

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра прикладної математики

## Дипломна робота

Розробка мобільного додатку для класифікації та розпізнавання об'єктів з використанням штучних нейронних мереж

Виконав: студент групи ПМП-42  
спеціальності  
113 - прикладна математика

Чернецький С. В.

(прізвище та ініціали)

Керівник Заневич О. Б.

(прізвище та ініціали)

Рецензент к.ф.-м.н Депутат Б. Я.

(прізвище та ініціали)

# ЗМІСТ

<b>Перелік умовних позначень, скорочень, термінів, одиниць</b>	<b>4</b>
<b>Вступ</b>	<b>6</b>
<b>Постановка задачі</b>	<b>9</b>
<b>1 Розпізнавання об'єктів</b>	<b>10</b>
1.1 OpenCV	10
1.2 TensorFlow mobile	11
1.3 TensorFlow lite	12
1.4 Snapdragon Neural Processing Engine	13
1.5 Емпіричні критерії оцінки	14
1.5.1 Попередня обробка	14
1.5.2 Розмір моделі	15
1.5.3 Час завантаження	15
1.5.4 Точність	16
1.5.5 Швидкість	16
1.5.6 Середнє використання оперативної пам'яті	17
1.5.7 Максимальне використання оперативної пам'яті	17
1.5.8 Акумулятор	17
1.5.9 Температура	18
1.5.10 Трекер	18
<b>2 Мобільний додаток</b>	<b>19</b>
2.1 Що таке нативна розробка мобільного додатка	19
2.2 Плюси розробки нативних додатків	19
2.3 Мінуси розробки нативних додатків	20
2.4 Що таке кросплатформна розробка	21
2.5 Плюси кросплатформної розробки додатків	21

2.6	Мінуси кросплатформної розробки додатків	22
2.7	Flutter	22
2.8	React Native	23
<b>3</b>	<b>Програмна реалізація</b>	<b>24</b>
	<b>Висновок</b>	<b>30</b>
	<b>Список використаної літератури</b>	<b>31</b>

# **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ, ОДИНИЦЬ**

Фреймворк – (framework) інфраструктура програмних рішень, що полегшує розробку складних систем.

Комп'ютерний зір – теорія та технологія створення машин, які можуть проводити виявлення, стеження та визначення об'єктів.

Штучний інтелект - розділ комп'ютерної лінгвістики та інформатики, що опікується формалізацією проблем та завдань, які подібні до дій, що виконує людина.

Convolutional neural network - (CNN) згорткова нейромережа.

TFM - TensorFlow mobile.

TFL - TensorFlow lite.

YOLO - You Only Look Once.

ROI - Region of interest.

SSD - Single Short Detector.

R-CNN – (Regions With CNNs) Згорткова мережа на основі вибору регіонів.

Selective Search – алгоритм для пошуку регіонів.

Нейромережа – сукупність з'єднаних штучних нейронів.

Нейрон – вузол нейромережі.

LSTM – (Long short-term memory) довга короткочасна пам'ять.

CPU - (Central processing unit) центральний процесор (ЦП).

GPU - (Graphics processing unit) графічний процесор.

DSP - (Digital signal processor) це спеціалізований програмований мікропроцесор, призначений для маніпулювання в реальному масштабі часу потоком цифрових даних.

Snapdragon Neural Processing Engine SDK (SNPE) — це програмне забезпечення Qualcomm Snapdragon.

DLC - Deep Learning Container.

YUV - це колірна модель, яка зазвичай використовується як частина конвеєра кольорових зображень. Він кодує кольорове зображення або відео, враховуючи людське сприйняття, дозволяючи зменшити пропускну здатність для компонентів кольоровості в порівнянні з «прямим» відображенням RGB.

## Вступ

Штучний інтелект набрав обертів останніми роками у світлі величезного потенціалу в широкому діапазоні додатків, і існує тенденція до використання машинного навчання в легких, вбудованих системах, таких як смартфони для високої мобільності, низької вартості, швидкого розгортання та інших переваг. Можливості згорткової нейронної мережі (CNN) на основі смартфона є технологією, яка забезпечує різноманітні нові випадки використання з підтримкою машинного навчання, коли розпізнавання об'єктів потрібне на відкритому повітрі, де існують інші обмежуючі фактори, такі як потреба у свободі пересування для користувача або обмежена інфраструктура та покриття, доступні в географічній зоні. Обчислювальна потужність смартфонів різко зросла за останні кілька років, і тепер їх можна порівняти з комп'ютерами, доступними кілька років тому. Тим не менш, запуск таких моделей CNN на мобільних пристроях все ще є складним завданням через обмежену обчислювальну потужність та доступну енергію. Традиційно моделі CNN працюють на високопродуктивних комп'ютерних серверах через вимоги до обладнання та недоступні для роботи на смартфонах. Щоб подолати ці проблеми, існує життєво важлива потреба в платформі машинного навчання, яка підходить для смартфонів, щоб виконувати інтенсивні обчислювальні задачі комп'ютерного зору, такі як розпізнавання об'єктів.

