ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. ML — Machine Learning або машинне навчання
2. AI — Artificial Intelligence або штучний інтелект
3. IoT — Internet of things або інтернет речей
4. ANN — Artificial Neural Networks або штучні нейронні мережі
5. SDK — Software development kit або засіб для розробки програмного забезпечення
6. API — Application program interface, інтерфейс взаємодії з програмою
7. R-CNN —- Region-Based Convolutional Neural Network, регіони з функціями згорткових нейронних мереж
8. CNN — згорткові нейронні мережі
9. MVC — Model-View-Controller, Модель-Вигляд-Контролер
10. MVP — Model-View-Presenter, Модель-Вигляд-Презентер
11. MVVM — Model-View-ViewModel, Модель-Вигляд-Модель вигляду

ВСТУП

Сучасний бурхливий розвиток технологій призвів до збільшення потужності та доступності обчислювальних пристроїв, що у свою чергу дозволяє виконувати складні розрахунки за допомогою комплексних алгоритмів, що вимагають значних ресурсів. Серед значної кількості технологій, що з'явилися у результаті збільшення можливостей обчислювальної техніки одним з найпопулярніших напрямків є практична реалізація штучного інтелекту та засобів машинного навчання.

Машинне навчання — це програма штучного інтелекту, що надає системі можливість автоматично навчатися та вдосконалюватися на основі досвіду, а не явного програмування [3]. Це можливо, тому, що сьогодні доступна велика кількість даних, які дозволяють навчати машини, а не програмувати. Існує думка, що саме розвиток цієї технології є новою великою технічною революцією, яка може аналізувати величезну кількість даних.

Станом на сьогоднішній день, майже всі комплексні рішення великих програмних систем використовують засоби машинного навчання. Машинне навчання змінює світ приносячи значний вклад у всі сегменти людського життя, зокрема: медичні послуги, освіту, їжу, транспорт, розваги та багато іншого. Такі технолої як Інтернет Речей і хмарні обчислення найактивніше впроваджують машинне навчання, щоб зробити об’єкти та гаджети “розумними” для себе. Машинне навчання пропонує потенційну цінність для компаній, які намагаються використовувати великі дані для задоволення клієнтів. Прихований шаблон, захований у

даних, може бути дуже корисним для бізнесу. Машинне навчання дає можливість проаналізувати великі обсяги даних, що забезпечує отримання більш точних результатів, що можуть допомогти в визначені можливостей отримання прибутку, чи виявити можливі приховані небезпеки. Широке значення для всіх видів завдань введення даних і класифікації, які раніше вимагали втручання людини, тепер можуть виконуватися машинами. Експерти з екології використовують машинне навчання та датчики з підтримкою штучного інтелекту для аналізу даних із тисяч джерел, щоб зробити точний прогноз забруднення та прогноз погоди. Машинне навчання може надавати попередження про збій системи, щоб плани резервного копіювання або відновлення могли бути виконані вчасно. Це зменшує час простою бізнесу. ML допомагає краще прогнозувати попит і може бути передовою технологією для управління змінами пропозиції. Це допомагає точно поділяти ринок і відповідно планує маркетингові стратегії, що безсумнівно покращує рентабельність інвестицій на маркетинговий бюджет. Банківська та фінансова галузь значною мірою покладається на ML для таких речей, як обслуговування клієнтів, захист від шахрайства, інвестиції тощо.

Згідно з статистики зібраної платформою статистики WorldoMeters U.N. data станом на 2021 рік в світі є 6,37 міліярдів користувачів смартфонів та прогнозується тенденція росту кількості користувачів ще більше [1]. Дивлячись на дану інформацію, стає зрозуміло, що розробка під мобільні платформи є однією з найприбутковіших і найбільш затребуваних технологій в сучасному світі. Мобільні додатки є швидшими у виконанні ніж їх веб відповідники, є доступнішими і більш практичними. Розробка для мобільних платформ дозволяє компаніям створити користувачам більш персоніфікований досвід користування; дозволяє

користувачам використовувати можливості пристрою, зокрема камеру, різноманітні датчити, тощо; та покращує брендованість продукту. Враховуючи перераховані переваги не дивно, що 88% користувачів надають перевагу саме мобільним додаткам на ринку [2].

Враховуючи популярність мобільних додатків та тенденції росту машинного навчання, дослідження використання засобів машинного навчання для мобільних платформ є актуальним напрямом наукових розвідок. Метою даної магістерської роботи є дослідження можливостей використання засобів машинного навчання при розробці на мобільні платформи. Для досягнення поставленої мети було сформовано наступні завдання:

1. проаналізувати можливі засоби машинного навчання доступні для використання на мобільних платформах;
2. проаналізувати засоби розробки додатків для платформ IOS та Android;
3. спроектувати та реалізувати додаток для однієї з мобільних платформ;
4. протестувати розроблений додаток.

Після виконання всіх поставлених завдань було продемонстровано практичне використання засобів машинного навчання для розробки додатків на мобільні платформи.

1 СУЧАСНИЙ СТАН ДОСЛІДЖЕНЬ ЗАСОБІВ МАШИННОГО НАВЧАННЯ

1.1 Загальні відомості

Машинне навчання — це розділ інформатики, який займається побудовою алгоритмів які, спираються на збірку прикладів якогось явища. Ці приклади можуть виникнути з природи, бути створені вручну людьми або згенеровані іншим алгоритмом [4]. Машинне навчання також можна визначити як процес вирішення практичної задачі за збирання набору даних і алгоритмічне побудова статистичної моделі на основі цього набору даних. Передбачається, що ця статистична модель якось використовується для вирішення практичної проблеми.

Дуже часто машинне навчання прирівнюють до статистичного аналізу даних, але це не зовсім вірно. Так, машинне навчання має статистичну основу, але дана технологія робить зовсім інші припущення ніж статистика, через те, що цілі та засоби машинного навчання відрізняються від цілей статистичного аналізу даних. Наприклад, в машинному навчанні, дані обробляються за допомогою нейронних мереж та графів з використанням не опрацьованих даних, що розділяються на тренувальні та тестові дані, а в статистиці моделі використовуються для створення наукового припущення на основі малої вибірки [6].

Для визначення необхідності використання засобів машинного навчання потрібно звертати увагу на складність програми та потреби в адаптивності [5].

Завдання які є занадто складними для того, щоб їх явно запрограмувати:

1. Рутинні завдання з обробкою великого об’єму даних. Існує багато завдань, які ми виконуємо як рутину, але наш самоаналіз щодо того, як ми робимо їх недостатньо складно, щоб витягти чітко визначену програму. Прикладами таких завдань є керування автомобілем, розпізнавання мовлення та розуміння образу. У всіх цих завданнях найсучасніші програми машинного навчання, програми, які «вчаться на своєму досвіді», досягають досить задовільні результати, коли вони були піддані достатній кількості навчальних прикладів.

2. Завдання, що виходять за межі людських можливостей. Інша широка група завдань, які використовують методи машинного навчання, пов’язані з аналізом дуже великі та складні набори даних: астрономічні дані, точний медичний архів в медичні знання, прогноз погоди, аналіз геномних даних, веб пошукові системи та електронна комерція. З більшою кількістю доступних даних, записаних в цифровому вигляді, стає очевидним, що в архівах даних є надто великі і занадто великі скарби значущої інформації. складний для розуміння людьми. Навчання виявляти значущі закономірності у великих і складних наборах даних є багатообіцяючою областю, в якій комбінація програм, які навчаються з майже необмеженою пам'яттю потужність і постійно зростаюча швидкість обробки комп'ютерів відкривають нові горизонти.

Для кращого різниці між традиційним, програмуванням та засобами машинного навчання потрібно зрозуміти, що саме представляє собою явне програмування електронно обчислювальної машини. На найвищому рівні традиційне програмування передбачає створення правил, які діють на дані і які віддають відповіді. Наприклад, розглядаючи сценарій знаходження

рівняння прямої за заданими координатами двох точок, потрібно написати програму таким чином, щоб вона користувалася правилами лінійної геометрії, зокрема загальним рівнянням прямої, знаходила коефіцієнти рівняння прямої. Тоді записавши вказівки, ми могли б отримати нове рівняння прямої відносно нових даних. Основною роботою програміста при даному сценарії було з'ясування правил і вказівок для програми. Це те, чим займаються програмісти зазвичай, розбиваючи проблему на задачі і правила згідно яких можна з вхідних даних отримати вихідні, а потім перетворити дані правила на терміни, що може розуміти машина засобами мови програмування. Але не завжди просто можна висловити ці правила, саме тоді стає у нагоді машине навчання [7].

