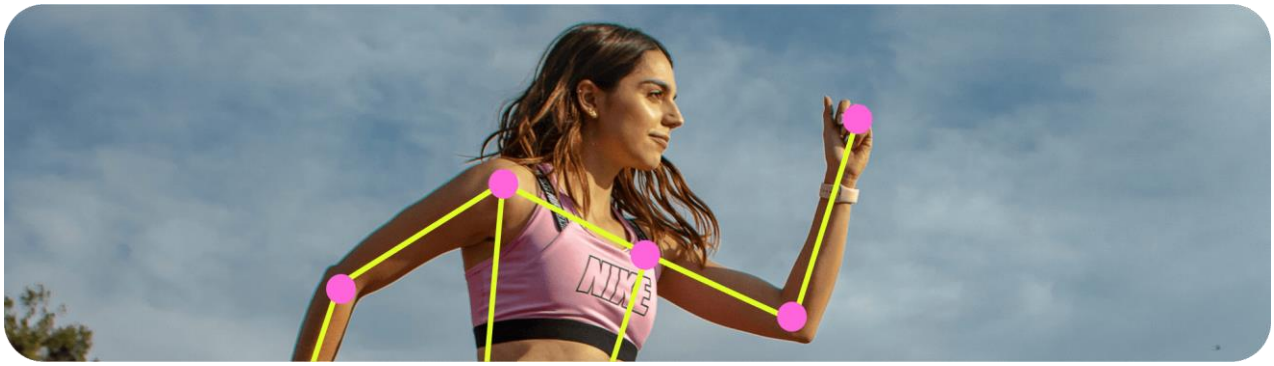

Рис.1.12. Позиціювання об’єктів

Позиціювання — це завдання, яке включає визначення розташування певних точок на зображенні, які зазвичай називаються ключовими точками (рис.1.12. [16]). Ключові точки можуть представляти різні частини об’єкта, такі як стики, орієнтири або інші відмінні риси. Розташування ключових точок зазвичай представлено у вигляді набору 2D [x, y]або 3D [x, y, z] координат [14].

Результатом моделі позиціювання є набір точок, які представляють ключові точки об’єкта на зображенні, зазвичай разом із оцінками достовірності для кожної точки. Позиціювання є хорошим вибором, коли вам потрібно визначити конкретні частини об’єкта в сцені та їх розташування відносно одна одної.

1.5. Розробка додатків для мобільних пристроїв. Загальні положення

Для створення додатків для мобільних пристроїв знадобляться знання наступних технологій [17]:

- CSS і HTML, середовище розробки Cocoa / Xcode, мови програмування Objective-C / C ++, Java; HTTP, XML, принципи об’єктно-орієнтованого програмування, СУБД;
- Android, iOS SDK, шаблони проектування, поширені бібліотеки і архітектуру iOS, Android, скриптові мови програмування (Ruby, Python), принципи клієнт-серверної моделі взаємодії додатків;
- вимоги до релізів додатків в AppStore і Google Play;
- мати навички роботи з Core Data (фірмової локальною базою даних від Apple, яка побудована за типом SQL);

Для створення мобільних додатків спочатку визначають платформу розробки і обирають відповідну мову програмування. Для розробки Android додатків найчастіше використовують Java або Kotlin. Для iOS додатків використовуються Objective-C і Swift, причому Swift стає більш популярною. Деякі додатки для обох платформ можуть бути написані мовою C++. Іноді для розробки мобільних додатків використовують інші мови, такі як Python, C#, або Unity, а також спеціалізовані фреймворки та бібліотеки.

Існує два методи розробки мобільних додатків:

- нативна розробка;
- кросплатформенна, або гібридна розробка (ReactNative, Flutter, Xamarin).

Нативна розробка мобільних додатків полягає у створенні програмного забезпечення, яке розробляється спеціально для конкретної мобільної платформи,

використовуючи мови програмування та інструменти, що підтримуються цією платформою. Для розробки додатків під iOS використовуються мови програмування Swift та Objective-C, а також середовище розробки Xcode. Для розробки додатків під Android використовуються мови програмування Java та Kotlin, а також середовище розробки Android Studio.

Однією з основних переваг нативної розробки є висока продуктивність додатків. Оскільки додаток розробляється безпосередньо для конкретної платформи, він може максимально ефективно використовувати апаратні ресурси пристрою, такі як процесор, графічний процесор та оперативну пам'ять. Це забезпечує швидку реакцію на дії користувача та плавну роботу додатка. Крім того, нативні додатки мають доступ до всіх функцій і можливостей операційної системи та апаратного забезпечення, що дозволяє реалізувати широкий спектр функціональності.

Іншою важливою перевагою нативної розробки є можливість створення інтерфейсу, який відповідає рекомендаціям та стандартам конкретної платформи. Це забезпечує високу якість користувацького досвіду та інтуїтивну зрозумілість інтерфейсу для користувачів. Нативні додатки також мають кращу інтеграцію з екосистемою платформи, що дозволяє використовувати сервіси та компоненти операційної системи, такі як повідомлення, доступ до камери та сенсорів, а також роботу в офлайн-режимі.

Проте нативна розробка має і певні недоліки. Одним з головних є високі витрати на розробку та підтримку додатків для різних платформ. Оскільки для кожної платформи необхідно розробляти окремий додаток, це вимагає більше часу та ресурсів. Крім того, для кожної платформи необхідні окремі команди розробників з відповідними знаннями та навичками. Це також ускладнює процес оновлення та підтримки додатків, оскільки зміни потрібно вносити в кожну версію додатка окремо.

Кросплатформенна розробка мобільних додатків полягає у створенні програмного забезпечення, яке може функціонувати на різних мобільних операційних системах з використанням єдиного базового коду. Для реалізації

цього підходу застосовуються web-технології, такі як HTML, CSS і JavaScript, а також спеціалізовані фреймворки, наприклад, React Native, Flutter і Xamarin.

Основною перевагою кросплатформенної розробки є зниження витрат на розробку та підтримку додатків, оскільки немає необхідності створювати окремі версії для кожної платформи. Крім того, це дозволяє скоротити час виходу продукту на ринок, оскільки один розробник може створювати додаток для декількох платформ одночасно.

Однак, кросплатформенна розробка має і певні недоліки. Зокрема, можуть виникати труднощі з оптимізацією додатків для специфічних платформ, що може призвести до зниження продуктивності та затримок у реакції на дії користувача. Інтерфейс таких додатків може виглядати менш органічно в порівнянні з нативними рішеннями, що вимагає додаткових зусиль для досягнення прийнятного рівня зручності користування. Також можуть виникати проблеми з доступом до деяких специфічних функцій апаратної частини пристроїв, що обмежує можливості додатка.

1.5.1. Розробка Android додатків

Розробка програмного забезпечення для Android - це процес, за допомогою якого створюються додатки для пристроїв, що працюють під управлінням операційної системи Android. Google заявляє, що [18] "додатки для Android можна писати мовами Kotlin, Java та C++" за допомогою комплекту для розробки програмного забезпечення (SDK) для Android, хоча використання інших мов також можливе. Усі мови віртуальних машин, відмінні від Java, такі як Go, JavaScript, C, C++ або асемблер, потребують допомоги мовного коду JVM, який може бути наданий інструментами, ймовірно, з обмеженою підтримкою API. Деякі мови програмування та інструменти дозволяють підтримувати крос-платформні додатки (тобто для Android та iOS). Сторонні інструменти, середовища розробки та мовна підтримка також продовжують розвиватися і розширюватися з моменту випуску першого SDK у 2008 році. Офіційним механізмом розповсюдження додатків для Android серед кінцевих користувачів є

Google Play; він також дозволяє поетапний випуск додатків, а також розповсюдження попередніх версій додатків серед тестувальників.

Пакет Android (Android package), який є архівним файлом із суфіксом .apk, містить вміст програми для Android, необхідний під час виконання, і це файл, який пристрої на базі Android використовують для встановлення програми.

Пакет програми для Android (Android App Bundle), який є архівним файлом із суфіксом .aab, містить вміст проекту програми для Android, включно з деякими додатковими метаданими, які не є обов'язковими під час виконання. AAB - це формат публікації, і його не можна встановити на пристрої Android. Він відкладає генерацію та підписання APK на пізніший етап.

Наприклад, відбувається процес поширення додатку через Google Play, сервери Google Play генерують оптимізовані APK, які містять лише ті ресурси та код, які потрібні конкретному пристрою, що запитує встановлення додатку.

Кожна програма для Android живе у власній пісочниці безпеки, захищеній наступними функціями безпеки Android:

Операційна система Android - це багатокористувацька Linux-система, в якій кожна програма є окремим користувачем.

За замовчуванням система присвоює кожному додатку унікальний ідентифікатор користувача Linux, який використовується тільки системою і невідомий додатку. Система встановлює дозволи для всіх файлів у програмі таким чином, щоб тільки ідентифікатор користувача, присвоєний цій програмі, міг отримати до них доступ.

Кожен процес має власну віртуальну машину (ВМ), тому код програми виконується ізольовано від інших програм.

За замовчуванням, кожна програма запускається у власному процесі Linux. Система Android запускає процес, коли потрібно виконати будь-який з компонентів програми, а потім закриває процес, коли він більше не потрібен або коли система повинна звільнити пам'ять для інших програм.

Система Android реалізує принцип найменших привілеїв. Це означає, що кожна програма за замовчуванням має доступ лише до тих компонентів, які їй

потрібні для роботи, і не більше. Це створює дуже безпечне середовище, в якому програма не може отримати доступ до частин системи, на які вона не має дозволу.

1.5.2. Розробка IOS додатків

IOS SDK (Software Development Kit) забезпечує можливість розробки мобільних додатків для операційної системи iOS.

Під час розробки iPhone до його презентації у 2007 році генеральний директор Apple Стів Джобс (рис.1.13.) не планував дозволяти стороннім розробникам створювати власні додатки для iOS, спрямовуючи їх на створення веб-додатків для браузера Safari. Проте, у відповідь на реакцію розробників, компанія переглянула своє рішення. У жовтні 2007 року Джобс оголосив, що до лютого 2008 року Apple випустить набір для розробки програмного забезпечення, доступний для розробників. Цей набір SDK був випущений 6 березня 2008 року[19].