Через появу інтересу до операційної системи Android деякі популярні фреймворки глибокого навчання були перенесені на цю операційну систему, включаючи TensorFlow Mobile (TFM), TensorFlow lite (TFL), OpenCV і Qualcomm Snapdragon. Ці платформи призначені для виконання завдання висновку на мобільних телефонах, що підходять для сценаріїв, де

підключення погане або відсутнє. Кожен з цих популярних фреймворків має свої сильні та слабкі сторони, які слід враховувати при виборі найкращої архітектури для розпізнавання об'єктів. Однак це відкрите питання, яке ще не вирішується практично. З огляду на дуже складні та гетерогенні моделі в цих структурах, найбільш практичним підходом буде емпірична методологія тестування та оцінки. Система розпізнавання об'єктів, як правило, складається з трьох кроків, а саме: попередня обробка зображень, розпізнавання зображень і відстеження зображення. Для кожного кроку необхідний вибір найкращої мобільної платформи на основі оцінки, щоб отримати найкращу продуктивність системи. Платформу зазвичай оцінюють за такими критеріями, як точність, швидкість, використання оперативної пам'яті та розмір моделі. Розпізнавання зображень – це дорогий обчислювальний процес, здатний позначити об'єкт зображення за допомогою обмежувальної рамки. Однак, хоча відстеження об'єкта набагато менш затратне в обчисленні, воно потребує початкових координат об'єкта, щоб відстежувати його рух у наступних кадрах. Таким чином, поєднання розпізнавання зображень і відстеження зображень може створити конкурентоспроможну систему за рахунок скорочення обчислень.

Детектори об'єктів на основі CNN поділяються на дві основні категорії: одноступінчасті та двоступеневі детектори. Двоступінчасті детектори дорогі в обчисленні, оскільки спочатку виділяються регіони інтересу (ROI), а потім виконується класифікація. Цей метод не підходить для розпізнавання об'єктів у реальному часі на пристроях з обмеженнями потужності. Тим часом одноступінчасті детектори, такі як You Only Look Once (YOLO) і Single Shot Detector (SSD), досягають розпізнавання об'єктів у реальному часі шляхом одночасного вибору та класифікації ROI. Ці методи швидші за ціною меншої точності. YOLOv3 є швидшим і точнішим, ніж інші детектори на основі YOLO, і тому досліджується в цьому дослідженні, по-перше, для аналізу

популярних платформ машинного навчання, а по-друге, для всебічного тестування запропонованої архітектури. Отже, основні внески цієї статті підсумовані таким чином:

- 1) Практичне розгортання та тестування популярних фреймворків машинного навчання, комплексна оцінка та аналіз популярних мобільних платформ на основі критеріїв, що впливають на продуктивність системи, а також внесок у емпіричну методологію у виборі найбільш підходящих технологій у задачах розпізнавання об'єктів на основі машинного навчання.
- 2) Розробка та впровадження нової архітектури розпізнавання об'єктів на основі емпіричного аналізу популярних фреймворків машинного навчання, що вивчаються, в операційному середовищі на базі смартфонів, що призвело до високо оптимізованої системи практичного використання.
- 3) Новий тест машинного навчання для систем розпізнавання об'єктів на платформі смартфона, що надає нове уявлення про продуктивність таких систем.
- 4) Обговорення обмежень запропонованої архітектури щодо запуску систем розпізнавання об'єктів на портативних пристроях.

## Постановка задачі

Завдання полягає у написанні згорткової нейронної мережі, яка могла б достатньо точно розпізнавати об'єкти на зображенні чи відео, а також правильно їх класифікувати на смартфоні.

Давайте для початку поговоримо що розуміє під собою ідентифікація об'єкта. Перед нами зображення з камери телефона, нам не складно вказати де знаходяться ті чи інші об'єкти.

Системі необхідно знайти об'єкти прийняти певне рішення.

В наш час існує велика кількість архітектур нейромерж, як от OpenCV, TensorFlow mobile, TensorFlow lite, Snapdragon Neural Processing Engine, розглянемо кожну з них і порівняємо.

Також для задачі розглядалася як нативна так і кросплатформна розробка мобільного додатка.