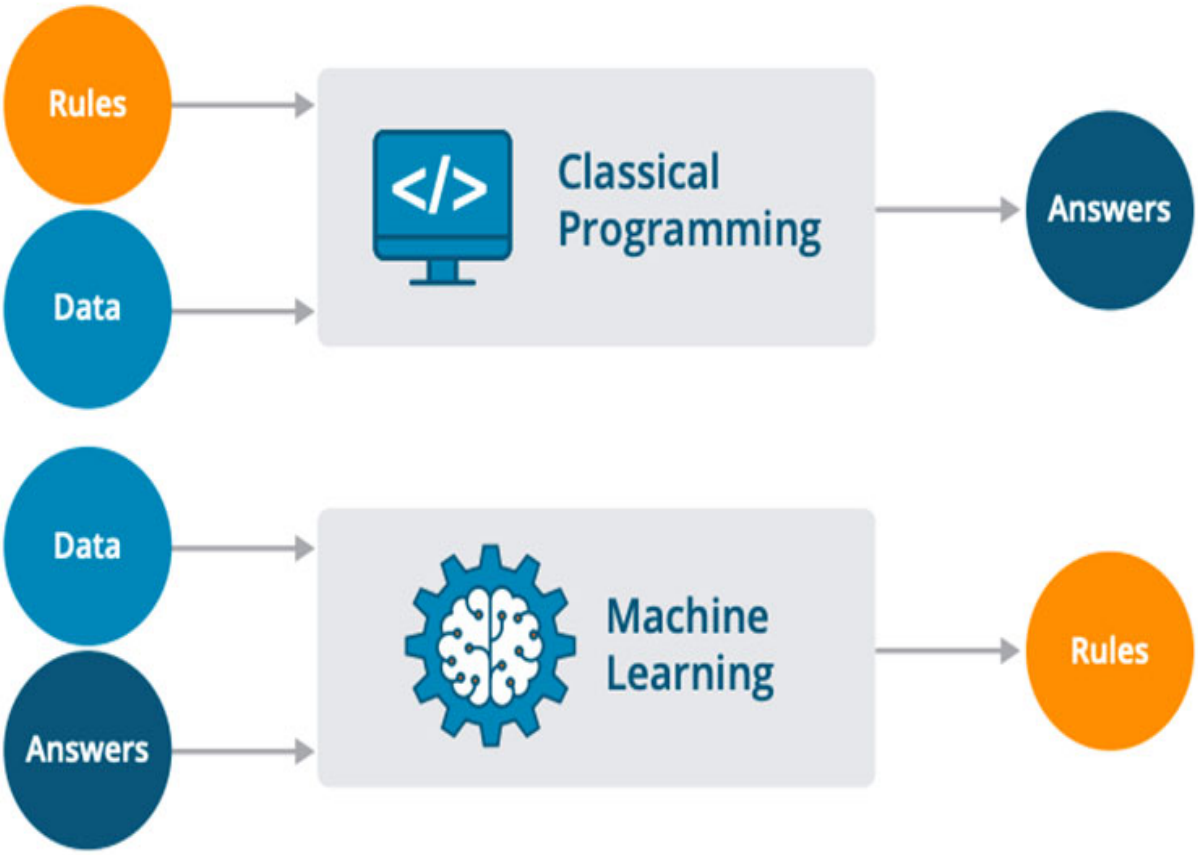

Рисунок 1.1 — Різниця між традиційним програмуванням та машинним навчанням

Засоби машинного навчання надають програмам можливість адаптуватися до вхідних даних користувача. поведінка яких адаптується до їхніх вхідних даних – пропонують рішення ипові успішні застосування машинного навчання для вирішення таких проблем включають програми, які декодують рукописний текст, де фіксована програма може адаптуватися до змін почерку різних користувачів; програми виявлення спаму, що автоматично адаптуються до змін характеру спаму електронних листів; і програми розпізнавання мовлення.

1.2 Класифікація машинного навчання

Засоби машинного навчання прийнято класифікувати за способом тренування алгоритму, кожний з цих методів має власні переваги та недоліки. На сьогоднішній день, найпопулярнішими методами є кероване навчання, напівавтоматичне навчання, навчання без учителя та навчання з підкріпленням [2].

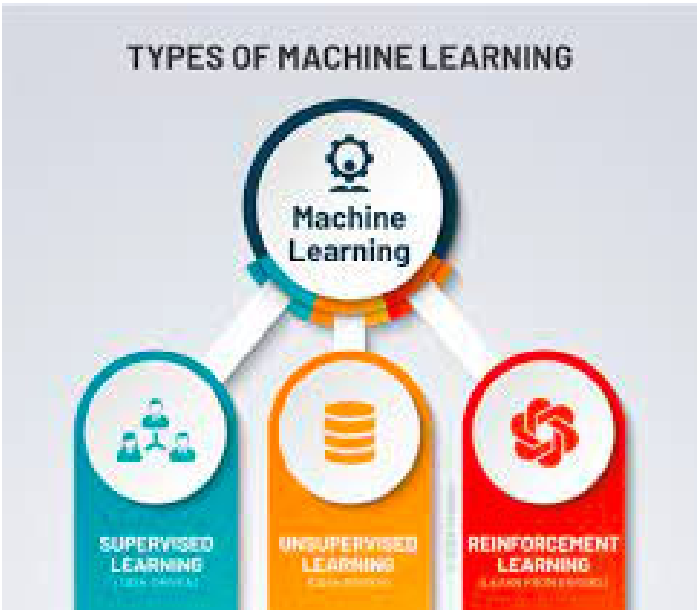

Рисунок 1.2 — типи машинного навчання

1.2.1 Контрольоване навчання

У контрольованому навчанні набір даних являє собою набір позначених прикладів $\{(x_i, y_i)\}_{i=1}^{N}$. Кожен елемент x_i серед N називається вектором ознак. Вектор ознак — це вектор, у якому кожен вимір $j = 1, \ldots, D$ містить значення, яке якось описує приклад. Це значення називається ознакою і позначається як $x^{(j)}$. Наприклад, якщо кожен приклад x у нашій колекції представляє людину, тоді перша ознака, $x^{(1)}$, може містити висоту в сантиметрах, друга ознака, $x^{(2)}$, може містити вагу в кілограмах, $x^{(3)}$може містити стать тощо. Для усіх прикладах у наборі даних об'єкт у позиції j у векторі ознак завжди містить таку саму інформацію. Це означає, що якщо $x_i^{(2)}$ і містить вагу в кілограмах у деякому прикладі x_i, тоді $x_k^{(2)}$ також міститиме вагу в кілограмах у кожному прикладі x_k, $k = 1, \ldots, N$. Мітка y_i може бути або елементом, що належить кінцевій множині класів {1, 2,...,C}, або дійсним числом, або складною структурою, наприклад вектор, матриця, дерево або графік. Значення y_i є або одним із скінченної множини класів, або дійсним числом. Ви можете побачити клас як категорію, до якої належить дані. Наприклад, якщо ваші дані – повідомлення електронної пошти і ваша проблема полягає в виявленні спаму, тоді у вас є два класи {spam, not_spam}. Метою алгоритму навчання з наглядом є використання набору даних для створення моделі який приймає вектор ознак x як вхід і виводить інформацію, яка дозволяє вивести мітку для цього вектора ознак.

Наприклад, модель, створена з використанням набору даних людей, може приймати як вхідні дані вектор ознак, що описує людину, і вивести ймовірність того, що ця людина має рак.

Розглянемо детальніше як працює процес навчання під наглядом. Даний процес починається зі збору даних. Дані для нагляду навчання — це сукупність пар (вхідних, вихідних). Вхідними можуть бути будь-які дані, наприклад листи електронної пошти, повідомлення, зображення або вимірювання датчика. Вихідними сигналами зазвичай є дійсні числа або мітки (наприклад, «спам», «не_спам», «кіт», «собака», «миша» тощо). У деяких випадках результати є векторами (наприклад, чотири координати прямокутника навколо людини на малюнку), послідовності (наприклад, [“прикметник”, «прикметник», «іменник»] для введення «великий красивий автомобіль») або мають іншу структуру. Скажімо, проблема, яку ви хочете вирішити за допомогою контрольованого навчання, — це виявлення спаму. Ви збираєте дані, наприклад, 10 000 листів електронної пошти, кожне з міткою «спам» або “not_spam” (ви можете додати ці мітки вручну або заплатити комусь, щоб він зробив це за нас). Далі ви повинні перетворити кожне повідомлення електронної пошти у вектор функцій. На основі свого досвіду аналітик вирішує, як перетворити реальну сутність. Наприклад, як перетворити листи, у вектор ознак. Один із поширених способів перетворення тексту в об’єкт вектора, який називається пакетом слів в термінології машинного навчання, це взяти словник англійських слів (скажімо, він містить 20 000 слів, відсортованих за алфавітом) і вкажіть, що в нашому векторі ознак:

1. перша ознака дорівнює одиниці, якщо лист містить займенник “купити”, якщо лист не містить даного слова ця функція дорівнює нулю;

2. друга ознака дорівнює одиниці, якщо емейл містить слово “реклама”, чи нуль, якщо не містить;

3. …..;

4. двадцятитисячна ознака дорівнює одиниці, якщо лист містить слово “продукт”, і нуль в протилежному випадку.

Ми повторюємо дану операцію для кожного електронного листа в нашій вибірці даних, що дасть нам 10 000 векторів ознак, кожний вектор матиме 20 000 полів і позначку “спам” / “не_спам”.

Тепер у нас є машиночитні вхідні дані, але вихідні мітки все ще мають форму тексту, який зрозумілий людині. Деякі алгоритми навчання вимагають перетворення міток у числа. Наприклад, деякі алгоритми вимагають числа на кшталт 0 (щоб представляти мітку «не_спам») і 1 (що позначає мітку «спам»). Для ілюстрації контрольованого навчання я використовую алгоритм називається машиною опорних векторів (SVM). Цей алгоритм вимагає, щоб позитивна мітка (в нашому випадку це "спам") має числове значення +1 (один) і негативну мітку ("не_спам") має значення ≠1 (мінус один). На цьому етапі у вас є набір даних і алгоритм навчання, тож ви готові застосувати алгоритм навчання набору даних, щоб отримати модель. SVM розглядає кожен вектор ознак як точку у просторі високої розмірності (у нашому випадку простір є 20 000-вимірним). Алгоритм розміщує всі вектори ознак на уявних 20 000-розмірний графік і малює уявну 20 000-мірну лінію (гіперплощину), яка розділяє приклади з позитивними мітками з прикладів з негативними мітками. У машинному навчанні межа, що розділяє приклади різних класів, називається межею рішення. Рівняння гіперплощини задається двома параметрами, дійсним вектором w він такої ж розмірності, як і наш вхідний вектор ознак x, і дійсне число b, як це:

$$wx - b = 0$$

Де вираз wx означає $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \cdots + w^{(D)}x^{(D)}$, і D це розмірність ознак вектора x. Тепер для визначення мітки при вхідному векторі ознак x використовується формула:

$$y = sign(wx - b)$$

У даному виразі операція sign -— це математичний оператор, що приймає будь яке значення і повертає +1, якщо вхідні дані були додатнім номером, чи -1, у всіх інших випадках.