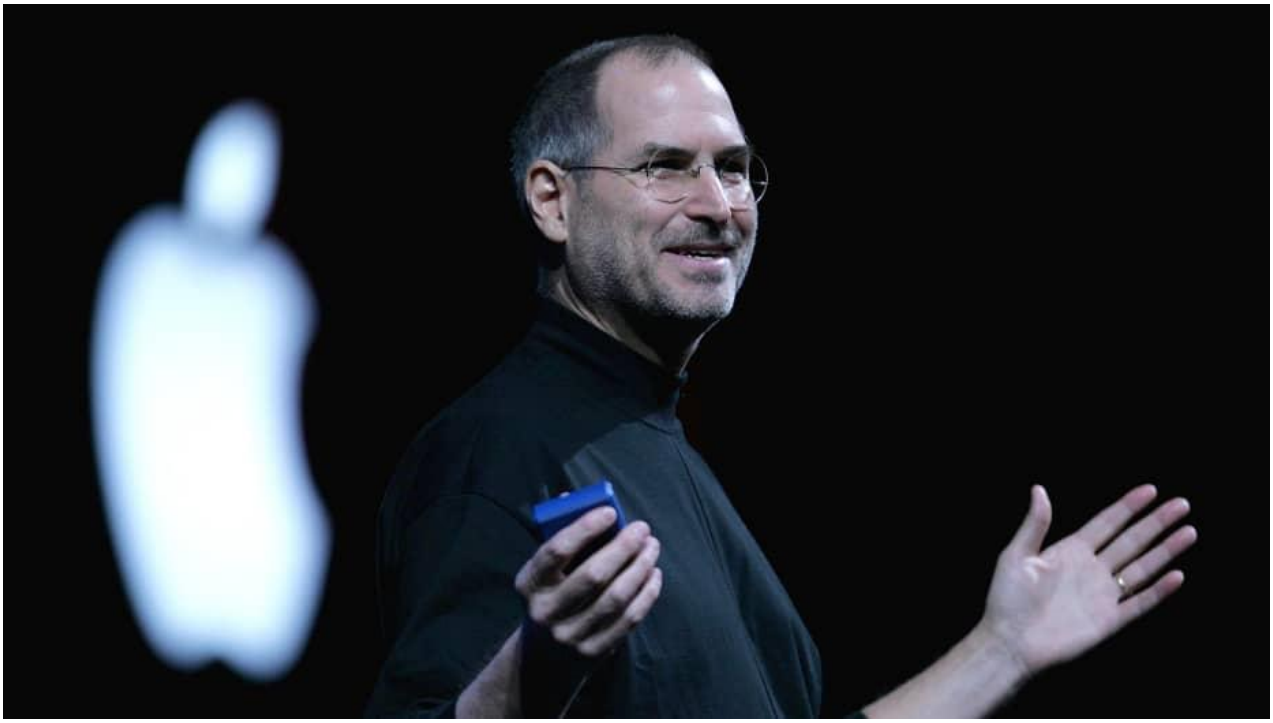

Рис.1.13. Стів Джобс презентує продукт Apple

SDK можна безкоштовно завантажити для користувачів персональних комп'ютерів Mac, тоді як для комп'ютерів під управлінням Microsoft Windows він недоступний. SDK містить інструменти, що надають розробникам доступ до

різноманітних функцій і сервісів пристроїв iOS, таких як апаратні та програмні атрибути. До складу SDK також входить симулятор iPhone, що дозволяє імітувати зовнішній вигляд пристрою на комп'ютері під час розробки. Нові версії SDK випускаються разом із новими версіями iOS. Для тестування програм, отримання технічної підтримки та розповсюдження додатків через App Store розробники повинні підписатися на Програму для розробників Apple.

У поєднанні з **Xcode, iOS SDK** дозволяє розробникам писати додатки для iOS, використовуючи офіційно підтримувані мови програмування, включаючи **Swift** та **Objective-C**. Крім того, інші компанії розробили інструменти, що дозволяють створювати нативні додатки для iOS з використанням альтернативних мов програмування:

Java. У 2008 році компанія Sun Microsystems оголосила про намір випустити віртуальну машину Java (JVM) для iOS, засновану на платформі Java Micro Edition, що дозволило б запускати Java-додатки на iPhone та iPod Touch [19]. Проте, умови угоди SDK, такі як заборона стороннім додаткам працювати у фоновому режимі, завантажувати код з інших джерел або взаємодіяти з додатками третіх сторін, значно ускладнили реалізацію цього плану без співпраці з Apple. Sun також працювала з компанією Innaworks над впровадженням Java на iPhone. Незважаючи на відсутність офіційної підтримки з боку Apple, витік прошивки iPhone 2007 року виявив наявність процесора ARM з підтримкою Jazelle для вбудованого виконання Java.

.NET. У вересні 2009 року компанія Novell оголосила про успішну розробку MonoTouch, програмного фреймворку, який дозволяє розробникам писати нативні додатки для iPhone мовами програмування C# та .NET, зберігаючи при цьому сумісність з вимогами Apple [**Помилка! Джерело посилання не знайдено.**].

Flash. Операційна система iOS не підтримує Adobe Flash. Незважаючи на наявність двох версій цього програмного забезпечення, Flash і Flash Lite, Apple не вважає їх придатними для iPhone, стверджуючи, що повний Flash "занадто повільний, щоб бути корисним", а Flash Lite "не може використовуватися з Інтернетом". У жовтні 2009 року Adobe оголосила, що майбутнє оновлення її

Creative Suite буде містити компонент, який дозволить розробникам створювати нативні програми для iPhone за допомогою інструментів розробки Flash. Це програмне забезпечення було офіційно випущено як частина колекції професійних додатків Creative Suite 5 [21].

У квітні 2010 року Apple внесла зміни до своєї Угоди для розробників iPhone, вимагаючи використовувати лише "схвалені" мови програмування для публікації додатків в App Store і забороняючи додатки, створені за допомогою сторонніх інструментів розробки. Це рішення зачепило, зокрема, інструмент Adobe Packager, який перетворював Flash-додатки на додатки для iOS. Після реакції розробників та новин про можливе антимонопольне розслідування, Apple переглянула свою угоду у вересні, дозволивши використання сторонніх інструментів розробки [22].

Mac Catalyst. Mac Catalyst, спочатку відомий як "Project Marzipan", допомагає розробникам перенести досвід роботи з додатками для iPadOS на macOS. Цей інструмент полегшує адаптацію додатків, розроблених для пристроїв iPadOS, на комп’ютери Mac, уникаючи необхідності переписування основного програмного коду для обох платформ.

1.5.3 Кросплатформенна розробка мобільних додатків

Кросплатформена розробка мобільних додатків - це підхід, коли один і той самий код може бути використаний для створення додатків для різних платформ, таких як Android та iOS. Цей метод дозволяє економити час і зусилля, оскільки розробникам не потрібно писати окремий код для кожної платформи, а вони можуть сконцентруватися на створенні одного універсального додатку.

Одним з основних інструментів для кросплатформеної розробки є фреймворки, такі як React Native, Xamarin, Flutter та інші. Ці фреймворки надають розробникам можливість використовувати знайомі мови програмування, такі як JavaScript, C# або Dart, для створення додатків, які працюватимуть як на Android, так і на iOS.

Flutter. Flutter, створений компанією Google, є кросплатформеним середовищем розробки, що використовує мову програмування Dart. Flutter

підтримує вбудовані функції, такі як служби визначення місцезнаходження, функціональність камери та доступ до жорсткого диска. У випадку, якщо необхідна функція не підтримується у Flutter, розробники можуть написати код для конкретної платформи, використовуючи технологію Platform Channel.

Додатки, створені за допомогою Flutter, використовують спільні UX та UI шари, що може призводити до незначних відмінностей у відчутті нативності додатків. Однією з найвагоміших переваг цього фреймворку є функція гарячого перезавантаження, яка дозволяє розробникам миттєво бачити внесені зміни.

Flutter може бути оптимальним вибором у таких ситуаціях:

- Потреба в обміні компонентами інтерфейсу між додатками при збереженні їх нативного вигляду.
- Високі вимоги до продуктивності додатку, який сильно навантажує CPU/GPU.
- Необхідність розробки MVP (мінімально життєздатного продукту).

Серед найвідоміших додатків, створених за допомогою Flutter, є Google Ads, Xianyu від Alibaba, eBay Motors та Hamilton.

React Native. React Native надає доступ до численних сторонніх бібліотек інтерфейсів з готовими компонентами, що сприяє економії часу у процесі розробки. Так само як Flutter, він дозволяє миттєво бачити внесені зміни завдяки функції швидкого оновлення.

React Native є доцільним вибором для додатків у таких випадках:

- відносна простота додатку та очікувана легкість його роботи;
- команда розробників добре володіє JavaScript або React.

Серед відомих додатків, створених на основі React Native, є Facebook, Instagram, Skype та Uber Eats.

Kotlin Multiplatform. Kotlin Multiplatform – це технологія з відкритим вихідним кодом, розроблена JetBrains, що дозволяє розробникам ділитися кодом між різними платформами, зберігаючи переваги нативного програмування. Ключові переваги цієї технології включають:

- можливість повторного використання коду на Android, iOS, в Інтернеті,

на десктопних і серверних платформах, зберігаючи нативний код, якщо це необхідно;

- плавна інтеграція з існуючими проектами, що дозволяє використовувати специфічні для платформи API, одночасно отримуючи максимальну віддачу від нативної та кросплатформеної розробки;
- гнучкість у спільному використанні коду та можливість ділитися як логікою, так і інтерфейсом завдяки Compose Multiplatform, сучасному декларативному кросплатформеному фреймворку інтерфейсу користувача, створеному JetBrains.

Використання Kotlin Multiplatform дозволяє уникнути впровадження нової мови в кодову базу, якщо вже використовується Kotlin для Android, що знижує ризики, пов'язані з міграцією на цю технологію, у порівнянні з іншими альтернативами.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ

Для розробки програмної системи розпізнавання дорожніх знаків використовуються наступні технології: мова програмування Python, Roboflow, YOLOv8 (YOLOv9), Docker, Inference та Supervision, Google Colab Pro та React Native.