## Розділ 1

# Розпізнавання об'єктів

Як зазначалося, алгоритми розпізнавання об'єктів, засновані на CNN, можна розділити на дві основні категорії: двоступеневі та одноступінчасті детектори. У двоступеневих детекторах, таких як Fast R-CNN, Faster R-CNN і R-FCN, пропозиція області генерується на першому етапі. На другому етапі буде проведена класифікація об'єктів та регресія bounding-box (bbox). Ці методи мають високу точність, але повільні в розпізнаванні.

В одноступеневих детекторах, таких як SSD і YOLO, класифікація об'єктів і регресія bbox виконуються одночасно без етапу пропозиції регіону. Ці методи швидко розпізнаються, але мають низьку точність. YOLOv3 має високу швидкість розпізнавання кількох об'єктів в одному висновку. Крім того, низька точність YOLO та YOLOv2 вирішується за допомогою багатоетапного методу розпізнавання. YOLOv3 використовується, коли система має достатньо обчислювальних ресурсів. Tiny-YOLOv3 призначений для обмежених середовищ і менш точний, ніж YOLOv3.

### 1.1 OpenCV

Open Source Computer Vision Library (OpenCV) — це фреймворк з відкритим вихідним кодом для аналізу зображень і відео. Хоча ця структура була розгорнута, щоб зосередитися на обробці зображень, вона також надає функції для машинного навчання в реальному часі. OpenCV має можливість читати конфігурацію нейронної мережі та файл ваги безпосередньо без необхідності в конвертері для завантаження моделі CNN у пам'ять у

відповідному форматі. Між тим OpenCV обмежений в плані сумісності з процесорами смартфонів. Платформа просто здатна запускати моделі нейронної мережі за допомогою центрального процесора (CPU), упускаючи можливість скористатися перевагами високої обчислювальної потужності графічного процесора (GPU) і процесора цифрових сигналів (DSP).

Завдяки тому, що OpenCV є дуже зрілим і потужним, розробники добре впроваджують і застосовують у всьому світі. Його основна увага приділяється обробці зображень у реальному часі, що робить його ефективним і швидким для попередньої обробки зображень у процесі розпізнавання.

## **1.2 TensorFlow mobile**

TensorFlow Mobile — це платформа машинного навчання, розроблена Google. TFM має на меті підтримувати різні сучасні обчислювальні пристрої від смартфонів до вбудованих пристроїв, таких як Raspberry Pi та мобільні додатки. TFM було отримано від TensorFlow. Майже всі операції в стандартному TensorFlow підтримуються бібліотекою TFM. З огляду на те, що його загальна реалізація може бути застосована в широкому діапазоні архітектур, TFM має недолік у продуктивності, коли мова йде про обмежені середовища. Це обмеження пов'язане не тільки з обчислювальною потужністю та пам'яттю, але й із споживанням енергії. Ця платформа зараз застаріла і замінена на TFL, яка в основному зосереджена на системах з низькими обчислювальними ресурсами. Тим не менш, TFM все ще використовується в програмах, які не були оновлені.

На відміну від OpenCV, TFM потребує конвертера для перетворення файлу конфігурації нейронної мережі та його ваг у файл protobuf (.pb), який повністю сумісний з цією бібліотекою. Використовуючи файл protobuf,

TensorFlow може застосовувати різні методи оптимізації, такі як квантування, для подальшого підвищення швидкості виконання. Хоча цей тип оптимізації моделі може бути корисним для обмежених середовищ, він може поставити під загрозу точність. Подібно до OpenCV, TFM просто запускає навчену модель на ЦП. Це збільшує переносимість між процесорами. Однак це призводить до того, що платформу не можна легко розгорнути в обмежених середовищах.

### **1.3 TensorFlow lite**

TensorFlow Lite — це полегшена версія TFM для обмежених середовищ, що зменшує затримку та підвищує ефективність. Крім того, випущена Google, TFL — це фреймворк машинного навчання для запуску моделей TensorFlow на смартфонах, вбудованих системах та пристроях Інтернету речей (IoT) з низькою затримкою та малим розміром акумулятора. Порівняно з TFM, TFL потребує конвертера, щоб змінити файл конфігурації нейронної мережі та його ваги в модель плоского буфера (.tflite).

Тим часом TFL є більш універсальним, ніж TFM, оскільки він може виконувати алгоритми машинного навчання на всіх вбудованих процесорах. Ідентичний TFM і OpenCV, TensorFlow Lite може виконувати нейронну мережу на ЦП у простий спосіб, хоча TFL дає швидший час висновку.