Цілю алгоритму навчання, SVM у даному випадку, є пошук оптимальних значень w^* та b^* для параметрів w та b. Одразу як алгоритм машиного навчання визначить оптимальні значення для цих параметрів модель для визначення мітки матиме вигляд:

$$y = sign(w^*x - b^*)$$

Тому, щоб передбачити, чи є повідомлення електронної пошти спамом чи не спамом за допомогою моделі SVM, потрібно взяти текст повідомлення, перетворити його у вектор ознак, а потім помножити його вектор на w^* та відняти b^* і взяти знак результату. Це дасть нам прогноз (+1 означає «спам», -1 означає «не_спам»). Тепер розглянемо як саме машина знаходить w^* і b^*. Це вирішується за допомогою оптимізації. Машини добре оптимізують функції під обмеженнями. Тож які обмеження потрібно задовільнити? Перш за все, потрібно, щоб модель передбачала мітки наших 10 000 прикладів правильно. Пам’ятаймо, що кожен приклад i = 1,..., 10000 дорівнює заданій парі (x_i, y_i), де x_i — вектор ознак прикладу i, а y_i — його мітка, що приймає значення -1 або +1. Отже, обмеження:

- $wx_i - b \geq 1$ при $y_i = +1$

- $wx_i - b \leq -1$ при $y_i = -1$

Також необхідно, щоб гіперплощина відокремлювала позитивні приклади від негативних з найбільшою маржою. Маржа — це відстань між найближчими прикладами двох класів, як визначено межею рішення. Великий запас сприяє кращому узагальненню, саме так модель буде класифікувати нові приклади в майбутньому, щоб цього досягти, нам потрібно мінімізувати евклідову норму w , позначену як $|w|$, за допомогою формули:

$$|w| = \sqrt{\left(\Sigma_{j=1}^{D} \left(w_j^{(i)}\right)^2\right)}$$

Виходячи з цього завдання оптимізації полягатиме в зменшені евклідової норми, що можна записати в спрощеному варіанті за допомогою виразу:

$$y_i(wx_i - b) \geq 1$$

Рішенням даної проблеми з заданими w^* і b^* називають статистичною моделю, або просто моделю.

Для двохвимірних векторів ознак вирішення проблеми можна побачити на Рисунку 1.3. Фіолетові кружки відповідають за позитивні значення, а зелені - за негативі. А лінія, задана рівнянням $wx - b = 0$, є межею рішень [8].

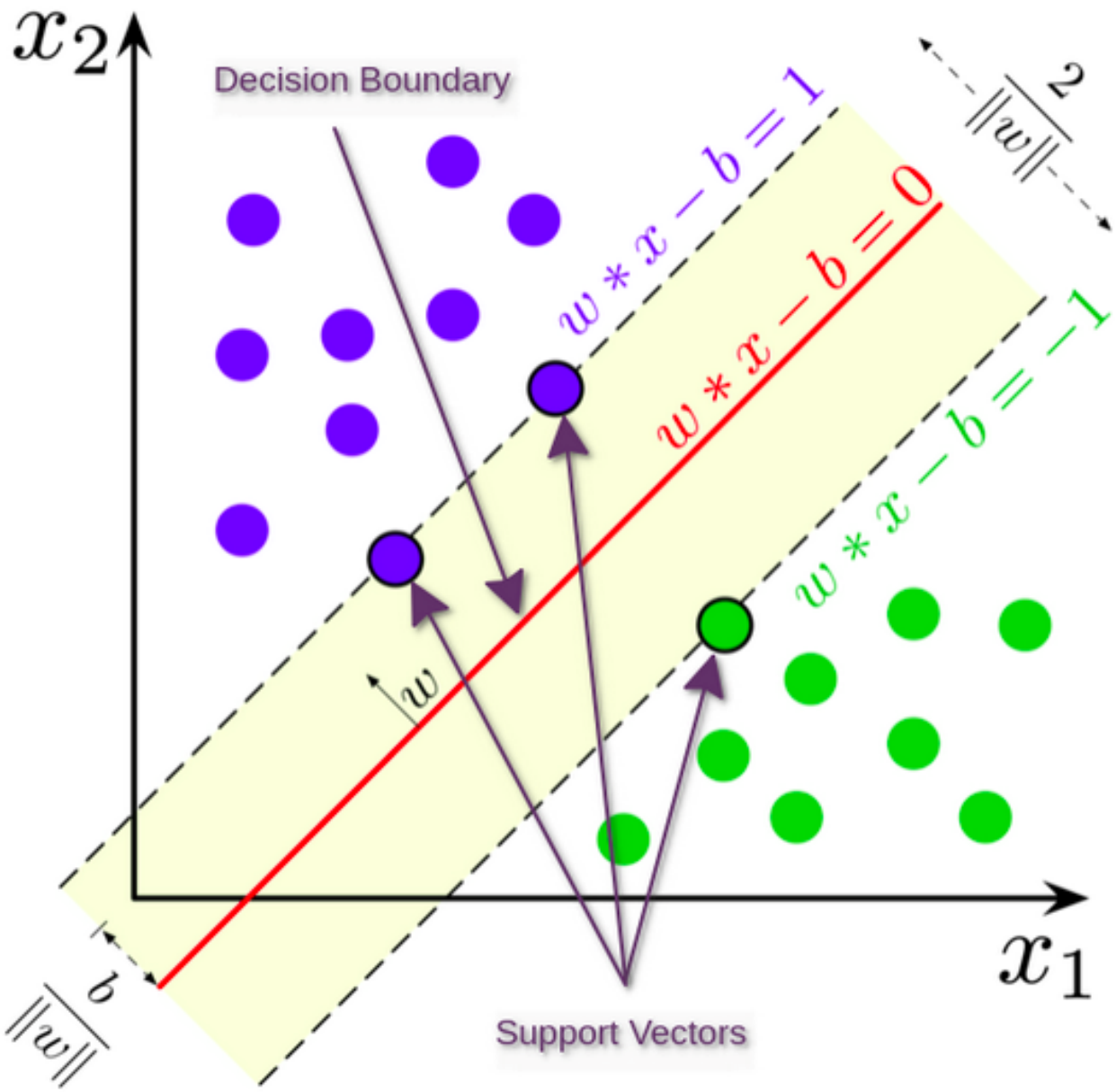

Рисунок 1.3 — Приклад вигляду SVM моделі для двохмірного вектору ознак

Ось як працюють машини опорних векторів (SVM). Ця конкретна версія алгоритму будується так звана лінійна модель. Його називають лінійним, оскільки межа рішення є прямою лінією (або площиною, або гіперплощиною). SVM також може включати ядра, які можуть приймати рішення границя довільно нелінійна. У деяких випадках ідеально

розділити неможливо дві групи точок через шум у даних, помилки маркування або викиди (приклади які дуже відрізняються від «типового» прикладу в наборі даних). Інша версія SVM також може включити гіперпараметр штрафу за неправильну класифікацію навчальних прикладів конкретних класів. Будь-який алгоритм навчання класифікації, який будує модель неявно або явно створює межі рішення. Межа рішення може бути прямою, вигнутою, мати складну форму, або бути суперпозицією деяких геометричних фігур. Форма межі рішення визначає точність моделі (тобто співвідношення прикладів, мітки яких передбачені правильно). Форма межі рішення, спосіб її алгоритмічного чи математичного обчислення на основі навчальні дані відрізняють один алгоритм навчання від іншого. На практиці існують ще дві основні відмінності алгоритмів навчання: швидкість побудови моделі та час обробки прогнозу. У багатьох практичних випадках надають перевагу алгоритму навчання, який швидко створює менш точну модель.

1.2.2 Напівавтоматичне навчання

У напів керованому навчанні набір даних містить як мічені, так і немарковані приклади. Зазвичай кількість немаркованих прикладів значно перевищує кількість позначених приклади. Мета алгоритму навчання з напів наглядом така ж, як і мета алгоритм навчання під наглядом. Сподіваємося, що використання багатьох не зазначених прикладів можна допомогти алгоритму навчання знайти (ми можемо сказати «виробляти» або «обчислювати») кращу модель. Може здатися не розумним, що навчання може отримати користь від додавання більшої кількості невизначених прикладів. Здається ніби ми додаємо до проблеми більше

невизначеності. Однак, коли ви додаєте приклади без міток, ви додаєте більше інформація про вашу проблему: більша вибірка краще відображає розподіл ймовірностей даних. Теоретично, алгоритм навчання повинен мати можливість використовувати цю додаткову інформацію [2].

1.2.3 Навчання без учителя

У неконтрольованому навчанні набір даних являє собою набір непозначених прикладів $\{x_i\}_{i=1}^{N}$. Знову ж таки, x — це вектор ознак, і мета алгоритму навчання без нагляду — це щоб створити модель, яка бере вектор ознак x як вхідні дані та перетворює його в інший вектор або у значення, яке можна використати для вирішення практичної задачі. Наприклад, при кластеризації модель повертає ідентифікатор кластера для кожного вектора ознак у наборі даних. При зменшенні розмірності вихідним результатом моделі є вектор ознак, який має менше особливості, ніж вхідні x; при виявленні викидів вихідним є реальне число, яке вказує чим x відрізняється від «типового» прикладу в наборі даних [2].

1.2.4 Навчання з підкріпленням

Навчання з підкріпленням — це під сфера машинного навчання, де машина «живе» в середовищі і здатна сприймати стан цього середовища як вектор особливостей. Машина може виконувати дії в будь-якому стані. Різні дії приносять різні винагороди, а також може перевести машину в інший стан середовища. Мета алгоритму навчання з підкріпленням полягає

в тому, щоб вивчити політику. Політика — це функція f (подібна до моделі в контрольованому навчанні), яка приймає вектор ознак стану як вхідні дані та виводить оптимальну дію для виконання в цьому стані. Дія є оптимальним, якщо воно максимізує очікувану середню винагороду [2].