Python грає важливу роль у створенні моделей нейронних мереж для автоматичного розпізнавання дорожніх знаків на зображеннях. У контексті цієї теми, Python використовується для реалізації алгоритмів машинного навчання та глибокого навчання, зокрема згорткових нейронних мереж (CNN), що є ефективними у розпізнаванні образів. Він виступає як інструмент для обробки зображень, підготовки даних, навчання моделей, оцінки їх продуктивності та оптимізації алгоритмів розпізнавання. Python має широкий вибір бібліотек, таких як ultralytics(YOLOv8, YOLOv9), Inference, Supervision та багато інших, які дозволяють легко розробляти та вдосконалювати моделі для точного розпізнавання дорожніх знаків. Ця мова програмування забезпечує швидкий прототипування та експериментування завдяки своїй простоті, величезній кількості доступних ресурсів, а також підтримці активної спільноти розробників. Python дозволяє розробникам легко працювати з алгоритмами навчання та забезпечує гнучкість у реалізації та оптимізації моделей для розпізнавання дорожніх знаків на зображеннях.

2.1. Середовище для анотування Roboflow

Roboflow - це платформа, яка спеціалізується на обробці та підготовці наборів даних для застосувань у сфері комп'ютерного зору та машинного навчання. Її основна мета полягає в спрощенні та оптимізації процесу роботи з наборами зображень, необхідними для навчання та вдосконалення моделей машинного навчання. Roboflow пропонує різноманітні інструменти для редагування та підготовки даних, включаючи анотацію зображень, а також

зручний інтерфейс для управління різними версіями та форматами даних. Платформа також надає можливості аугментації даних, дозволяючи розширити набори даних шляхом додавання різноманітних варіацій та змін до початкових зображень [23].

Roboflow грає важливу роль у підготовці даних для моделей нейронних мереж, зокрема в контексті автоматичного розпізнавання дорожніх знаків на зображеннях. Однією з ключових переваг Roboflow є зручність інтеграції з різними фреймворками та платформами машинного навчання. Платформа забезпечує можливість експорту даних у формати, які легко розуміються та використовуються різними інструментами машинного навчання. Іншою важливою характеристикою Roboflow є його універсальність та придатність для різних застосувань у сфері комп'ютерного зору. Від візуального розпізнавання об'єктів до аналізу зображень, платформа забезпечує гнучкість та інструменти, необхідні для розвитку різноманітних проектів.

Загалом, Roboflow створений з метою спростити та оптимізувати процес підготовки наборів даних для застосування в сфері комп'ютерного зору та машинного навчання, роблячи його доступнішим та продуктивнішим для розробників.

2.2. Нейронна мережа YOLOv8 (YOLOv9)

YOLO (You Only Look Once) є популярною архітектурою нейронної мережі для об'єктного виявлення на зображеннях. YOLOv8 - це одна з ітерацій цієї архітектури, що поєднує в собі швидкість та точність виявлення об'єктів. YOLO базується на ідеї "об'єктній локалізації та класифікації з одним проходом". Це означає, що алгоритм розпізнавання об'єктів відбувається шляхом одноразового прогону зображення через нейронну мережу. Основною перевагою цього підходу є його висока швидкість, оскільки обробка зображення відбувається за один прохід через мережу. YOLOv8 включає в себе ряд вдосконалень, спрямованих на покращення точності та швидкості виявлення об'єктів на зображеннях. Ця версія архітектури враховує різноманітні фактори

для підвищення точності, такі як використання різних методів аугментації даних для поліпшення навчання моделі, оптимізація алгоритмів та використання передових методів навчання глибоких нейронних мереж.

YOLOv8 використовується в області комп'ютерного зору для виявлення об'єктів на зображеннях у реальному часі. Завдяки своїй ефективності та точності в роботі з об'єктами на зображеннях, він знаходить застосування в різних галузях, таких як автономні автомобілі, системи відеоспостереження, розпізнавання об'єктів на зображеннях та багато іншого. YOLOv8 забезпечує високу швидкість і точність, що робить його цінним інструментом для розробки систем автоматичного розпізнавання дорожніх знаків. Це, в свою чергу, покращує безпеку на дорогах і оптимізує процес управління транспортними потоками.

З появою YOLOv9, ця архітектура отримала подальші покращення, які роблять її ще більш потужною та ефективною. YOLOv9 представляє собою поворотне подію в області виявлення об'єктів у реальному часі, пропонуючи значні поліпшення в плані ефективності, точності та адаптивності. Архітектура включає в себе оптимізації, що дозволяють підвищити точність виявлення об'єктів і знизити затримки в обробці зображень. Завдяки новим алгоритмічним удосконаленням та оптимізації використання апаратних ресурсів, YOLOv9 демонструє ще кращу продуктивність у реальному часі, що є критично важливим для застосувань, де швидкість реакції має вирішальне значення.

YOLOv9 вирішує важливі завдання завдяки таким інноваційним рішенням, як PGI (Progressive Generalization Improvement) та GELAN (Global Ensemble Learning and Normalization), що створює новий прецедент для майбутніх досліджень та застосувань у цій галузі. Завдяки цим вдосконаленням, YOLOv9 підвищує рівень продуктивності та точності, що робить його незамінним інструментом для створення систем автоматичного розпізнавання дорожніх знаків, покращуючи безпеку на дорогах та оптимізуючи процес управління транспортними потоками. Загалом, як YOLOv8, так і YOLOv9, завдяки своїй швидкості та точності в розпізнаванні об'єктів на зображеннях, є важливими

інструментами для створення систем автоматичного розпізнавання дорожніх знаків. Використання цих моделей може значно підвищити безпеку на дорогах та оптимізувати процес управління транспортними потоками, роблячи їх незамінними в сучасних інтелектуальних транспортних системах.

2.3. Хмарне середовище Google Colab Pro

Google Colab Pro є розширеною версією середовища для обчислень на базі хмарних технологій, яка надає користувачам суттєві переваги для виконання обчислювальних завдань та розробки проектів у галузі штучного інтелекту (ШІ) та машинного навчання (МН). Завдяки підтримці потужних обчислювальних ресурсів, таких як GPU та TPU, Google Colab Pro значно прискорює процес навчання моделей та проведення експериментів.

Однією з ключових переваг Google Colab Pro є безперервний доступ до високопродуктивних обчислювальних ресурсів. Це дозволяє користувачам ефективніше проводити навчання моделей, скорочуючи час обчислень та підвищуючи загальну продуктивність. Для задач, що потребують великих обсягів обчислень, таких як навчання глибоких нейронних мереж або обробка великих обсягів даних, можливість використовувати GPU та TPU є критично важливою. Це забезпечує значні переваги в контексті скорочення часу на навчання та покращення результатів моделей.

Крім того, Google Colab Pro підтримує широкий спектр інструментів для експериментування з моделями, оптимізації гіперпараметрів та проведення аналізу нейронних мереж. Це дозволяє швидше і ефективніше тестувати різні архітектури моделей та алгоритми, що сприяє пришвидшенню наукових досліджень та інновацій у сфері ШІ та МН.

З наукової точки зору, використання Google Colab Pro надає доступ до потужних інструментів для проведення складних обчислень та аналізу великих обсягів даних. Це відкриває нові можливості для досліджень, знижуючи бар'єри, пов'язані з доступом до високопродуктивних обчислювальних ресурсів. Таким чином, Google Colab Pro сприяє підвищенню ефективності дослідницької

діяльності та забезпечує суттєвий внесок у розвиток сучасних технологій штучного інтелекту.

2.4. Середовище для розгортання моделі нейронної мережі Docker

Docker - це платформа, яка надає можливість упаковувати, розгортати та виконувати програмне забезпечення в ізольованих контейнерах. Ця технологія забезпечує стандартизацію середовища виконання, що дозволяє запускати програми незалежно від операційної системи та різних інсталяційних середовищ. Контейнери Docker містять все необхідне для виконання програми, включаючи код, залежності, бібліотеки та середовище виконання. Вони ізольовані один від одного, що забезпечує безпеку та надійність в роботі програм. Це робить Docker ідеальним інструментом для розробки та розгортання програмного забезпечення у різних середовищах, включаючи локальні комп'ютери, сервери та хмарні обчислення. Однією з ключових переваг Docker є його портативність та легкість використання. Контейнери Docker можна легко створювати, запускати та масштабувати, що робить їх популярними серед розробників та команд розробки програмного забезпечення. Крім того, Docker надає можливість швидко розгортати та управляти додатками, що полегшує розробку та тестування програмних продуктів.

Для автоматичного розпізнавання дорожніх знаків на зображеннях, Docker та Inference може використовуватися для упакування та розгортання різних компонентів системи: від самої моделі машинного навчання до додаткових інструментів та середовища виконання. Це дозволить створити стандартизоване середовище для запуску системи розпізнавання знаків на різних платформах та середовищах, забезпечуючи при цьому консистентність та ефективність виконання програми.

2.5. Inference та Supervision

Roboflow Inference дозволяє розгортати моделі комп'ютерного зору швидше, ніж будь-коли. Що надає ця бібліотека:

- написання власної логіки виведення, яка часто вимагає знань з машинного навчання.
- управління залежностями.
- оптимізація продуктивності та використання пам'яті.
- написання тестів, щоб переконатися, що логіка виведення працює.
- написання кастомних інтерфейсів для запуску моделі через веб-камери та потоки, якщо система розгортається в реальному часі.
- Inference справляється з усіма цими завданнями, прямо з коробки.

За допомогою однієї установки pip і однієї команди для запуску Inference можна налаштувати сервер, який запускає точно налаштовану модель на будь-якому зображенні або відеопотоці.

Inference підтримує виявлення об'єктів, класифікацію, сегментацію екземплярів і навіть базові моделі (такі як CLIP і SAM). Ви можете навчити і розгорнути власну модель або скористатися однією з 50 000+ налаштованих моделей, якими ділиться спільнота.

2.6. Фреймворк для розробки мобільних додатків React Native

React Native — це відкритий програмний фреймворк, розроблений компанією Meta Platforms, Inc., призначений для створення додатків на платформах Android, Android TV, iOS, macOS, tvOS, Web, Windows та UWP [25]. Цей фреймворк дозволяє розробникам застосовувати технології React разом з нативними можливостями платформ. React Native активно використовується в компаніях Facebook, Microsoft та Shopify для розробки мобільних додатків, а також в Oculus для створення додатків віртуальної реальності.

Функціональні принципи React Native схожі з React, за винятком того, що в React Native відсутня маніпуляція DOM через Virtual DOM. Замість цього, він функціонує у фоновому процесі, який інтерпретує JavaScript-код безпосередньо на пристрої користувача і взаємодіє з нативною платформою через серіалізовані дані, використовуючи асинхронний та пакетний міст.

Компоненти в React Native обгортають наявний нативний код і взаємодіють

з нативними API через декларативну парадигму користувацького інтерфейсу React та JavaScript. У сучасних додатках React Native часто застосовується TypeScript, що забезпечує кращу типобезпеку порівняно з JavaScript.