Крім того, TFL має можливість делегувати частину або все виконання алгоритму мобільному графічному процесору. Це делегування здійснюється за допомогою двох ключових методів: делегування GPU і API нейронної мережі Android (NNAPI). У техніці делегування GPU TFL визначає рівень абстракції, щоб програміст міг спілкуватися з GPU. Це делегування допомагає досягти кращої продуктивності з точки зору часу висновку та споживання ресурсів. Хоча, оскільки вони сумісні зі смартфонами IOS і

багатьма пристроями Android, деякі моделі та операції не підтримуються. Це обмежує можливості розробників виконувати різні типи нейронних мереж на цій платформі. Наприклад, операції YOLOv3 не підтримуються на цій платформі. Однак операції для моделей Inception і Mobilenet підтримуються та використовуються в навчальних посібниках TFL.

Як уже було сказано, інший метод виконання CNN на GPU - це NNAPI. Для TFL реалізовано інтерфейс для зв'язку з API нейронної мережі Android. NNAPI має намір використовувати апаратні прискорювачі для фреймворків машинного навчання. Щоб зрозуміти це, постачальники повинні надати драйвери для своїх власних процесорів, щоб використовувати API і зробити його сумісним з процесорами на пристрої, включаючи CPU, GPU і DSP. Після цього NNAPI може застосовувати операції нейронної мережі з TFL до обладнання мобільних пристроїв. Тим не менш, подібно до техніки делегування GPU, існують обмеження під час виконання операцій YOLOv3.

## **1.4 Snapdragon Neural Processing Engine**

Snapdragon Neural Processing Engine SDK (SNPE) — це програмне забезпечення Qualcomm Snapdragon, прискорене під час виконання для запуску глибоких нейронних мереж. Наша система має інтегрований процесор Qualcomm Snapdragon для сумісності з SNPE. SNPE потребує моделі, відформатованої у файл Deep Learning Container (DLC), тому використовується перетворювач. Цей перетворювач, наданий Qualcomm, вимагає в якості входу моделі protobuf; таким чином, системі також потрібен той самий перетворювач, що й TFM.

Ця платформа повністю сумісна з власними драйверами, що забезпечують можливість безпосереднього керування вбудованими процесорами (CPU, GPU і DSP). Це призведе до підвищення ефективності.

Існує два основних недоліки розгортання цього SDK. По-перше, на відміну від попередніх платформ, сумісність знижена через те, що не всі смартфони оснащені Qualcomm Snapdragon. По-друге, розгортання SNPE є більш складним через низькорівневі функції, з якими мають працювати розробники, що змушує їх глибше розуміти апаратне забезпечення.

## **1.5 Емпіричні критерії оцінки**

У цьому підрозділі проводиться повна оцінка, щоб вибрати найкращі фреймворки для проектування нашої системи. У системі розпізнавання об'єктів використовуються три кроки, включаючи попередню обробку зображень, розпізнавання зображень і відстеження зображень. Щоб досягти найкращої продуктивності системи розпізнавання об'єктів, найкращу платформу слід вибрати для кожного вищезгаданого кроку на основі показників, що впливають на продуктивність системи. З цією метою в цьому розділі наведено покроковий підхід до вибору найкращої платформи для кожного кроку та отримання найефективнішої та швидкої архітектури розпізнавання машинного навчання після розгортання та впровадження кожної платформи в одному середовищі.

### **1.5.1 Попередня обробка**

Попередня обробка є фундаментальним кроком під час роботи конвеєра розпізнавання. Він перетворює необроблені кадри (YUV) у відповідний формат для введення нейронної мережі. Цей крок включає масштабування зображення, нормалізацію зображення та перетворення колірного простору. Масштабування зображення необхідне для зменшення розміру вхідного сигналу з  $1280 \times 720$  до  $416 \times 416$ , щоб підтримувати компроміс між швидкістю і точністю і досягати виявлення в режимі реального часу для

критичних додатків. Подібно до авторів YOLO, в цьому рукописі ми вибрали вхідний розмір  $416 \times 416$ , щоб забезпечити легше порівняння результатів з платформами інших дослідників. Нормалізація зображення – це ще один етап для масштабування значень пікселів від 0 до 255 до діапазону 0–1. Нарешті, в деяких випадках потрібна трансформація колірного простору. OpenCV завантажує фрейм у BGR. Отже, потрібне перетворення колірного простору в RGB. TFL і SNPE не забезпечують цей крок, тому покладаються на бібліотеки Android і тривають 56,3 мс. OpenCV, однак, дає чудові результати, оскільки він потужний і зрілий в обробці зображень із 9,3 мс.