1.2.5 Глибоке навчання

Однією з популярних сфер розвитку машинного навчання є глибоке навчання. Алгоритми глибокого навчання можна розглядати як складну та математично складну еволюцію алгоритмів машинного навчання. Останнім часом ця галузь привертає багато уваги, і недарма: останні події привели до результатів, які раніше вважалися неможливими.

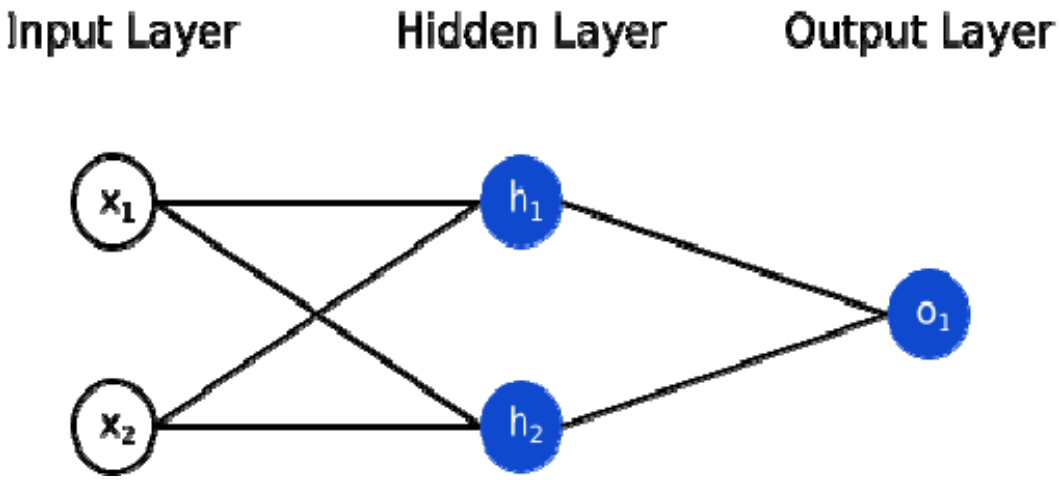

Рисунок 1.4 — Приклад простої штучної нейронної мережі

Глибоке навчання описує алгоритми, які аналізують дані з логічною структурою, подібною до того, як людина робить висновки. Зауважте, що це може статися як шляхом навчання під наглядом, так і без нагляду. Щоб досягти цього, програми глибокого навчання використовують

багатошарову структуру алгоритмів, яку називають штучною нейронною мережею [9]. Конструкція такої ANN натхнена біологічною нейронною мережею людського мозку, що веде до процесу навчання, який набагато ефективніший, ніж стандартні моделі машинного навчання.

2 СУЧАСНИЙ СТАН МОЖЛИВОСТЕЙ РОЗРОБКИ ДЛЯ МОБІЛЬНИХ ПЛАТФОРМ

2.1 Загальні відомості

Розробка мобільних додатків — це бурхливо розвиваючася індустрія в сучасному світі де розробники використовують різноманітні мови програмування для різних операційних систем щоб створювати додатки для смартфонів. Станом на сьогодні, це одна з найперспективніших галузей в розробці. Все більше компаній розуміють важливість власної мобільної аплікації. Навіть компанії, що були популярні на ринку, але не мали власних додатків інвестують в розробку мобільних додатків. Стає все більше стартапів пов’язаних з розробкою мобільних додатків у всіх сферах людського життя: шопінг, подорожі, відпочинок, харчування, тощо. Станом на червень минулого 2020 року, користувачі платформи Android мали можливість користуватися більш ніж 3,48 мільйонів додатків доступними для завантаження в PlayMarket, в той ж час користувачі платформи Apple IOS мали можливість завантажити 2.22 мільйони додатків [10].

2.2 Аналіз переваг розробки під мобільні платформи

Для того, щоб зрозуміти таку шалену популярність мобільних додатків потрібно проаналізувати переваги їх розробки.

Покращення ефективності в порівнянні з веб-сервісами. Оскільки бізнес-додатки створені на замовлення, враховуючи бізнес-вимоги клієнтів, вони діють як комплексний додаток, що виконує різноманітні функції та вирішує потребу користувача в користуванні кількох інших програм. Крім того, оскільки ці програми розроблені відповідно до стилю роботи специфічного клієнта, вони підвищують продуктивність співробітників і, в свою чергу, рентабельність інвестицій.

Звичайні програми створені для роботи з обмеженою кількістю ресурсів і процесів. У разі зростання вашого бізнесу ці програми можуть не витримати навантаження. З іншого боку, програми, виготовлені на замовлення, створені з урахуванням усіх цих параметрів і можуть бути легко масштабовані, коли виникне необхідність.

Загальні бізнес-програми можуть мати або не мати спеціалізованих функцій безпеки, які можуть поставити клієнтські бізнес-дані під загрозу. Наявність спеціального додатка, створеного виключно для специфічного бізнесу, може зміцнити систему безпеки даних, оскільки під час розробки додатків відповідні заходи безпеки будуть прийняті відповідно до потреб специфічного клієнтського бізнесу.

Загальні бізнес-програми можуть мати збої у роботі з наявним програмним забезпеченням. Програми, виготовлені на замовлення, створені з урахуванням клієнтського поточного програмного забезпечення для бізнесу, і, отже, гарантовано добре інтегруються та функціонують без будь-яких помилок.

Використовуючи звичайні програми для повсякденних бізнес-операцій, існують ризики в покладені на невідомих розробників мобільних додатків. Якщо їх компанія вирішить припинити роботу програми з якоїсь причини, вам залишається лише знайти нову програму та припинити

поточні операції. Створюючи клієнтський бізнес-додаток, клієнт отримує повний контроль над ним і втрачається необхідність відносин бізнесу з стороніми компаніями, що звісно позитивно впливає на брендованість продукції.

Індивідуальні бізнес-додатки дозволяють надсилати персоналізовані оновлення, пов’язані з клієнтськими продуктами та послугами, існуючим користувам в режимі реального часу. Крім цього, це дає змогу отримати доступ до деталей користувачів та отримати зворотній зв’язок, що можна використовувати для покращення довгострокових відносин з користувачами. В результаті цього значно збільшується лояльність до бізнесу.

Додавання простих форм та опитувань у спеціальний мобільний додаток може допомогти отримати потрібну інформацію про користувача. Крім того, що це стриманий спосіб збору інформації, він також економить час клієнтів і співробітників, оскільки їм не потрібно фізично подавати документи.

При великій поширеності мобільних додатків, не дивно, що і засобів для розробки є доволі багато. Засоби для розробки можна поділити на дві великі групи: кросплатформені та нативні [11].

2.3 Нативні засоби розробки під мобільні платформи

2.3.1 Нативні засоби для платформи IOS

2.3.1.1 Історичні відомості про платформу IOS

Перший iPhone був представлений на конференції Mac World на початку січня 2007 року і пізніше випущений у червні того ж року. Спочатку нативні програми сторонніх розробників були заборонені. Керівники Apple стверджували, що розробники можуть створювати веб-додатки (HTML/CSS), що будуть вести себе як рідні програми для iPhone. Розробники відступили від даної ідеї, і в жовтні 2007 року Apple оголосила про SDK, що дозволив розробникам створювати власні програми для iPhone. Багато хто стверджує, що рішення Apple дозволити розробникам створювати нативні програми було засноване на тому, що платформа Android незабаром мала вийти на ринок і була надзвичайно відкритою та гнучкою платформою в якій розробники могли робити те, що вони не могли в iOS. Рішення дозволити нативні додатки всередині iOS створили нову бізнес-модель для розробників, де невеликі проекти після декількох годин роботи перетворилися на повноцінні компанії. Станом на лютий 2012 року Apple App Store містив понад 725 700 додатків, які разом було завантажено більше ніж 25 мільярдів разів. З червня 2007 року айфони допомогли розширити мобільний бум. Компанія Apple перетворила покоління iPod користувачів iPhone із простими та ефективними методами користувальницького інтерфейсу, і зробила свій продукт «крутим». У травні 2010 року 59 відсотків усього споживання мобільних веб-даних у США припадало на iPhone. iPhone змінив гру: персональний органайзер, ігрова платформа, веб-браузер і телефон все в одній маленькій упаковці.

Коли Стів Джобс представив світу iPhone, він проголосив його революційним продуктом — це змінило все, з абсолютно новим «мультитач» інтерфейсом. Він став справжнім захоплюючим проривом, як і інтерфейс комп’ютерної миші, який був представлений у 1960-х роках. Саме iPhone поклав початок мобільному буму. iPhone був проданий

споживачам, а його простота використання змусила майже всіх захотіти придбати даний девайс. На момент його випуску основна унікальність iPhone виходила далеко за межі веб-браузера і тісно інтегрувала веб-функціональність. Хоча на ринку мобільних телефонів, на момент виходу iPhone, були “смартфони”, саме після виходу в продаж iPhone та хорошої маркентигової компанії Apple, виробники телефонів справді помітили перспективи розвитку даної галузі [11].

2.3.1.2. Мови програмування, що застосовуються при нативній розробці під платформу ios

2.3.1.2.1 Використання Objectvie-C при програмуванні для мобільних додатків

Objective-C це одна з основних мов програмування, яку використовують під час написання програмного забезпечення для OS X та iOS. Це наднабір мови програмування C, що надає об’єктно-орієнтовані можливості та динамічний час виконання. Objective-C успадковує синтаксис, примітивні типи та оператори керування потоком C і додає синтаксис для визначення класів і методів. Він також додає підтримку на рівні мови для керування графами об’єктів та літералів об’єктів, одночасно забезпечуючи динаміче введення та зв’язування, відкладаючи багато обов’язків до часу виконання.

Створюючи програми для OS X або iOS, більша частина часу витрачається на роботу з об’єктами. Ці об’єкти є екземплярами класів

Objective-C, деякі з яких надаються вам Cocoa або Cocoa Touch, а деякі є повністю створені розробником [13].