Стилізація в React Native використовує синтаксис, подібний до CSS, але при цьому не застосовує HTML або CSS. Натомість для управління нативними представленнями використовуються повідомлення з JavaScript-потоку [**Помилка! Джерело посилання не знайдено.**].

React Native доступний для операційних систем Windows та macOS, і його підтримкою займається Microsoft.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ЛОКАЛІЗАЦІЇ ТА РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

3.1. Формування набору даних

Процес створення датасета для розпізнавання дорожніх знаків почався зі збору великого обсягу зображень, що містили різні типи дорожніх знаків. Ці зображення були отримані з різних джерел, таких як відеоспостереження на дорогах, бази даних дорожніх знаків або фотографії з мобільних пристроїв.

Рис.3.1. Процес збору зображень для датасета

Для формування датасету основна частина зображень була отримана через перегляд вулиць на Google Maps. Використання можливості панорамного перегляду кожного елемента вулиці дозволило детально розглядати кожен аспект і при певному масштабуванні виділити необхідні елементи, такі як дорожні знаки. Наступним кроком було зроблення знімка екрану, що створило готове зображення для використання у датасеті (рис.3.1).

Основна увага при формуванні датасету приділялася вулицям міста Рівне, з метою полегшення сприйняття оточення нейронною мережею на етапі тестування. Під час проходження по вулицям здійснювалися знімки екрану, зафіксовуючи дорожні знаки та їхнє розташування. Цей підхід дозволив

створити репрезентативний датасет із зображеннями дорожніх знаків у контексті різних вуличних сценаріїв та умов, сприяючи ефективнішому навчанню нейронної мережі.

Після отримання зображень був виконаний процес анотування. Кожен дорожній знак на зображенні був аккуратно виділений та підписаний з використанням спеціального програмного забезпечення для анотування. Цей етап вимагав великої уваги до деталей та точності, щоб забезпечити коректність анотацій для подальшого навчання моделі.

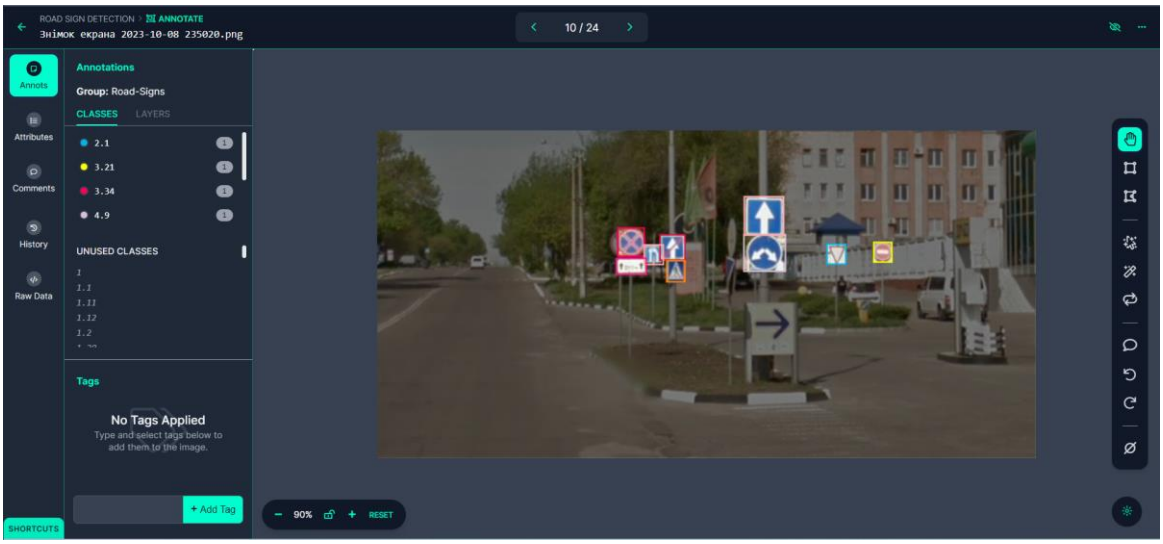

Рис.3.2. Процес анотування об’єктів на зображені

На початковому етапі моделі нейронної мережі було використано платформу Roboflow. Після реєстрації могли завантажувати зображення та відео для подальшої обробки. Процес анотування вимагав вручного виділення об'єктів на кожному зображенні, дотримуючись вказаних умов, і продовжувався, доки не було зібрано достатньо даних для навчання нейронної мережі (рис.3.2). Отримані дані слугували основою для навчання моделі розпізнавання дорожніх знаків [27].

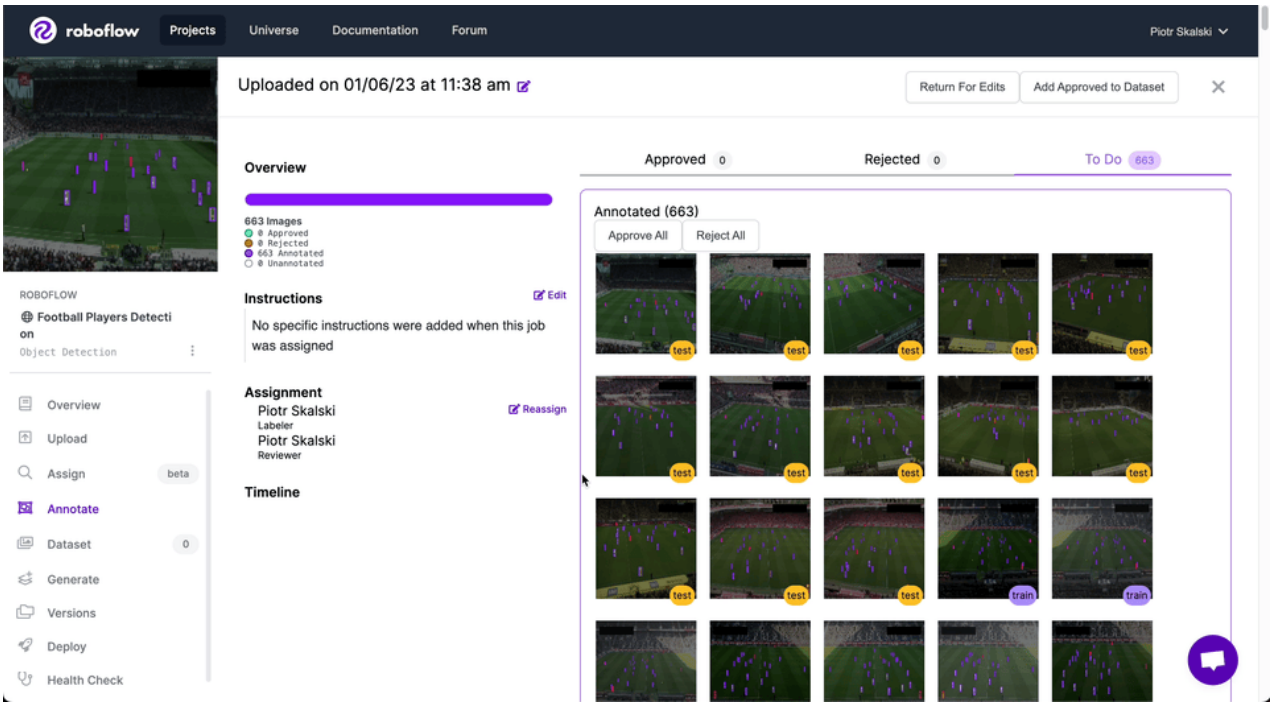

Рис.3.3. Генерація датасету. Аугментація даних

Після анотування необхідної кількості даних, був створений датасет, до якого було застосовано ряд аугментацій (рис.3.3).. Аугментація зображень — це етап, на якому застосовуються зміни до існуючих зображень у датасеті. Цей процес може допомогти покращити здатність моделі узагальнювати і, отже, ефективніше працювати з новими, невидимими зображеннями.

Roboflow підтримує такі види аугментацій:

- Відзеркалення
- Поворот на 90 градусів
- Випадковий поворот
- Випадкове обрізання
- Випадкове зміщення
- Розмиття
- Експозиція
- Випадковий шум і т.д

Крім того, було звернуто увагу на розмір датасету, його різноманітність та репрезентативність, щоб забезпечити адекватне навчання моделі на різних

умовах та типах дорожніх знаків.

Таким чином, процес створення датасету включав у себе етапи збору, обробки, анотування та перевірки даних для підготовки високоякісного та збалансованого набору даних для подальшого навчання моделі розпізнавання дорожніх знаків.

3.2. Навчання моделі YOLOv8

Задача навчання моделі YOLOv8 на власному датасеті є важкою і вимагає значних обчислювальних ресурсів. Однак, використовуючи Google Colab Pro, можна вирішити цю проблему.

Для початку, необхідно завантажити власний датасет та підготувати його до навчання. Наступним кроком є налаштування середовища для навчання моделі. У Google Colab Pro можна використовувати безкоштовні графічні процесори (GPU) або сплатити за використання більш потужних ресурсів, таких як TPU (рис.3.4).

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   64C    P0    31W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Рис.3.4. Використання ресурсів хмарних технологій

Після вибору конфігурації моделі YOLOv8 важливо уважно встановити параметри навчання. Це включає визначення оптимальної швидкості навчання (learning rate), яка визначає темп оновлення ваг моделі під час тренування. Вибір

правильної швидкості навчання є критичним для досягнення швидкого збіжності та уникнення перенавчання або низької точності моделі. Далі, алгоритм оптимізації (optimizer) визначає метод корекції ваг моделі для мінімізації функції втрат. Вибір оптимального алгоритму оптимізації, такого як Adam, SGD, RMSprop, впливає на ефективність навчання та стабільність збіжності моделі. Також важливо встановити розмір пакета (batch size), що визначає кількість прикладів даних, які використовуються перед оновленням ваг моделі. Відповідний розмір пакета може вплинути на швидкість навчання та стабільність процесу. Усі ці гіперпараметри взаємодіють між собою та мають великий вплив на результативність навчання моделі, тому їх відповідний вибір та налаштування дозволяють оптимізувати процес навчання, враховуючи особливості конкретного завдання та набору даних [14].