### **1.5.2 Розмір моделі**

Розмір моделей призводить до використання пам'яті. Підходи Snapdragon і TFL використовують серіалізацію для перетворення файлу конфігурації та файлу ваги CNN у формат, який можна читати. Це призводить до кращого результату з точки зору розміру моделі. У нашому дослідженні формат моделі плоского буфера підходить для TensorFlow Lite. Тим часом подвійне перетворення з Protobuf в DLC в SNPE зменшує розмір моделі.

### **1.5.3 Час завантаження**

Менші моделі не обов'язково означають, що модель швидше завантажуватиметься в пам'ять. Найнижча модель для завантаження – це DLC від Qualcomm Snapdragon. Далі йде OpenCV, який завантажує два файли окремо за 456 мс, що є прийнятним результатом. Нарешті, інтерпретатор формату flatbuffer є найшвидшим серед усіх. У таблиці 2 показано середній час завантаження кожної моделі на основі платформи та формату.

Тим не менш, середній час завантаження не є визначальним ключовим фактором. Це пов'язано з тим, що моделі лише один раз завантажуються в пам'ять на початку процесу.

#### **1.5.4 Точність**

Точність не залежить від платформи машинного навчання, де виконується нейронна мережа. Це безпосередньо залежить від самого алгоритму та отриманого навчання. Отже, три платформи працюють однаково з точки зору точності. Єдиний спосіб підвищити точність і змінити швидкість і розмір моделі - це оптимізація моделі.

#### **1.5.5 Швидкість**

У обмежених середовищах швидкість є ключовим показником, на який слід впливати. Час висновку — це час, який проходить від початку обробки кадру до отримання результатів з точки зору розпізнавання об'єкта. OpenCV є найповільнішим. Аналогічно, TFL отримав ті самі результати. Snapdragon Library показала найкращий результат майже з двома кадрами в секунду. Крім того, на рис. 2 показано кумулятивне середнє значення для 500 кадрів для кожної платформи. Оскільки TFL і OpenCV виконуються в ЦП, перші ітерації процесів вимагають більше часу через доступ до пам'яті. З часом ці значення досягають контрольованого та стабільного виконання. Як зазначалося, час висновку для TFL і OpenCV поступово збільшується. Це означає, що обидві платформи зазнають насичення, коли кадри обробляються нейронною мережею. На відміну від цього, Snapdragon залишається стабільним для кожного кадру.

### **1.5.6 Середнє використання оперативної пам'яті**

Завдання, що вимагають багато обчислень, такі як розпізнавання об'єктів та обробка зображень, можуть призвести до більшого середнього використання оперативної пам'яті. У цьому підрозділі оцінюється використана оперативна пам'ять протягом години під час процесу розпізнавання об'єктів. TFL, Snapdragon і OpenCV використовують ОЗУ 236 МБ, 707 МБ і 633 МБ відповідно. Таким чином, TFL працює найкраще.

### **1.5.7 Максимальне використання оперативної пам'яті**

У певні моменти використання оперативної пам'яті може досягти піку в процесі розпізнавання, що спричинить аномалії в системі. TFL і Snapdragon працюють майже однаково, а OpenCV використовує на 0,5 ГБ більше пам'яті, коли досягає максимуму. Ці середнє та максимальне використання оперативної пам'яті не є значущими в системі смартфона, коли смартфон має більше 4 ГБ пам'яті.

### **1.5.8 Акумулятор**

У обмежених середовищах батарея може швидко розряджатися під час виконання алгоритмів із інтенсивними обчисленнями. У нашій системі споживання батареї було дуже однаковим для всіх відповідних платформ, оскільки найбільш ресурсомістким завданням була яскравість екрану під час відтворення відео, що є обов'язковим процесом для всіх. У нашій системі акумулятор ємністю 3280 мАг розрядився для всіх платформ після години роботи CNN.

### 1.5.9 Температура

Процес із інтенсивними обчисленнями впливає на стабілізацію температури, особливо в умовах обмеженого середовища. Висока температура в пристрої може значною мірою знизити продуктивність процесів машинного навчання. У нашому випадку використання температура збільшувалася, поки не досягла плато 46–47°C для всіх платформ.

### 1.5.10 Трекер

Як визначено в підрозділі швидкості, швидкість реального часу може бути недосяжною за допомогою глибоких моделей CNN, таких як YOLOv3, у системах із обмеженнями; тому трекер як зовнішній допоміжний засіб необхідний для створення візуального сприйняття реального часу для кінцевого користувача. Серед платформ машинного навчання, які вивчаються в цій статті, OpenCV і TFL є єдиними платформами, які забезпечують трекери. В результаті Snapdragon не був оцінений в цьому розділі. Є дві причини на користь TFL, коли справа доходить до відстеження. Перш за все, TensorFlow швидший за OpenCV під час відстеження об'єкта. По-друге, TensorFlow має можливість одночасно виконувати трекер, щоб стежити за кількома розпізнаними об'єктами паралельно, використовуючи різні ядра. Ці причини роблять TFL найкращим вибором для відстеження кількох об'єктів на екрані.