2.3.1.2.2 Використання Swift при програмуванні для мобільних додатків

Історія Swift набагато простіше і коротше. Він був створений компанією Apple і представлений на її всесвітній конференції розробників у 2014 році. Це абсолютно нова мова, розроблена для розробки програмного забезпечення для операційних систем Apple. З того часу мова отримала величезний розвиток. Остання стабільна версія була випущена 25 жовтня 2021 року під назвою 5.5.1.

Swift — це фантастичний спосіб писати програмне забезпечення, будь то для телефонів, настільних комп'ютерів, серверів чи будь-чого іншого, що може виконувати код. Це безпечна, швидка та інтерактивна мова програмування, яка поєднує найкраще в сучасному мовному мисленні з мудрістю інженерної культури Apple і різноманітним внеском спільноти з відкритим кодом. Компілятор оптимізовано для продуктивності, а мова оптимізована для розробки без шкоди ні для того, ні від іншого [15].

Swift дружній до новачків. Це мова програмування промислової якості, яка настільки ж виразна та приємна, як і мова сценаріїв. Написання коду Swift на ігровому майданчику дає змогу експериментувати з кодом і відразу бачити результати, не витрачаючи на створення та запуск програми.

Swift визначає великі класи поширених помилок програмування, приймаючи сучасні шаблони програмування:

- Змінні завжди ініціалізуються перед використанням.

- Індекси масиву перевіряються на наявність помилок за межами.
- Цілі числа перевіряються на переповнення.
- Додаткові параметри гарантують, що нульові значення обробляються явно.
- Управління пам'яттю здійснюється автоматично.

Обробка помилок дозволяє контролювати відновлення після несподіваних збоїв.

Swift-код зібрано й оптимізовано, щоб отримати максимальну віддачу від сучасного обладнання. Синтаксис і стандартна бібліотека були розроблені на основі керівного принципу, згідно з яким очевидний спосіб написання коду також має бути найкращим. Його поєднання безпеки та швидкості робить Swift чудовим вибором для всього, починаючи з «Hello, world!» на всю операційну систему [15].

Swift поєднує потужний висновок типу та узгодження шаблонів із сучасним, легким синтаксисом, що дозволяє чітко та стисло висловлювати складні ідеї. В результаті код не тільки легше писати, але також легше читати та підтримувати.

Swift створювався роками, і він продовжує розвиватися з новими функціями та можливостями.

2.3.1.3 Різниця між Objective-C та Swift

Порівняння між цими двома мовами програмування необхідне, тому, що Swift не є прямим наслідником Objective-C, ці дві мови мають різні можливості і одночасно використовуються для мобільної розробки.

2.3.1.3.1 Порівняння між середовищами розробки

Середовище розробки – це те, де відбувається магія. Важливо, чи є середовище, яке просте у використанні, наскільки воно дороге, і чи є на ринку альтернативні рішення. Якщо ви використовуєте Objective-C для розробки програми iOS або OS X, ви, швидше за все, використовуватимете IDE Xcode. Доступні й інші середовища, наприклад AppCode або Visual Studio Code від Microsoft, але для належної роботи їм потрібен Xcode. Xcode надає вам набори для розробки програмного забезпечення, які дають практично все, що вам знадобиться для розробки додатків iOS. Інструменти, компілятори, API та фреймворки — все це є. Крім того, є готові елементи, які можна налаштувати у своїй програмі. Xcode інтегрований з Cocoa і, що ще важливіше, Cocoa Touch — середовищем для розробки додатків, яке ви будете використовувати для iOS. Однак, якщо ви вирішите використовувати Swift, Cocoa Touch також дає змогу створювати програми для Apple Watch і Apple TV. Однак це не єдина різниця щодо середовища розробки. Існує середовище, що називається Swift Playgrounds. Ви можете використовувати його, щоб навчитися писати код на Swift, а також як середовище для тестування невеликих частин коду без компіляції всього коду та створення повної програми. Просто, коли у вас є ідея рішення, ви пишете код у Playgrounds і миттєво перевіряєте, як він працює. З Swift ви все ще можете працювати в тому ж середовищі, що й Objective-C, але ви також отримуєте доступ до новіших рішень, які розширюють можливості [14].

2.3.1.3.2 Порівняння користувацького інтерфейсу

Ці дві мови програмування є нативними для IOS. Тому не буде різниці, яку мову вибере розробкин, додатки виглядатимуть одинаково. Найбільшою різницею цих мов буде сумісність з різними версіями IOS та простота розробки користувацького інтерфейсу [14].

Objective-C є дуже старою технологією для розробки, тому ваша програма працюватиме на будь-якій доступній версії iOS. Swift, з іншого боку, потребує щонайменше iOS 7. Це означає, що вам потрібно проаналізувати, які пристрої є у ваших користувачів, і вирішити, чи можна обмежитися тими, з iPhone та iPad, випущеними після 2013 року. Чесно кажучи, існує не так багато людей, які все ще користуються пристроями віком від 7 років, тому це не повинно бути проблемою.

Що стосується простоти розробки інтерфейсу користувача, Apple представила SwiftUI. Цей інструмент дизайну інтерфейсу користувача, вбудований у Xcode 11, працює з iOS 13 (або новішої, як тільки вона з'являється). Завдяки цьому створювати власний інтерфейс користувача для всіх платформ Apple дуже просто. Ви можете зібрати свій інтерфейс користувача, написавши код або налаштувавши попередній перегляд. Зміни видно як у коді, так і в вікні попереднього перегляду одночасно.

2.3.1.3.3 Порівняння продуктивності Objective-C та Swift

Чим кращу та відповіднішу технологію вибирає розробник, тим вище продуктивність додатку. З вищою продуктивністю приходить кращий UX, простіший рефакторинг та обслуговування тощо. Apple стверджує, що Swift в 2,6 рази швидше, ніж Objective-C [14]. Це можливо завдяки тому, що Swift був створений як абсолютно нова мова з метою бути... швидкою. Незважаючи на те, що Apple оснастила Objective-C збирачем сміття, він все ще не такий ефективний, як добре написаний код

Swift. Простіший синтаксис і виконання перевірки типів під час компіляції допомагають Swift перевершити Objective-C. Для оптимізації управління пам’яттю Swift використовує ARC (автоматичний підрахунок посилань). Крім того, Swift підтримує динамічні бібліотеки, які також підвищують продуктивність програми.

2.3.1.3.4 Порівняння стабільності

Objective-C з нами вже більше 30 років. Остання випущена версія була 2.0, і вона існує з 2016 року. Стабільнішою вона не буде. Якщо є якісь помилки, ми це вже знаємо. Swift зараз лише 6 років, а Swift 5 був випущений рік тому, у березні 2019 року. Сьогодні ми можемо використовувати версію 5.2.4. Це показує, що Swift менш стабільний, ніж Objective-C, але це відбувається через постійні удосконалення та розвиток.

2.3.1.3.5 Порівняння документації

Важливість документації неможливо переоцінити. Чим краще ви знаєте мову та її інструменти, тим легше уникнути помилок. Обидві мови мають велику документацію. Objective-C головним чином тому, що вона існує вже понад 30 років. Її документацію можна знайти в архіві Apple. Проте документація для Swift постійно оновлюється. Ви знайдете більше нових джерел для цієї мови. Крім того, майте на увазі, що це відкритий вихідний код, тому є багато інших джерел, які ви можете знайти. Почніть із сервісу Apple для розробників та спеціальної веб-сторінки Swift. Також доступний репозиторій GitHub для Swift [14].

2.3.1.3.6 Порівняння прикладів додатків написаних з використанням Objective-C та Swift

Ще одним фактором, який можна взяти до уваги, є те, хто і як використовувати певну мову. Додатки iOS, створені до 2014 року, швидше за все, будуть написані на Objective-C, і головною причиною цього є просто відсутність Swift до цього року. Сьогодні досить часто компанії розглядають можливість переходу з Objective-C на Swift [14]. Тим часом не відомо прикладів, щоб хтось намагався зробити звотнє перетворення свого коду. Більше того, оскільки обидві мови можна використовувати в одній програмі, що дуже часто зустрічається в великих програмних продуктах. Наприклад, Uber використовує як Objective-C, так і Swift. Lyft, з іншого боку, повністю перейшов на Swift. Те саме з LinkedIn та їхніми додатками. Важко знайти компанію, яка б хотіла залишатися на Objective-C. І це не дивно.

2.3.1.3.7 Підсумки порівняння нативних засобів розробки під платформу IOS

Підсумовуючи усе порівняння можна зробити висновок, що з сучасним станом розвитку нативних мов програмування для платформи IOS, немає жодної необхідності у виборі мови програмування Objective-C. Мова програмування Swift є більш кращим вибором для розробки додатку під платформу IOS. Вона є продуктивнішою, простішою в розумінні, з більшою кількістю засобів для розробки і більшою спільнотою розробників [14].

2.3.2 Нативні засоби розробки Android додатків

2.3.2.1 Розробка Android додатків з використанням Java

Спочатку Java була офіційною мовою для розробки додатків для Android (але тепер її замінив Kotlin), і, отже, вона також є найбільш використовуваною мовою. Багато програм у Play Store створено на Java, і це також найбільш підтримувана мова Google. На додаток до всього цього, Java має чудову онлайн-спільноту для підтримки в разі будь-яких проблем. Однак Java є складною мовою для початківців, оскільки вона містить такі складні теми, як конструктори, винятки нульових покажчиків, паралельність, перевірені винятки тощо. Крім того, Android SDK підвищує складність на новий рівень. Загалом, Java — чудова мова, щоб відчути всі переваги розробки додатків для Android.

2.3.2.2 Розробка нативних Android додатків з використанням Kotlin

Kotlin — це мова програмування зі статичною типізацією із відкритим кодом, яка орієнтована на JVM, Android, JavaScript та Native. Він розроблений JetBrains. Він повністю сумісний і замінний з мовою програмування Java [17].