Після налаштування параметрів навчання, модель YOLOv8 ініціює процес тренування на навчальному наборі даних (рис.3.5). Процес тренування полягає у поступовій оптимізації внутрішніх параметрів моделі шляхом адаптації до вхідних даних. Модель в процесі навчання вдосконалює свою здатність розпізнавати дорожні знаки на зображеннях шляхом вирішення оптимізаційних задач, що полягають у мінімізації функції втрат та вдосконаленні уявлення про патерни, які вона виявляє на зображеннях. Цей процес базується на використанні алгоритмів оптимізації та апдейтів ваг моделі з метою підвищення її точності та універсальності у розпізнаванні дорожніх знаків.

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size	
50/50	14.4G	0.6571	0.5771	0.9339	32	800: 100% 65/65 [00:56<00:00, 1.14it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 4/4 [00:05<00:00, 1.48s/it]
	all	98	287	0.807	0.671	0.807	0.651
	1.12	98	1	1	0	0.995	0.497
	1.20	98	1	1	0	0.995	0.597
	1.28	98	1	0.783	1	0.995	0.895
	1.30	98	1	0.815	1	0.995	0.497
	1.32	98	1	1	0	0	0
	1.33	98	2	0.826	1	0.995	0.847
	1.4.1	98	2	1	0	0.504	0.302
	2.1	98	22	0.991	0.955	0.992	0.814
	2.2	98	13	0.925	0.948	0.99	0.784
	2.3	98	12	0.845	1	0.914	0.831
	2.4	98	1	1	0	0.995	0.398
	3.1	98	6	0.934	0.833	0.972	0.868
	3.2	98	1	0.787	1	0.995	0.895

Рис.3.5. Процес навчання нейронної мережі

Після завершення тренування моделі YOLOv8 на навчальному наборі даних, проводиться оцінка її результатів. Цей процес включає в себе прогон моделі по

валідаційному набору даних, який не використовувався під час тренування. Під час цієї оцінки перевіряється ефективність моделі на нових, раніше не бачених даних.

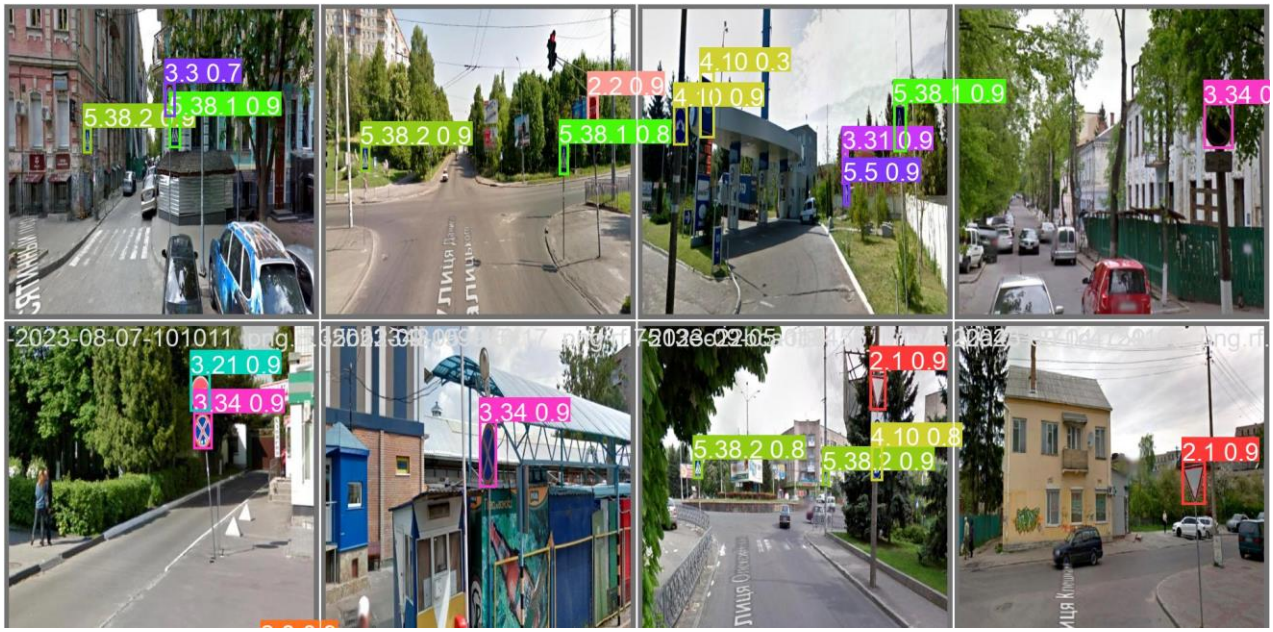

Рис.3.6. Проходження по валідаційному набору даних

Валідація допомагає виявити, наскільки добре модель узагальнює свої знання на нових зображеннях (рис.3.6). Результати валідації дають уявлення про точність моделі та її здатність розпізнавати дорожні знаки на різних умовах зйомки.

Цей етап дозволяє оцінити, чи досягнута моделлю необхідна точність у розпізнаванні дорожніх знаків та чи є необхідність у подальших вдосконаленнях чи оптимізаціях.

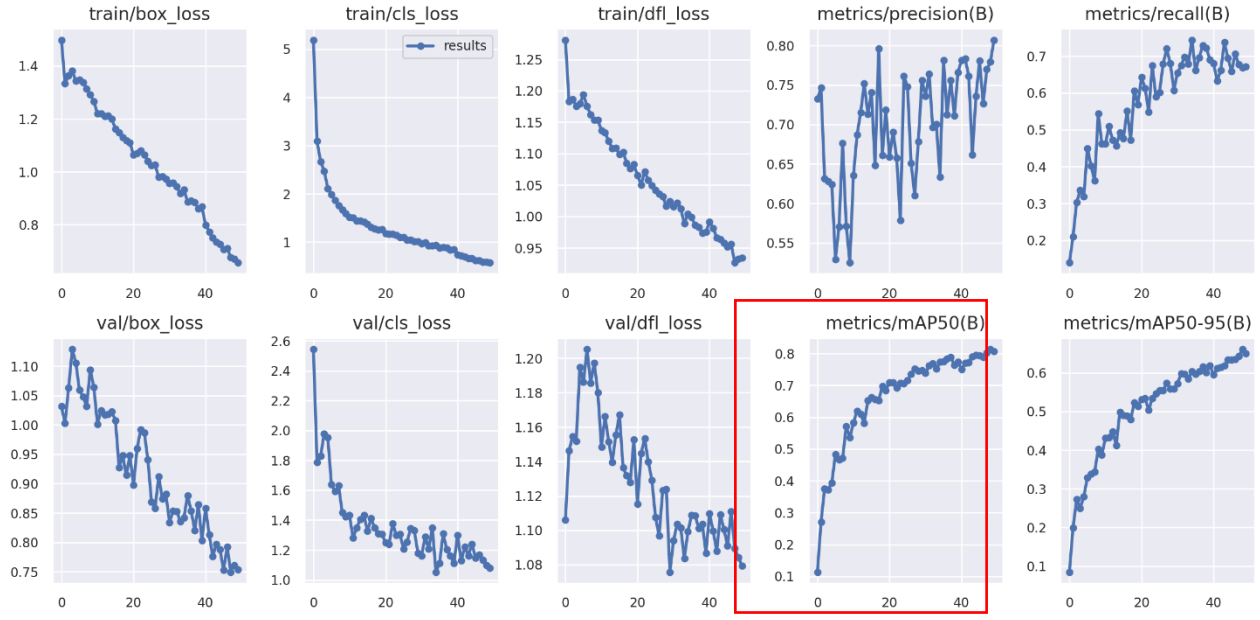

Рис.3.7. Метрика результатів навчання нейронної мережі

На рис 3.7 [29] видно результати навчання. Як видно, було досягнено mAP близько 82%, що є задовільним.

Після завершення процесу навчання моделі YOLOv8 формується набір навчених параметрів, які представлені у вигляді вагових коефіцієнтів. Ці ваги зберігаються у вказаній структурі каталогів (/runs/detect/train/weights/best.pt) у вашому проекті. Завантаження цих ваг до платформи Roboflow Deploy надає можливість використовувати навчені ваги на масштабованій інфраструктурі для реалізації завдань розпізнавання об'єктів [30].

Функціональність .deploy() у пакеті Roboflow pip дозволяє передавати ваги моделі YOLOv8 у режимі онлайн, забезпечуючи можливість ефективного використання цих ваг у дистанційних середовищах (рис.3.8).