## Розділ 2

### Мобільний додаток

#### 2.1 Що таке нативна розробка мобільних додатків

Термін нативна розробка нативного додатка відноситься до створення мобільного додатка виключно для однієї платформи. Додаток створено з використанням мов програмування та інструментів, специфічних для однієї платформи. Наприклад, ви можете розробити нативну програму для Android з Java або Kotlin і вибрати Swift і Objective-C для програм iOS.

Відомо, що нативні програми забезпечують винятковий користувацький досвід, оскільки вони, як правило, мають високу продуктивність. Користувацький досвід також покращується, оскільки візуальні елементи адаптуються до UX платформи.

#### 2.2 Плюси розробки нативних додатків

- Широка функціональність

Ви матимете доступ до всіх API та інструментів, наданих платформою, над якою ви працюєте. Технічно немає обмежень щодо того, як програмісти можуть працювати з новим додатком.

- Краща підтримка AppStore/Google Play Market

Рівну програму легше публікувати, і вона зазвичай займає вищий рейтинг у магазині додатків платформи, оскільки вона забезпечує кращу продуктивність та швидкість.

- Підвищена масштабованість

Програми, створені для рідного середовища, також мають тенденцію бути більш масштабованими завдяки гнучкості в управлінні ресурсами та набору доступних інструментів.

- Висока продуктивність і чудовий UX

Безпосередня взаємодія між кодом і базовими ресурсами призводить до високої продуктивності. Крім того, нативні програми зазвичай мають кращий UX, який є синонімом платформи.

## **2.3 Мінуси розробки нативних додатків**

- Велика ціна проекту

Створення рідних програм може бути дорогим, якщо вам потрібно запуснути як для iOS, так і для Android. Це означає, що вам потрібно буде розгорнути дві команди, які працюють на різних платформах.

- Забирає багато часу

Розробка нативного додатка займає багато часу, оскільки роботу, виконану для однієї платформи, неможливо повторити на іншій. Натомість для роботи над іншою версією потрібна окрема команда.

## **2.4 Що таке кросплатформна розробка**

Міжплатформна розробка вказує на процес створення програми, яка працює на кількох платформах. Це робиться за допомогою таких інструментів, як Flutter, React Native і Xamarin, де створені програми можна розгорнути як на Android, так і на iOS.

Хоча кросплатформна розробка економить час і кошти, ви ризикуєте пожертвувати якістю в цьому процесі. Важко налаштувати програму, яка оптимально працює на різних платформах, і під час роботи додатку потрібен додатковий рівень абстракції, що призведе до зниження продуктивності.

## **2.5 Плюси кросплатформної розробки додатків**

- Не така велика ціна, порівняно з нативною розробкою

Замість того, щоб мати дві команди розробників, вам знадобиться лише одна, щоб створити кросплатформний додаток. Таким чином, ви заощадите на витратах на розробку.

- Швидший розвиток

Для створення програми, яка працює на кількох платформах, потрібен лише один цикл розробки.

- Єдина кодова база

Оскільки додаток створюється за допомогою одного кросплатформного інструмента розробки, створюється лише одна кодова база.

## 2.6 Мінуси кросплатформної розробки додатків

- Повільніший додаток

Потреба в додатковому рівні абстракції та процесу візуалізації робить кросплатформний додаток повільнішим, ніж його рідний аналог.

- Обмежена функціональність

У розробників можуть виникнути труднощі з доступом до функцій смартфона, таких як мікрофон, камера та геолокація способами, можливими для рідної програми.

- Обмежений UX

Міжплатформні програми не можуть скористатися перевагами рідних компонентів UX. Тому він не може забезпечити той самий досвід UX, який звик до платформи.

Для даної задачі було вибрано саме кросплатформну розробку додатку, щоб покрити більшу кількість користувачів, але лишається вибрати якою технологією писати додаток. Розглянемо дві провідні технології для кросплатформної розробки - Flutter і React Native.

## 2.7 Flutter

Flutter - це портативний набір інструментів інтерфейсу користувача для створення власно скомпільованих додатків для мобільних пристроїв, веб сторінок та комп'ютерних додатків з єдиною кодовою базою. Розроблений фреймворк компанією Google, використовує мову Dart.

## 2.8 React Native

React Native - це фреймворк для створення нативних програм за допомогою React. Розроблений компанією Facebook і використовує мову JavaScript.