Kotlin SDK з’явився, коли розробці Android знадобилася більш сучасна мова, щоб додати до якостей Java і допомогти розробці мобільних пристроїв. JetBrains, люди, які створили Intellij, інтегроване середовище розробки, також створили Kotlin. Це статично типізована мова з відкритим вихідним кодом, заснована на віртуальній машині Java (JVM). Перевага

Kotlin полягає в тому, що ви можете компілювати його в JavaScript і взаємодіяти з Java. Це дозволяє розробникам не тільки легко оновлювати старі програми Java до Kotlin, але й продовжувати свою стару роботу на Java до Kotlin.

Переваги Kotlin для розробників Android, безсумнівно, це майбутнє розвитку. Багато компаній використовують Kotlin як офіційну мову програмування на Android. Майбутнє Kotlin як нової мови Android, безумовно, яскраве.

Порівнюючи мови нативної розробки для платформи Android. Перевагу має Kotlin, оскільки це новіша мова програмування, що прийнята корпорацією Google, розробниками операційної системи, основною мовою програмування для цієї платформи [16].

2.4 Кросплатформенні засоби мобільної розробки

Кросплатформна мобільна розробка – це ще один тип розробки додатків, який дозволяє продукту працювати на кількох мобільних операційних системах і бути написаним однією мовою програмування. Коли код для програми готовий, він проходить через проміжне програмне забезпечення, яке перекладає його на власні API для платформи iOS або Android [18].

Переваги даного підходу:

- Швидкий розвиток за короткий проміжок часу
- Економічна ефективність
- Багаторазовий код

- Ідеальна бізнес-можливість залучити велику частку ринку мобільних додатків

Цей тип мобільної розробки допоможе вам не упустити шанс завоювати велику частку ринку мобільних пристроїв за допомогою програми, яка підходить для всіх основних операційних систем. Замовники заощадять свої гроші, оскільки розробникам потрібна лише одна кодова база для запуску програми на кількох ОС. Використовуючи кросплатформну розробку, з'являється вигідна можливість створити додаток, який буде привабливим для користувачів Android, iOS та інших ОС, не витрачаючи при цьому багато грошей.

Недоліки:

- Обмежений доступ до рідних функцій
- Проблеми з користувацьким досвідом

Існує багато кросплатформних інструментів розробки мобільних додатків, зокрема Flutter, Xamarin, React Native. Правильний вибір залежить від ваших потреб, вимог і проблем.

2.5 Вибір між кросплатформленими та нативними засобами розробки додатків для мобільних платформ

Виходячи з усього вищесказаного можна підсумувати, 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. Однак стартапи стурбовані високою вартістю розробки нативних додатків, оскільки їм потрібно розробляти обидві платформи

паралельно. Щодо кросплатформлених засобів, хоча вони і економлять час і кошти, замовники ризикують пожертвувати якістю самого додатку. Кросплатформенні додатки важко налаштувати так, щоб вони одинаково оптимально працювали на різних платформах, і під час роботи додатку потрібний додатковий рівень абстракції, що призводить до зниження продуктивності. Стартапам піде на користь скорочення часу та витрат на таку розробку. Однак потрібно мати на увазі, що може бути складніше розширювати, підтримувати та налаштовувати додаток для роботи з функціоналом, що виходить за межі певного фреймворку.

3 СУЧАСНИЙ СТАН ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ НА МОБІЛЬНИХ ПЛАТФОРМАХ

Дивлячись на велетенську популярність смартфонів та мобільних додатків в цілому, не дивно, що така гнучка та стрімко розвиваючася технологія як машинне навчання не змогла обійти стороною розробку для мобільних платформ. Станом на сьогодні машине навчання використовуються в вирішенні різноманітних завдань при розробці мобільних додатків, зокрема: підбирає рекомендації контенту на основі дій користувача, здійснює обробку відео в реальному часі, проводить визначення об’єктів за допомогою їх зображень та навіть звуків, працює за суперника людям в різноманітних іграх.

Рисунок 3.1 — Засоби машинного навчання на мобільних платформах

Поле розробки додатків з використанням AI та машинного навчання отримує визнання за постійне, залежне від даних навчання. Даний підхід використовує аналітику в реальному часі, щоб виявити поведінку, аномалії, відмінності та закономірності. ML має значний вплив на розвиток мобільних додатків 21-го століття. Перехід від комп'ютерних програм на базі машинного навчання до програм для смартфонів є дуже швидким і високопродуктивним.

Маючи це на увазі, програми машинного навчання оновлюють використання смартфонів за допомогою таких ефективних підходів:

1. Сучасні мобільні користувачі очікують простоти, зручності, радості та функціональності. За допомогою програм з використанням машинного навчання вони записують хобі, передбачають бажання, керують календарями, повідомляють про події та рекомендують рішення.
2. Невгамовні шукачі хочуть швидких, релевантних та інформативних результатів. Інструменти ML ефективні для відстеження типових дій і зберігання історичних даних. Ці інструменти виправляють орфографію, відповідають на голосовий пошук і пропонують списки пов'язаних результатів.
3. ML-commerce — це благо для тих, хто хоче купувати та продавати на ходу. Алгоритми ML покращують клієнтів і рекомендують найкращі продукти. Вони допомагають у цифровому маркетингу за допомогою аналізу даних, реклами в реальному часі та розпізнавання шаблонів покупок для компаній.
4. Лідери бізнесу очікують, що додатки мають найпередовіші та найточніші функції. Алгоритми ML допомагають з метриками, аналізом і

збором інформації. Вони забезпечують точність даних, прийняття рішень, доставку продуктів, персоналізовані послуги та безпечність підключення.

5. Користувацький досвід у додатків написаних з використанням ML є плавним та інтуїтивно зрозумілим. Їх практичний підхід підходить для високошвидкісних розумних пристроїв. Машинне навчання в мобільних додатках допомагає розробникам створювати різноманітні потужні програми за допомогою фільтрації даних, навчання алгоритмів, вибору моделі, налаштування параметрів і аналітичних передбачень, зроблених тренованою моделлю.

ML допомагає зменшити розрив між розумінням поведінки користувачів і використанням її для створення індивідуального рішення. Розробники програм вбудовують ML у мобільні додатки, щоб створювати індивідуальні програми для кожного окремого користувача. ML залежить від безперервного навчання. На основі повсякденної діяльності користувача програма ML вчиться і заново навчається, щоб створити індивідуальне рішення. Це покращення допомагає створити адаптивну програму, яка може допомогти досягти унікального персоніфікованого користувацького досвіду. ML допомагає в прогнозному аналізі. Ця технологія допомагає програмі обробляти величезну кількість даних і отримувати кількісні прогнози, які можна налаштувати відповідно до вимог користувача. Розробники можуть навчати модулі машинного навчання фільтрувати спам і потенційно небезпечні сайти або електронні листи. Що сприяє збільшенню ролі засобів машинного навчання в секторі підвищення безпеки додатків. Розпізнавання символів і НЛП, у поєднанні з прогнозним аналізом, допоможуть створювати програми зі здатністю читати й розуміти мову. Це є важливою віхою в додатках на базі ML, і це допомагає створити низку різних програм для різних ніш.

Використання засобів машинного навчання для мобільних платформ полягає в тренуванні моделі і її подальшому використані в додатку на платформі IOS чи Android.

3.1 Фреймворки, що використовуються при розробці мобільних додатків засобами машинного навчання

3.1.1 Tensorflow розроблений компанією Google

TensorFlow Lite — це кросплатформний фреймворк глибокого навчання з відкритим вихідним кодом, готовий до продукту, який перетворює попередньо навчену модель у TensorFlow у спеціальний формат, який можна оптимізувати для швидкості або зберігання.

Модель спеціального формату може бути розгорнута на периферійних пристроях, таких як мобільні телефони, які використовують вбудовані пристрої на базі Android або iOS або Linux, такі як Raspberry Pi або мікроконтролери.

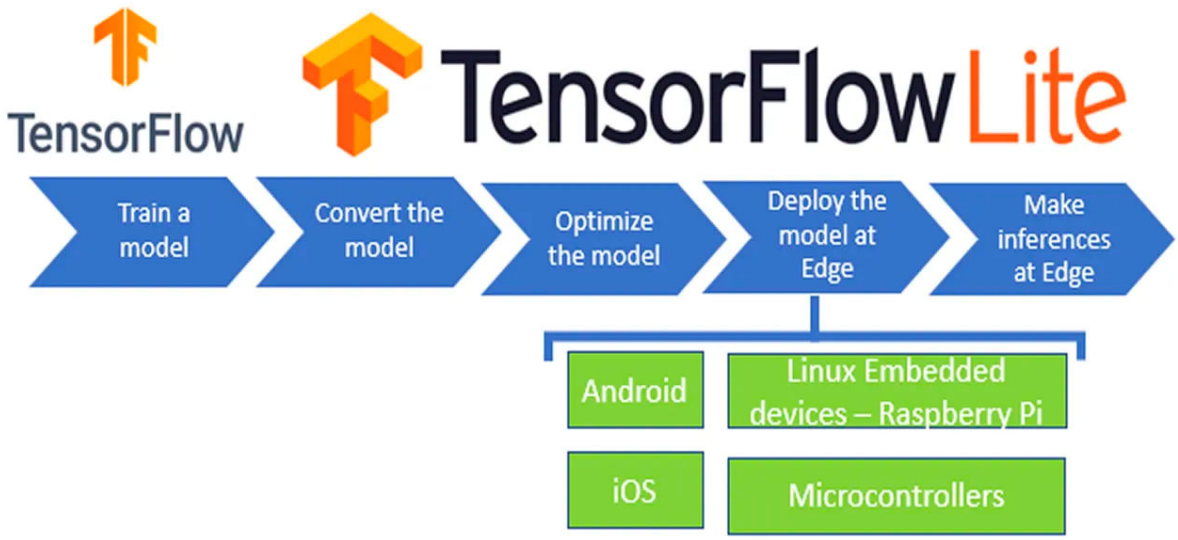

Рисунок 3.2 — Схема роботи з фреймворком TensorFlow Lite

Для того, щоб тренована модель працювала на мобільних девайсах її потрібно перетворити на модель, що підтримується фреймворком TensorFlow Lite. Модель TF lite – це модель спеціального формату, ефективна з точки зору точності, а також значно легша версія, яка буде займати менше місця. Ці функції роблять моделі TF Lite ідеальними для роботи на мобільних і вбудованих пристроях.