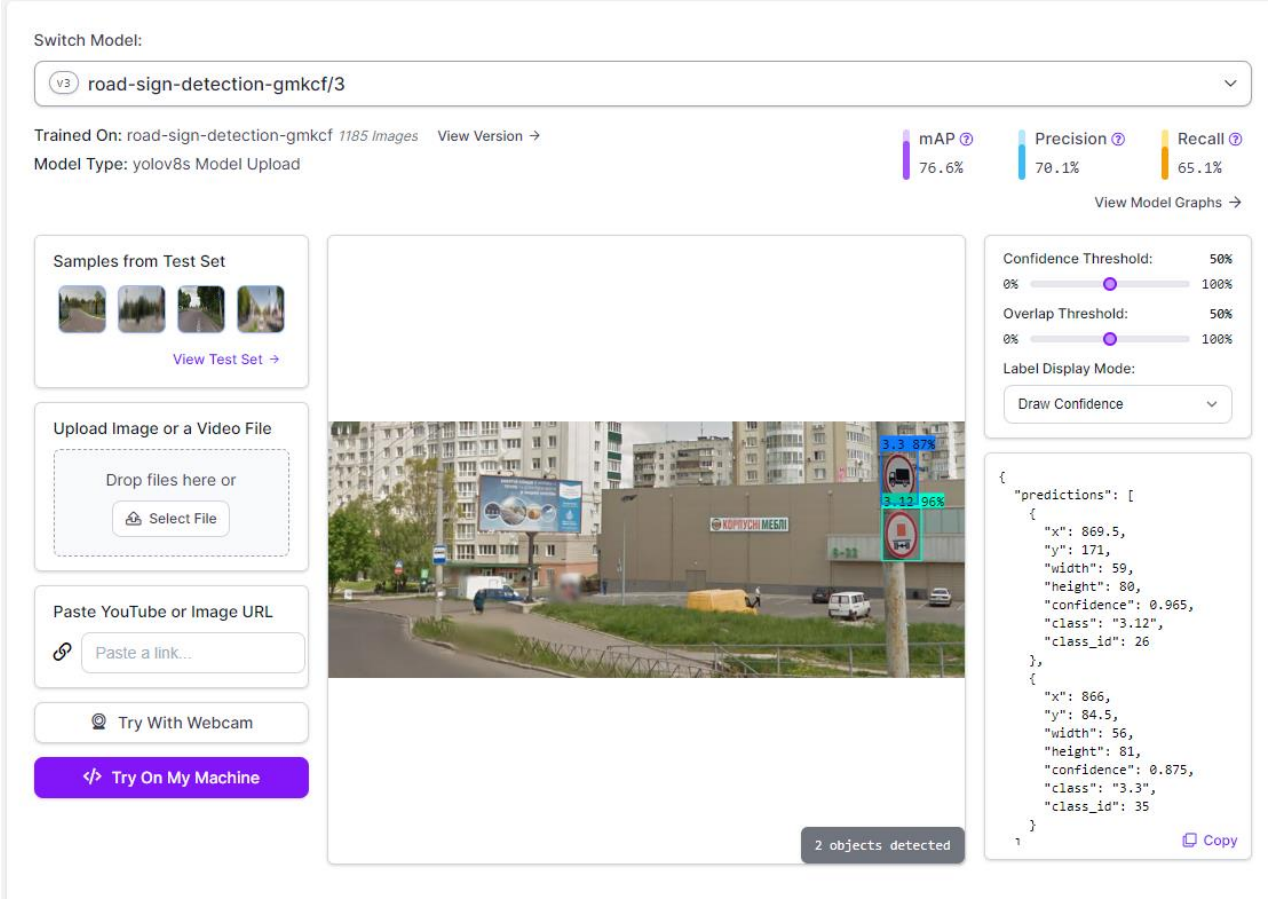

Рис.3.8. Модель розгорнута на середовищі Roboflow

3.3. Створення додатку

Для початку роботи над додатком на React Native встановлюються необхідні компоненти (Node.js і npm, React Native CLI). Створюємо новий проект і починаємо редагувати файл для додавання коду React Native, розширюююючи додаток.

Також потрібно враховувати кілька важливих моментів. Перш за все, варто знати, що React Native є кросплатформовим фреймворком, що дозволяє створювати мобільні додатки для обох платформ, проте деякі функції можуть працювати по-різному на Android та iOS, тому завжди переконуємось, що наш додаток функціонує коректно на обох платформах. По-друге, використання нативних компонентів може допомогти забезпечити більшу швидкість і кращу інтеграцію з операційною системою пристрою. По-третє, різні пристрої мають різні розміри та роздільні здатності екранів. Потрібно адаптувати інтерфейс

додатка до різних пристроїв. Ці аспекти допоможуть ефективно розробляти додатки на React Native, які працюють оптимально на обох платформах.

Для розпізнавання дорожніх знаків і відображення відповідної інформації на дорозі необхідно використовувати камеру. Інтеграція камери в React Native є досить простою і не викликає значних проблем. Єдине, що потрібно врахувати, це налаштування дозволів на використання камери.

Крім того, слід врахувати доступ до виведення звуку, оскільки буде використовуватися синтез мовлення. Для платформи Android ми використовуємо базовий компонент Expo, який забезпечує необхідний функціонал для синтезу мовлення. На платформі iOS необхідно використовувати нативні компоненти Swift для інтеграції синтезу мовлення, що може спричинити певні складнощі.

На iOS можна використовувати фреймворк AVFoundation для синтезу мовлення. Наприклад, можна створити клас Swift для синтезу мовлення, а потім інтегрувати цей клас у проєкт React Native за допомогою модуля Native Modules. Це вимагає додаткового налаштування та знання Swift, що може бути більш складним порівняно з використанням стандартних компонентів Expo на Android.

Таким чином, під час розробки додатків на React Native з використанням камери та синтезу мовлення необхідно враховувати специфіку налаштування дозволів і вибору відповідних компонентів для різних платформ.

В кінці-кінців треба реалізувати запит за допомогою асинхронної функції, яка використовується для захоплення зображення за допомогою камери, обробки цього зображення та надсилання його до зовнішнього API для розпізнавання дорожніх знаків. Функція перевіряє, чи дозволено використовувати камеру (cameraRef.current), і якщо так, захоплює зображення з базовим кодуванням у форматі base64. Отримані дані фотографії зберігаються у стані photoData, після чого відправляються POST-запитом до API сервісу Roboflow.

Запит включає базове кодування зображення у форматі base64 як тіло запиту та необхідний API-ключ для аутентифікації. Після отримання відповіді від API, в стан predictions зберігаються передбачення, отримані з сервісу розпізнавання. Ці передбачення містять інформацію про виявлені об'єкти на зображенні,

включаючи їхні класи та рівні впевненості. Після цього формується текстове повідомлення з описом усіх виявлених об'єктів, яке потім озвучується за допомогою функції speak, що забезпечує синтез мовлення для відтворення отриманих результатів. У разі виникнення помилок під час виконання запиту, вони логуються в консоль для подальшого аналізу.

Оптимізація додатка починається з аналізу існуючого коду для виявлення можливих місць для поліпшення продуктивності та зменшення часу відгуку. У нашому додатку слід звернути увагу на оптимізацію запитів до API, забезпечивши мінімальний розмір переданих даних, використовуючи параметр quality при захопленні зображення для зменшення розміру файлу без значної втрати якості, а також застосовувати ефективні алгоритми серіалізації даних та оптимізацію формату запиту. Важливо ефективно управляти станом, мінімізуючи кількість змін стану, що викликають повторний рендеринг компонентів, та використовуючи референції для збереження значень, що не впливають на рендеринг. Асинхронні операції повинні забезпечувати асинхронне виконання, щоб уникнути блокування головного потоку додатка.

Стилізація додатка є ключовою для покращення користувацького досвіду і досягнення привабливого інтерфейсу. Вона повинна включати гармонійний дизайн інтерфейсу, використання кольорової палітри для забезпечення доброї видимості тексту та елементів управління, а також зручне і інтуїтивно зрозуміле розташування елементів. Адаптивність інтерфейсу передбачає підтримку різних розмірів екранів і орієнтацій пристроїв, використання відносних одиниць вимірювання для розмірів елементів для забезпечення адаптивності дизайну. Покращення користувацького досвіду досягається за рахунок додавання анімацій та візуальних ефектів, а також забезпечення зворотного зв'язку для користувача при виконанні дій, наприклад, натискання кнопок або завантаження даних. Програмний код мобільного застосунку знаходиться в ДОДАТКУ Б.

Після оптимізації та стилізації необхідно провести всебічне тестування додатка для виявлення можливих помилок і забезпечення стабільної роботи. Інтеграційне тестування перевіряє взаємодію між різними компонентами додатка

та тестує процеси, які включають декілька етапів, наприклад, захоплення зображення та відправку його на сервер. Тестування продуктивності аналізує час відгуку додатка при виконанні різних дій та перевіряє використання ресурсів пристрою, таких як пам'ять і процесорний час. Необхідно також перевірити роботу додатка на різних платформах (iOS, Android) та пристроях з різними розмірами екранів, а також в різних умовах експлуатації, таких як різні рівні заряду батареї або відсутність інтернет-з'єднання. Після завершення тестування слід проаналізувати результати та внести відповідні виправлення для забезпечення максимальної надійності та зручності використання додатка.

3.4. Інтеграція зовнішніх периферійних пристроїв

Інтеграція зовнішніх периферійних пристроїв - це процес забезпечення взаємодії між програмним забезпеченням (наприклад, моделлю розпізнавання дорожніх знаків) та зовнішніми апаратними засобами, такими як камери, сенсори чи інші пристрої. Ця інтеграція може включати розробку апаратного та програмного забезпечення, щоб забезпечити взаємодію та обмін інформацією між ними.

У випадку розпізнавання дорожніх знаків, інтеграція може означати підключення камери або сенсорів до програми, яка використовує модель машинного навчання для аналізу вхідних зображень та розпізнавання дорожніх знаків на них. Це може вимагати розробки спеціалізованих інтерфейсів або протоколів комунікації для передачі даних між програмним та апаратним середовищами.

Важливо забезпечити сумісність між програмним та апаратним забезпеченням, налаштувати передачу даних та забезпечити стабільну роботу системи у реальному часі для успішної інтеграції зовнішніх пристроїв у роботу моделі розпізнавання дорожніх знаків.

Перед тим як розпочати, необхідно встановити Docker. Docker потрібен для запуску сервера Inference, який відповідає за керування інференсами (прогнозами моделі на нових даних) для комп'ютерного зору. Встановлення Docker

забезпечить необхідні засоби для ізоляції та управління середовищем, в якому працює сервер Inference.

Далі потрібно встановити Inference:

pip install inference

Після цього потрібно запустити сервер Inference. Цей сервер керує інференсами та розрахований на масштабування. Він може працювати на вашому локальному комп'ютері, віддаленому сервері або навіть на Raspberry Pi.

inference server start

Після запуску сервера Inference можна починати виконувати запити на виконання інференсів (прогнозів) за допомогою встановленого сервера. Це означає виконання різних запитів для роботи з комп'ютерним зором або використання навчених моделей для аналізу даних чи зображень, або об’єктів розпізнані на вебкамері.

Для налаштування підключення вебкамери та конфігурації файлів для використання в інференсі можна скористатися спеціальними бібліотеками чи інструментами комп'ютерного зору та обробки зображень, які дозволяють отримувати відеопотік з камери або читати зображення з файлів та обробляти їх на вхід для сервера Inference. Це включає імплементацію функцій, які забезпечують доступ до камери, захоплення кадрів та їх підготовку для подальшої передачі до сервера для виконання інференсу. Конфігурація файлів в цьому контексті може означати вказання шляхів до відео чи зображень, які будуть оброблятися. Для підключення вебкамери і роботи з файлами, потрібно виконати налаштування відповідних функцій для захоплення даних з камери або зчитування файлів, підготовки їх до подальшого використання у процесі інференсу на сервері.