Обидві технології мають як плюси, так і мінуси. Ми будемо порівнювати продуктивність, досвід розробки, різницю між Dart і JavaScript, стандартні бібліотеки та багато іншого.

Важко порівнювати продуктивність будь-якого фреймворку. Flutter і React Native достатньо швидкі для більшості ситуацій. Історично продуктивність Flutter була кращою ніж у React Native. Різниця продуктивності в основному пов'язана з асинхронним мостом між native code і JavaScript. Flutter використовує відому графічну бібліотеку Skia.

Налаштувати середовище для розробки Flutter, як правило, легше, ніж React Native. Hot Reload у Flutter зазвичай працює краще, ніж Fast Refresh у React Native. У них є дуже гарні інструменти для debugging, profiling та роботи з деревом віджетів. Їхнє вбудоване end-to-end тестування набагато краще, ніж у React Native Detox.

Flutter компілює код Dart на нативний і має досить круту модель інтеграції з рідним кодом, використовуючи Platform Channels. Він дозволяє виконувати синхронні вихідні виклики та дозволяє писати нативний код на Swift та Kotlin.

У React Native інтеграція з native вимагає певного досвіду. Не кажучи вже про те, щоб боротися з обмеженнями React Native bridge.

У зв'язку з цими перевагами я вибрав Flutter для своєї роботи.

## Розділ 3

### Програмна реалізація

Щоб інтегрувати TensorFlow lite у нашу програму Flutter, нам потрібно встановити пакет tflite, і нам потрібні два файли model.tflite та labels.txt . model.tflite – це навчена модель, а файл labels.txt – це текстовий файл, що містить усі мітки. Багато веб-сайтів надають нам можливість тренувати нашу модель з нашим набором даних і розгорнути їх на TensorFlow Lite, і ми можемо безпосередньо отримати ці два файли звідти.

Для навчання моделі було використано близько 300 картинок різних предметів, таких як ноутбук, клавіатура, ручка, мишка, екран, машина. Навчання моделі проходило на сайті: <https://teachablemachine.withgoogle.com/>

В Андроїд конфігураціях був поставлений minSdkVersion - 21, а також додані наступні налаштування:

```
aaptOptions {  
    noCompress 'tflite'  
    noCompress 'lite'  
}
```

Після цього можна приступати до написання самого додатку. Для початку нам потрібно ініціалізувати камеру:

```
List<CameraDescription> cameras;
```

```
Future<void> main() async {
```

```
WidgetsFlutterBinding.ensureInitialized();
cameras = await availableCameras();
runApp(MyApp());
}
```

Після цього нам потрібно через `CameraController` відловлювати всі зміни картинки, для цього ми використаємо `.startImageStream`:

```
CameraImage cameraImage;
CameraController cameraController;
initCamera() {
  cameraController = CameraController(cameras[0], ResolutionPreset.medium);
  cameraController.initialize().then((value) {
    if (!mounted) return;
    setState(() {
      cameraController.startImageStream((image) {
        cameraImage = image;
        runModel();
      });
    });
  });
}
```

`Tflite` надає нам метод `loadModel` для завантаження нашої моделі. Він приймає два значення шляху до файлу моделі та мітки шляху до файлу.

```
Future loadModel() async {
  Tflite.close();
  await Tflite.loadModel(
    model: "assets/ssd_mobilenet.tflite",
```

```
    labels: "assets/ssd_mobilenet.txt");  
}
```

runModel - у цьому методі ми запускаємо модель за допомогою Tflite. Тут ми використовуємо пряму трансляцію зображення, тому нам доведеться використовувати метод detectObjectOnFrame для запуску нашої моделі.

```
runModel() async {  
  recognitionsList = await Tflite.detectObjectOnFrame(  
    bytesList: cameraImage.planes.map((plane) {  
      return plane.bytes;  
    }).toList(),  
    imageHeight: cameraImage.height,  
    imageWidth: cameraImage.width,  
    imageMean: 127.5,  
    imageStd: 127.5,  
    numResultsPerClass: 1,  
    threshold: 0.4,  
  );  
  
  setState() {  
    cameraImage;  
  });  
}
```

Показати обмежувальні рамки навколо розпізнаних об'єктів:

```
List<Widget> displayBoxesAroundRecognizedObjects(Size screen) {  
  if (recognitionsList == null) return [];
```

```
double factorX = screen.width;
double factorY = screen.height;
```

```
Color colorPick = Colors.pink;
```

```
return recognitionsList.map((result) {
  return Positioned(
    left: result["rect"]["x"] * factorX,
    top: result["rect"]["y"] * factorY,
    width: result["rect"]["w"] * factorX,
    height: result["rect"]["h"] * factorY,
    child: Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.all(Radius.circular(10.0)),
        border: Border.all(color: Colors.pink, width: 2.0),
      ),
      child: Text(
        "${result['detectedClass']} ${(result['confidenceInClass'] *
100).toStringAsFixed(0)}%",
        style: TextStyle(
          background: Paint()..color = colorPick,
          color: Colors.black,
          fontSize: 18.0,
        ),
      ),
    ),
  );
}).toList();
}
```