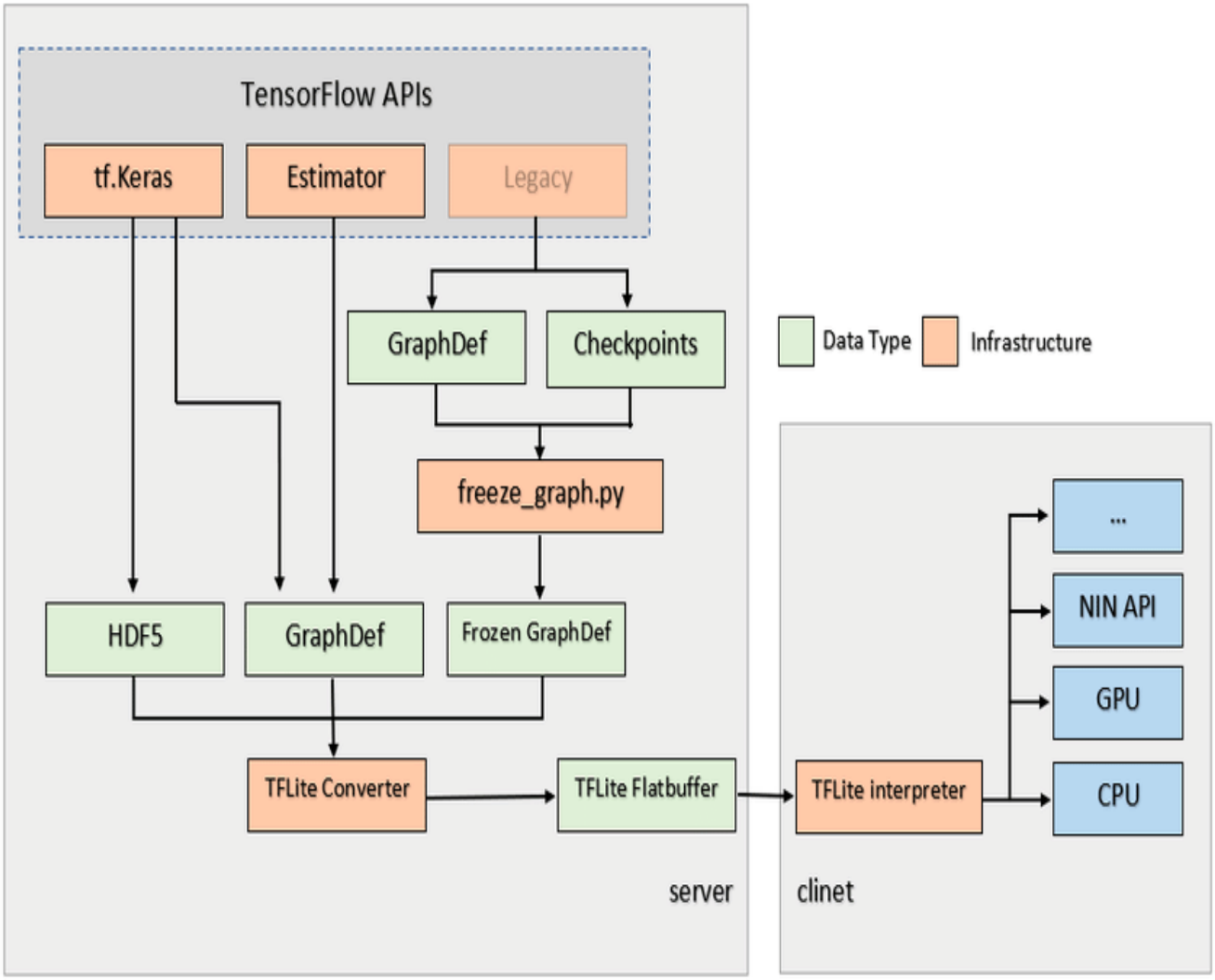

Рисунок 3.3 — Перетворення у модель підтримувану TensorFlow Lite

Під час процесу перетворення з моделі Tensorflow на модель Tensorflow Lite розмір файлу зменшується. У нас є вибір або піти на подальше зменшення розміру файлу за допомогою компромісу зі швидкістю виконання моделі. Tensorflow Lite Converter перетворює модель Tensorflow у файл плоского буфера TensorFlow Lite (.tflite). Плоский буферний файл Tensorflow Lite розгортається на клієнті, яким у наших випадках може бути мобільний пристрій під керуванням iOS або Android або вбудований пристрій. Як ми можемо перетворити модель TensorFlow

на модель TFlite? Після того, як натренували модель, потрібно її зберегти. Збережена модель серіалізує архітектуру моделі, ваги та зміщення, а також навчальну конфігурацію в одному файлі. Збережену модель можна легко використовувати для спільного використання або розгортання моделей.

Також необхідно оптимізувати модель. Це можна зробити виконавши квантування та брункування ваг моделі.

Коли ми зберігаємо модель TensorFlow, вона зберігається як графіки, що містять обчислювальну операцію, функції активації, ваги та зміщення. Функція активації, ваги та зміщення є 32-бітовими числами з плаваючими комами. Квантування знижує точність чисел, які використовуються для представлення різних параметрів моделі TensorFlow, і це робить моделі легшими.

Так само, як ми обрізаємо рослини, щоб видалити непродуктивні частини рослини, щоб зробити їх більш плодоносними та здоровими, так само ми можемо обрізати ваги моделі.

Зменшення ваги обрізає параметри всередині моделі, що має дуже незначний вплив на продуктивність моделі.

3.1.2 Core ML розроблений компанією Apple

Core ML — це платформа машинного навчання, яка використовується в продуктах Apple, таких як QuickType, Siri та Camera. Apple випустила його на WWDC 2017 для інтелектуальної розробки додатків для iOS [21]. За допомогою цієї платформи ML розробники можуть вбудувати функції машинного навчання комп’ютерного зору в програми iOS (тобто зробити програми здатними виконувати завдання, які виконують людські очі). Деякі з підтримуваних функцій включають

відстеження об’єктів, виявлення обличчя, визначення тексту, відстеження обличчя, виявлення штрих-коду тощо. Крім того, Core ML пропонує API для обробки природних мов, а також інші засоби машинного навчання, щоб досконало розуміти текст. Для цього Core ML використовує лексемизацію, ідентифікацію мови, частину мови, лемматизацію та пов’язані з нею ознаки. Розробники можуть вибирати з безлічі доступних моделей ML, таких як MobileNet, Squeezenet, Places205-GoogLeNet тощо. Ці моделі допомагають розрізняти тип завдання, яке потрібно виконати за допомогою ML. Ці готові до використання моделі ML можна використовувати або розробники можуть використовувати інструменти Core ML для перетворення моделі в основну модель ML. Щоб розпочати роботу з цим фреймворком, Apple надає розробникам детальну документацію.

Core ML використовується щоб інтегрувати засоби машинного навчання у додаток для платформи IOS. Додаток використовуватиме Core ML API і дані користувача для прогнозування, а також для навчання чи точного налаштування моделей і все ці операції будуть виконуватися на пристрої користувача.

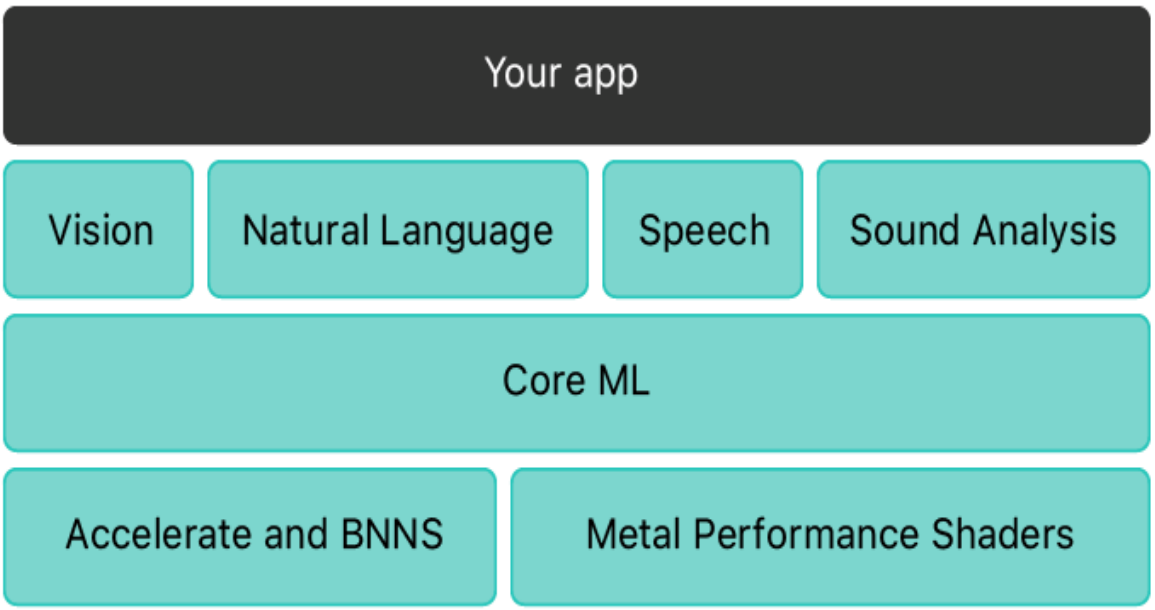

Рисунок 3.3 — Можливості використання Core ML

Модель – це результат застосування алгоритму машинного навчання до набору навчальних даних. Ви використовуєте модель, щоб робити прогнози на основі нових вхідних даних. Моделі можуть виконувати широкий спектр завдань, які було б важко або непрактично написати в коді. Наприклад, ви можете навчити модель класифікувати фотографії або виявляти певні об’єкти на фотографії безпосередньо з її пікселів [22].