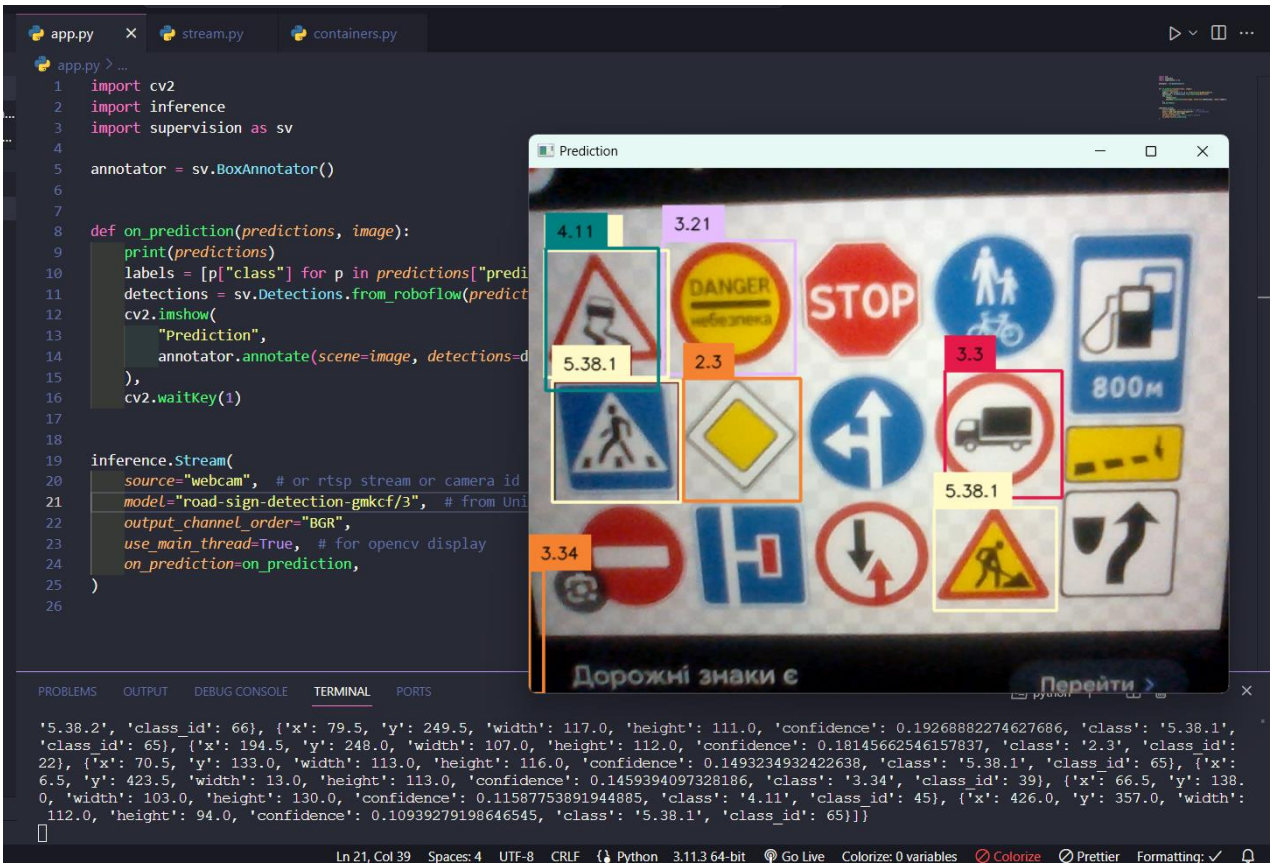

Рис.3.9. Тестування моделі нейронної мережі з використання веб-камери

Після отримання результатів від сервера Inference, дані передаються до додатка у вигляді масиву об'єктів predictions, що містить виявлені дорожні знаки. Кожен об'єкт у цьому масиві включає інформацію про тип виявленого знака, його координати на зображенні та рівень впевненості в розпізнаванні.

Отримані дані обробляються для формування інформативних речень. Кожне речення складається з опису виявленого знака та рівня впевненості в його розпізнаванні. Наприклад, для знака "Stop" з рівнем впевненості 0.95 формулюється речення "Виявлено знак Stop з впевненістю 95%". Ці речення формуються шляхом ітерації через масив predictions та об'єднання інформації з кожного об'єкта.

Для виведення інформації використовується технологія синтезу мовлення зазначена в розділі 3.3.

Таким чином, після отримання даних від сервера Inference, додаток формує зрозумілі для користувача речення, що описують виявлені дорожні знаки, та озвучує їх за допомогою технології синтезу мовлення. Це забезпечує ефективне

сприйняття інформації користувачем та підвищує функціональність додатка (рис. 3.10).

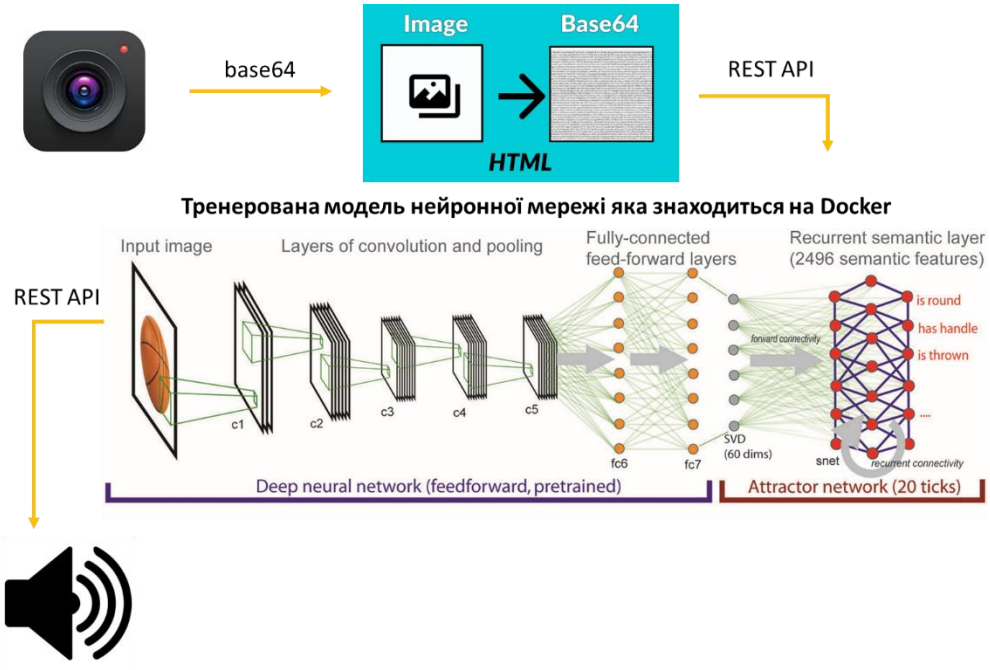

Рис. 3.10. Схематична модель потоку інформації

У майбутніх версіях додатка необхідно зосередитися на вдосконаленні інтерфейсу користувача з метою підвищення зручності та ефективності взаємодії з системою. Планується розробка більш інтуїтивного та естетично привабливого інтерфейсу, який враховуватиме потреби користувачів. Це включає вдосконалення елементів управління, забезпечення гармонійного поєднання кольорів і додавання анімацій для покращення користувацького досвіду.

Важливим аспектом є додання тимчасової бази даних, що дозволить зберігати результати розпізнавання та інші важливі дані локально на пристрої. Це покращить швидкість доступу до інформації та зменшить залежність від постійного інтернет-з'єднання. Тимчасова база даних забезпечить ефективне зберігання та обробку даних у режимі офлайн, що особливо важливо для мобільних додатків.

Впровадження української мови в синтез мовлення та локалізацію додатка є ще одним важливим покращенням. Це забезпечить ширший доступ до додатка для україномовних користувачів, підвищуючи його зручність. Локалізація інтерфейсу додатка на українську мову включає переклад усіх текстових

елементів, налаштування відповідних форматів дати і часу, а також інтеграцію української мови для функцій синтезу мовлення. Це дозволить озвучувати результати розпізнавання та інші повідомлення українською мовою.

Отже, ці покращення розширять функціональні можливості додатка, зроблять його доступним для ширшої аудиторії та підвищать ефективність роботи системи розпізнавання дорожніх знаків.

ВИСНОВКИ

У процесі виконання досліджень вивчались принципи побудови та реалізації штучних нейронних мереж для розпізнавання дорожніх знаків на зображеннях. Встановлено, що зазначену проблему можна вирішити на основі використання комплексу програмних засобів YOLOv8, React Native із залученням відповідних хмарних технологій.

Результатом роботи є кросплатформений програмний додаток, який розроблявся в середовищі React Native. Додаток використовує результати роботи штучної нейронної мережі YOLOv8. При цьому частина необхідної для роботи системи інформації розміщувалося на сервері Inference (середовище Docker). Реалізована інтеграція пристроїв введення-виведення з програмним додатком та конфігуровано відповідні файли зображень для їх подальшого використання у системі YOLOv8.

Проведено серію експериментів. Показано, що запропонований підхід до конфігурації програмної системи дозволяє вирішувати відповідні задачі розпізнавання з прийнятною швидкістю та точністю.

Результати проведених експериментів свідчать про перспективність обраного напрямку досліджень. У подальшому доцільно проводити роботу за наступними напрямками:

- удосконалення датасету (добавлення даних до датасету, оптимізація датасету);
- продовження процесу навчання на більшій кількості епох;
- розроблення алгоритму розпізнавання в реальному часі;
- розширення функціоналу нейронної мережі та програмної системи уцілому.

Використання штучних нейронних мереж і супутніх технологій дозволяє розширити можливості впровадження систем машинного зору в різноманітних застосуваннях, зокрема, для підвищення безпеки дорожнього руху.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ackerman, S. (1992). Discovering the brain. National Academies Press. URL: https://www.ncbi.nlm.nih.gov/books/NBK234146/ (дата звернення: 12.10.2023).

2. Добровська Л. М. Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – К.: НТУУ «КПІ» Вид-во «Політехніка», 2015. – 396 с.

3. Hardesty L (14 April 2017). "Explained: Neural networks". MIT News Office. Retrieved 2 June 2022. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (дата звернення: 12.10.2023).

4. Artificial neural networks Evergreen. URL: https://evergreens.com.ua/ua/development-services/neural-network.html https://evergreens.com.ua/ua/articles/object-recognition-case.html/ (дата звернення: 12.10.2023).

5. W.S. McCulloch and W. Pits. A Logical Calculus of the Ideas Immanent in Nervous Activity. Bull. Math. Biophys., Vol. 5, 1943, pp. 115-133.

6. Khademi F., Jamal S. M. Predicting the 28 Days Compressive Strength of / F. Khademi, S. M. Jamal. – I-Manager's Journal on Civil Engineering, 2016. – Vol. 6 (August). – pp. 1–7.

7. Хайкин Саймон. Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation / С. Хайкин. – 2-е изд. – М.: «Вильямс», 2006. – 1104 с. – ISBN 0-13-273350-1.