Це поле, яке буде відображатися навколо виявленого об'єкта. Кожен елемент `recognitionsList` містить такі деталі

```
{
  detectedClass: "car",
  confidenceInClass: 0.123,
  rect: {
    x: 0.15,
    y: 0.33,
    w: 0.80,
    h: 0.27
  }
}
```

`detetedClass` — ім'я виявленого об'єкта. `confidenceInClass * 100` — це % правильності. `rect` — це розміри об'єкта. Ми можемо використовувати ці параметри для відображення квадратів навколо ідентифікованого об'єкта.

Пакет `Tflite` надає нам `CameraPreview` віджет для попереднього перегляду камери на екрані програми, він потребує `cameraController`.

```
AspectRatio(
  aspectRatio: cameraController.value.aspectRatio,
  child: CameraPreview(cameraController)
```

Щоб разом відобразити блоки та попередній перегляд камери, нам потрібен `Stack`. `displayBoxesAroundRecognizedObjects` повертають список ящиків, нам потрібно додати цей список до стека, щоб ми створили змінну списку. Потрібно додати поля в список.

```
List<Widget> list = [];
```

```
list.add(
```

```
  Positioned(
```

```
    top: 0.0,
```

```
    left: 0.0,
```

```
    width: size.width,
```

```
    height: size.height - 100,
```

```
    child: Container(
```

```
      height: size.height - 100,
```

```
      child: (!cameraController.value.isInitialized)
```

```
        ? new Container()
```

```
        : AspectRatio(
```

```
          aspectRatio: cameraController.value.aspectRatio,
```

```
          child: CameraPreview(cameraController),
```

```
        ),
```

```
    ),
```

```
  ),
```

```
);
```

```
if (cameraImage != null) {
```

```
  list.addAll(displayBoxesAroundRecognizedObjects(size));
```

```
}
```

## Висновок

Було досліджено різні архітектури для побудови нейромереж для задач ідентифікації об'єктів у мобільних додатках. Був розроблений кросплатформний додаток за допомогою Flutter для ідентифікації об'єктів на основі однієї з найкращих архітектур, а саме TensorFlow lite. У планах удосконалити програмне забезпечення, а саме: розширити можливість класифікувати більшу кількість об'єктів.

Завдяки використанню нейромереж, при розробці програмного забезпечення вдалося уникнути жорсткої прив'язки до типів об'єктів, а також збільшити ймовірність коректної роботи в нестандартних ситуаціях. Це дало можливість використовувати і «навчати» одну і ту ж архітектуру для абсолютно різних задач. Наприклад, при зміні кількості класів об'єктів потрібно лише заново «навчити» нейромережу, не змінюючи її архітектуру. Нейромережа стійка до пошкоджень, при пошкодженні одного або декількох нейронів її точність і працездатність суттєво не змінюється, для сильнішого впливу потрібні дуже серйозні пошкодження.

Програма досить точно визначає контур для заданих об'єктів, а також, якщо на картинці вони відсутні, то ніяких помилок немає. Це досягається завдяки особливостям вибраної архітектури для розпізнавання (CNN), а також достатньою кількістю тренувальних даних і їх правильній розмітці.

На основі наведених вище результатів, можна зробити висновок, що програма здатно правильно розпізнає об'єкти на картинці, навіть якщо їх є багато і вони перекриваються.

## Список використаної літератури

[1]

Real Time Object Detection and Recognition using Deep Learning Methods// Sai Krishna Chadalawada.

[2]

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[3]

C. C. Nguyen, G. S. Tran, T. P. Nghiem, N. Q. Doan, D. Gratadour, J. C. Burie, and C. M. Luong, “Towards real-time smile detection based on faster region convolutional neural network,” in 2018 1st International Conference on Multimedia Analysis and Pattern Recognition (MAPR). IEEE, 2018, pp. 1–6.

[4]

Z. Deng, H. Sun, S. Zhou, J. Zhao, L. Lei, and H. Zou, “Multi-scale object detection in remote sensing imagery with convolutional neural networks,” ISPRS journal of photogrammetry and remote sensing, vol. 145, pp. 3–22, 2018.

[5]

Flutter site. [Online]. Available: <https://flutter.dev/>.

[6]

React Native site. [Online]. Available: <https://reactnative.dev/>.