8. What are Recurrent Neural Networks (RNN)? URL: https://botpenguin.com/glossary/recurrent-neural-network (дата звернення: 22.05.2024)

9. Illustrated Guide to LSTM's and GRU's: A step by step explanation URL: https://towardsdatascience. com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (дата звернення: 22.05.2024)

10. Generative adversarial networks (GANs) : a deep dive into the architecture and training process URL: https://www.leewayhertz.com/generative-adversarial-networks/ (дата звернення: 22.05.2024)

11. LeCun Y., Bengio Y., Hinton G. Deep learning / Y. LeCun, Y. Bengio, G. Hinton. – Nature, 2015. – Vol. 521, No. 7553. – pp. 436-444..

12. VGG16 – Convolutional Network for Classification and Detection URL: https://neurohive.io/en/popular-networks/vgg16/ (дата звернення: 22.05.2024)

13. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. E. Hinton. – Communications of the ACM, 2017. – Vol. 60, No. 6. – pp. 84-90..

14. Ultralytics YOLOv8 Docs: URL: https://docs.ultralytics.com/ (дата звернення: 12.10.2023)

15. Introducing Instance Segmentation in YOLOv5 v7.0 URL: https://www.ultralytics.com/blog/introducing-instance-segmentation-in-yolov5-v7-0 (дата звернення: 22.05.2024)

16. YOLO-Based Pose Estimation: Bridging Speed and Accuracy URL: https://medium.com/@umamahesvari10/yolo-based-pose-estimation-bridging-speed-and-accuracy-e84b211fca6c (дата звернення: 22.05.2024)

17. Розробка мобільних додатків від А до Я: повний гайд URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/ (дата звернення: 22.05.2024)

18. "Application Fundamentals". Android Developers. URL: https://developer.android.com/guide/components/fundamentals/ (дата звернення: 22.05.2024)

19. B Block R. Live from Apple's iPhone SDK press conference / R. Block. – Engadget – AOL, March 6, 2008. – URL: www.Engadget.com (дата звернення: 22.05.2024).

20. Krill P. Sun: we'll put Java on the iPhone / P. Krill. – InfoWorld – International Data Group, March 7, 2008. – URL: www.InfoWorld.com (дата звернення: 22.05.2024).

21. Paul R. MonoTouch drops .NET into Apple's walled app garden / R. Paul. – Ars Technica – Condé Nast, September 15, 2009. – URL: ArsTechnica.com (дата звернення: 22.05.2024).

22. Dove J. Adobe unleashes Creative Suite 5 / J. Dove. – Macworld – International Data Group, April 11, 2010. – ULR: www.MacWorld.com (дата звернення: 22.05.2024).

23. Sorrell C. Apple eases app development rules, Adobe surges / C. Sorrell. – Wired – Condé Nast, September 9, 2010. – URL: www.Wired.com (дата звернення: 22.05.2024).

24. What is data labeling? URL: https://aws.amazon.com/what-is/data-labeling/ (дата звернення: 12.10.2023).

25. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon, S. Divvala, R. Girshick, A. Farhadi. – 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. – doi: 10.1109/CVPR.2016.91.

26. Bush E. JavaScript Applications with Node.js, React, React Native and MongoDB / E. Bush. – 2018. – 392 p.

27. Lebensold J. React Native Cookbook: Bringing the Web to Native Platforms / J. Lebensold. – 2018. – 176 p.

28. What is data labeling? URL: https://aws.amazon.com/what-is/data-labeling/ (дата звернення: 12.10.2023)

29. Метрика результатів URL: https://app.roboflow.com/rivne-state-humanitarian-university-faculty-of-mathematics-and-information-science/road-sign-detection-gmkcf/3/train/results (дата звернення: 12.10.2023).

30. Road Sign Detection Computer Vision Project URL: https://universe.roboflow.com/rivne-state-humanitarian-university-faculty-of-mathematics-and-information-science/road-sign-detection-gmkcf (дата звернення: 12.10.2023).

31. Посилання на гітхаб на програмний код URL: https://github.com/Virtuozio/road-sign-recognititon (дата звернення: 12.10.2023).

ДОДАТКИ

ДОДАТОК А

Використання потоку даних для відео або наживо

```
# import the InferencePipeline interface
from inference import InferencePipeline

# import a built-in sink called render_boxes (sinks are the logic that happens after inference)
from inference.core.interfaces.stream.sinks import render_boxes

api_key = "YOUR_ROBOFLOW_API_KEY"

# create an inference pipeline object
pipeline = InferencePipeline.init(
    model_id="yolov8x-1280",  # set the model id to a yolov8x model with in put size 1280
    video_reference="https://storage.googleapis.com/com-roboflow-marketing/inference/people-walking.mp4",  # set the video reference (source of video), it can be a link/path to a video file, an RTSP stream url, or an integer representing a device id (usually 0 for built in webcams)
    on_prediction=render_boxes,  # tell the pipeline object what to do with each set of inference by passing a function
    api_key=api_key,  # provide your roboflow api key for loading models from the roboflow api
)
# start the pipeline
pipeline.start()
# wait for the pipeline to finish
pipeline.join()
```

ДОДАТОК Б

Програмний код кросплатформеного застосунку

```
import React, { useState, useRef } from "react";
import {
  Button,
  StyleSheet,
  Text,
  View,
  Image,
  Pressable,
  Dimensions,
  Platform,
} from "react-native";
import { Camera, CameraType } from "expo-camera";
import { Ionicons } from "@expo/vector-icons";
import axios from "axios";
import * as Speech from "expo-speech";
import { NativeModules } from "react-native";

const { SpeechSynthesizer } = NativeModules;

const speak = (text) => {
  SpeechSynthesizer.speak(text)
    .then((response) => {
      console.log("Speech started");
    })
    .catch((error) => {
      console.error("Error:", error);
    });
};

export default function App() {
  const [type, setType] = useState(CameraType.back);
  const [permission, requestPermission] = Camera.useCameraPermissions();
  const [photoData, setPhotoData] = useState(null);
```

```
const [predictions, setPredictions] = useState([]);
const cameraRef = useRef(null);

const speak = (text) => {
  if (Platform.OS === "web") {
    // Використовуємо Web Speech API для вебу
    const synth = window.speechSynthesis;
    const utterance = new SpeechSynthesisUtterance(text);
    utterance.lang = "en-GB"; // Встановлення української мови
    console.log(utterance);
    synth.speak(utterance);
  } else if (Platform.OS === "android") {
    // Використовуємо expo-speech на Android
    Speech.speak(text);
  } else if (Platform.OS === "ios") {
    // Використовуємо нативний модуль на iOS
    SpeechSynthesizer.speak(text)
      .then((response) => {
        console.log("Speech started on iOS");
      })
      .catch((error) => {
        console.error("Error on iOS:", error);
      });
  }
};

const generateRandomColor = () => {
  return "#" + Math.floor(Math.random() * 16777215).toString(16);
};

if (!permission) {
  return <View />;
}

if (!permission.granted) {
```

```
    return (
      <View style={styles.container}>
        <Text style={{ textAlign: "center" }}>We need your permission to show the
camera</Text>
        <Button onPress={requestPermission} title="Grant permission" />
      </View>
    );
  }

  async function takePicture() {
    if (cameraRef.current) {
      const options = { quality: 0.5, base64: true, skipProcessing: true };
      const data = await cameraRef.current.takePictureAsync(options);
      console.log(`${data.width} - width, ${data.height} - height`);
      setPhotoData(data);

      axios({
        method: "POST",
        url: "https://detect.roboflow.com/road-sign-detection-gmkcf/2",
        params: {
          api_key: "Ir5Sd1dhWA5tkwMauDL6",
        },
        data: data.base64,
        headers: {
          "Content-Type": "application/x-www-form-urlencoded",
        },
      })
        .then(function (response) {
          setPredictions(response.data.predictions);
          console.log(response.data.predictions);

          // Складаємо повідомлення із всіх виявлених
об'єктів
          const message = response.data.predictions
```

```
            .map((pred) => `Detected ${pred.class} with confidence
${pred.confidence.toFixed(2)}`)
            .join(", ");

          // Використовуємо функцію speak для відтворення
повідомлення
          speak(message);
        })
        .catch(function (error) {
          console.log(error.response);
        });
    }
  }

  function toggleCameraType() {
    setType((current) => (current === CameraType.back ? CameraType.front : CameraType.back));
  }

  return (
    <View style={styles.container}>
      {photoData ? (
        <>
          {predictions.map((pred, index) => {
            const borderColor = generateRandomColor();
            return (
              <View
                key={index}
                style={{
                  position: "absolute",
                  top: `${(pred.y * 100) / photoData.height}%`,
                  left: `${(pred.x * 100) / photoData.width}%`,
                  width: `${(pred.width * 100) / photoData.width}%`,
                  height: `${(pred.height * 100) / photoData.height}%`,
                  borderColor,
                  borderWidth: 2,
```

```
                transform: [
                  { translateX: -Dimensions.get("window").width * 0.05 },
                  { translateY: -Dimensions.get("window").height * 0.1 },
                ],
              }}
            >
              <Text style={[styles.classLabel, { backgroundColor: borderColor }]}>
                {pred.class}
              </Text>
            </View>
          );
        })}
      </>
    ) : (
      <Camera style={styles.camera} type={type} ref={cameraRef}>
        <Pressable style={styles.toggleButton} onPress={toggleCameraType}>
          <Ionicons name="camera-reverse-outline" size={48} color="white" />
        </Pressable>
        <View style={styles.centeredFlex}>
          <Pressable style={styles.captureButton} onPress={takePicture}>
            <View style={styles.innerCaptureButton} />
          </Pressable>
        </View>
      </Camera>
    )}
  </View>
 );
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   justifyContent: "center",
   border: 2,
   borderRadius: 50,
```

```
  },
  camera: {
    flex: 1,
  },
  centeredFlex: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    position: "absolute",
    left: 0,
    right: 0,
    bottom: 35,
  },
  toggleButton: {
    position: "absolute",
    left: 20,
    bottom: 20,
    width: 48,
    height: 48,
  },
  captureButton: {
    width: 70,
    height: 70,
    backgroundColor: "white",
    borderRadius: 35,
    justifyContent: "center",
    alignItems: "center",
  },
  innerCaptureButton: {
    width: 60,
    height: 60,
    backgroundColor: "red",
    borderRadius: 30,
  },
  classLabel: {
```

```
    color: "white",
    fontWeight: "bold",
    padding: 2,
    fontSize: 12,
  },
});
```