Ви можете створювати та навчати модель за допомогою програми Create ML, яка постачається з Xcode. Моделі, навчені за допомогою Create ML, мають формат моделі Core ML і готові до використання у вашій програмі. Крім того, ви можете використовувати широкий спектр інших бібліотек машинного навчання, а потім використовувати інструменти Core ML для перетворення моделі у формат Core ML. Як тільки модель буде на пристрої користувача, ви можете використовувати Core ML, щоб перенавчати або налаштовувати її на пристрої з даними цього користувача.

Core ML оптимізує продуктивність на пристрої, використовуючи процесор, графічний процесор і нейронний механізм, мінімізуючи при цьому обсяг пам’яті та енергоспоживання. Виконання моделі суворо на пристрої користувача усуває будь-яку потребу в мережевому з’єднанні, що допомагає зберегти конфіденційність даних користувача і реагувати ваш додаток.

Core ML є основою для специфічних для домену фреймворків і функціональних можливостей. Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення аудіо в текст і Sound Analysis для визначення звуків у аудіо. Сам Core ML будується на основі низькорівневих примітивів, таких як Accelerate і BNNS, а також шейдерів Metal Performance.

3.1.3 Amazon Machine Learning

Служба Amazon Machine Learning гарантує, що ML буде впроваджено в мобільні додатки розробниками всіх рівнів кваліфікації. Завдяки цій платформі від Amazon розробники отримують інструменти візуалізації та майстри, які дозволяють розробникам створювати моделі машинного навчання без складних алгоритмів або технологій. Коли моделі ML будуть готові, API можна використовувати для отримання прогнозів з моделі. Додаткового коду для отримання прогнозів не потрібно. Не потрібно також особливе управління інфраструктурою.

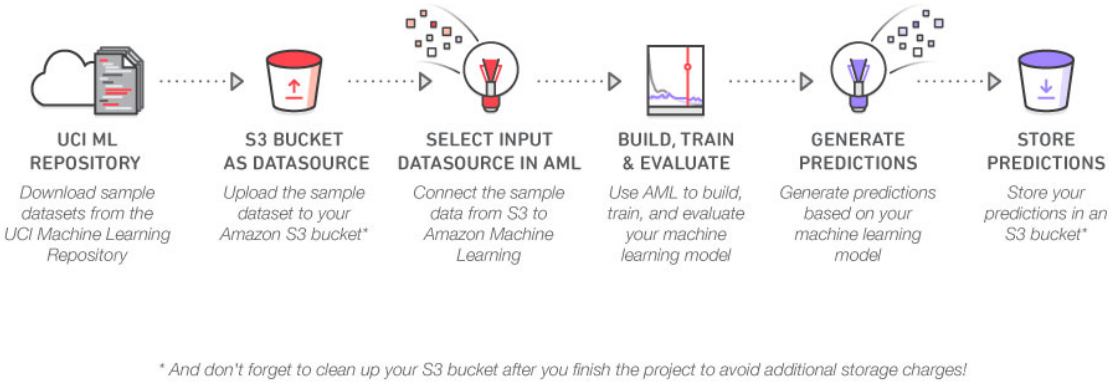

Рисунок 3.4 — схема роботи з AWS ML

З AML немає ніяких витрат на апаратне чи програмне забезпечення. Щоб отримати допомогу, розробникам надаються посібники для платформи, щоб розпочати роботу, які інформують їх про необхідні умови, створення клієнтів, приклади пропозицій тощо [23].

4 ПРОЕКТУВАННЯ ТЕСТОВОГО ДОДАТКУ ДЛЯ ДЕМОНСТРАЦІЇ МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ ПРОГРАМУВАННЯ ПІД МОБІЛЬНІ ПЛАТФОРМИ

4.1 Вибір тематики додатку

Для демонстрації засобів машинного навчання було обрано реалізувати додаток, що може прогнозувати, що зображено на зображені, за допомогою тренованої моделі машинного навчання для платформи IOS. Даний додаток зможе допомогти людям з поганим зором для того, щоб розглядати більш дрібні предмети, чи може використовуватися для того, щоб навчати дітей назвам предметів тощо. У даного додатку є доволі великий спектр можливого використання.

4.2 Аналіз предметної області

У наш час, засоби машинного навчання змінюють усі сфери людського життя. За допомогою машинного навчання, ми можемо керувати нашим телефоном за допомогою голосу (використовуючи технологію Siri), ми можемо керувати розумним будинком за допомогою команд, ми можемо розпізнавати що зображено на зображеннях просто навівши на це зображення камеру нашого телефону, та багато іншого. Також зараз існує багато систем, що допомагають людям керувати власними фінансами та піклуватися про своє здоров’я. Засоби машинного

навчання роблять дуже точні прогнози можливого стресу чи втоми на основі показників наших фітнес браслетів. За допомогою засобів машинного навчання сервіси, що ми використовуємо в повсякденному житті, підбирають для нас рекомендації новин, музики чи відео.

Залежно від того, що потрібно зробити алгоритму машинного навчання існують різні задачі машинного навчання. Наприклад, класифікація зображень відносно заданого параметру, розпізнавання об’єктів, що зображено на малюнках, розпізнавання тексту, зокрема рукописного почерку, класифікація різноманітних даних, що підходять за тими чи іншими параметрами, розпізнавання об’єктів в реальному часі, підведення статистики засобами машинного навчання, синтезування мови, спроби розробки штучного інтелекту, як моделі, що здатна буде приймати рішення та перенавчатися, без втручання людини, та багато інших завдань, що продовжують з’являтися і зараз. У рамках даної магістерської роботи було розглянуто детальніше саме процес визначення ймовірностей присутності об’єктів на зображеннях за допомогою нетренованої моделі.

Для базового розуміння способів вирішення проблем знаходження елементів на зображенні і як наслідок прогнозуванні того, що знаходиться на них необхідно не тільки зосередити власну увагу на класифікації зображень за характеристиками, а також взяти до уваги їх позицію та величину. Це завдання в наукових колах ставиться як завдання розпізнавання об’єктів. Розпізнавання об’єктів включає також такі типи підзадач, як розпізнавання облич, розпізнавання поз, розпізнавання руху. Як одна із фундаментальних проблем комп’ютерного зору, виявлення об’єктів здатне забезпечити цінну інформація для семантичного розуміння зображень і відео, і пов’язано з багатьма програмами, в тому числі з аналізом поведінки людини, розпізнаванням облич та автономним

водінням. Тим часом, успадкувавши нейронні мережі та пов’язані з ними системи навчання, прогрес у цих галузях допоможе і у вирішені основної задачі, а також матиме великий вплив на об'єктні методи виявлення, які можна вважати навчанням системи. Однак через великі розбіжності в точках зору, пози, оклюзії та умов освітлення, важко досягти ідеального виявлення об’єктів за допомогою додаткового завдання локалізації об’єкта. Визначення проблеми виявлення об'єкта полягає у визначенні де розташовані об'єкти на заданому зображенні (локалізація об'єкта) і до якої категорії належить кожен об’єкт (класифікація об’єктів). Отже, послідовність створення моделей виявлення об'єктів можна в основному розділити на три етапи: вибір інформативного регіону, виділення ознак та класифікація зображень.

Вибір інформативного регіону. Оскільки різні об’єкти можуть з’являтися в будь-яких позиціях зображення та мати різні співвідношення сторін або розміри, природним вибором є сканування всього зображення за допомогою багатомасштабного розсувного вікна. Хоча ця вичерпна стратегія може з'ясувати всі можливі положення об'єктів, її недоліки також очевидні. Через велику кількість вікон-кандидатів вона є дорогою з точки зору обчислень і створює занадто багато надлишкових вікон. Однак, якщо застосовано лише фіксовану кількість шаблонів розсувних вікон, можуть виникнути незадовільні області.

Виділення ознак. Щоб розпізнати різні об’єкти, нам потрібно витягти візуальні ознаки, які можуть забезпечити семантичне та надійне уявлення. Характерними ознаками є масштабно-інваріантне перетворення ознак та признаками Хаара. Це пов’язано з тим, що ці особливості можуть створювати уявлення, пов’язані зі складними клітинами в мозку людини. Однак через різноманітність зовнішніх виглядів, умов освітлення та фону

важко вручну розробити надійний дескриптор функцій, щоб ідеально описувати всі види об’єктів.

Класифікація зображень. Крім того, потрібен класифікатор, щоб виділити цільовий об’єкт з усіх інших категорій і зробити уявлення більш ієрархічними, семантичними та інформативними для візуального розпізнавання.

Класифікація зображень передбачає присвоєння зображенню мітки класу, тоді як локалізація об’єкта передбачає малювання обмежувальної рамки навколо одного або кількох об’єктів на зображенні. Виявлення об’єктів є складнішим і поєднує ці дві задачі та малює обмежувальну рамку навколо кожного об’єкта, який цікавить, на зображенні та призначає їм мітку класу. Разом усі ці проблеми називаються розпізнаванням об’єктів.

Розпізнавання об’єктів – це сукупність пов’язаних завдань для ідентифікації об’єктів на цифрових фотографіях. Згорткові нейронні мережі на основі регіонів, або R-CNN, — це сімейство методів для вирішення завдань локалізації та розпізнавання об’єктів, розроблених для продуктивності моделі [24]. You Only Look Once, або YOLO, — це друге сімейство методів розпізнавання об’єктів, розроблених для швидкого використання в режимі реального часу. Inception V3 — це ще одна архітектура запропонована інженерами Google, це хороша модель згорткованої нейроної мережі для комп’ютерного бачення і був розроблений для апроксимації та покриття оптимальної локальної розрідженої структури мережі згорткового бачення через доступні локально щільні компоненти.

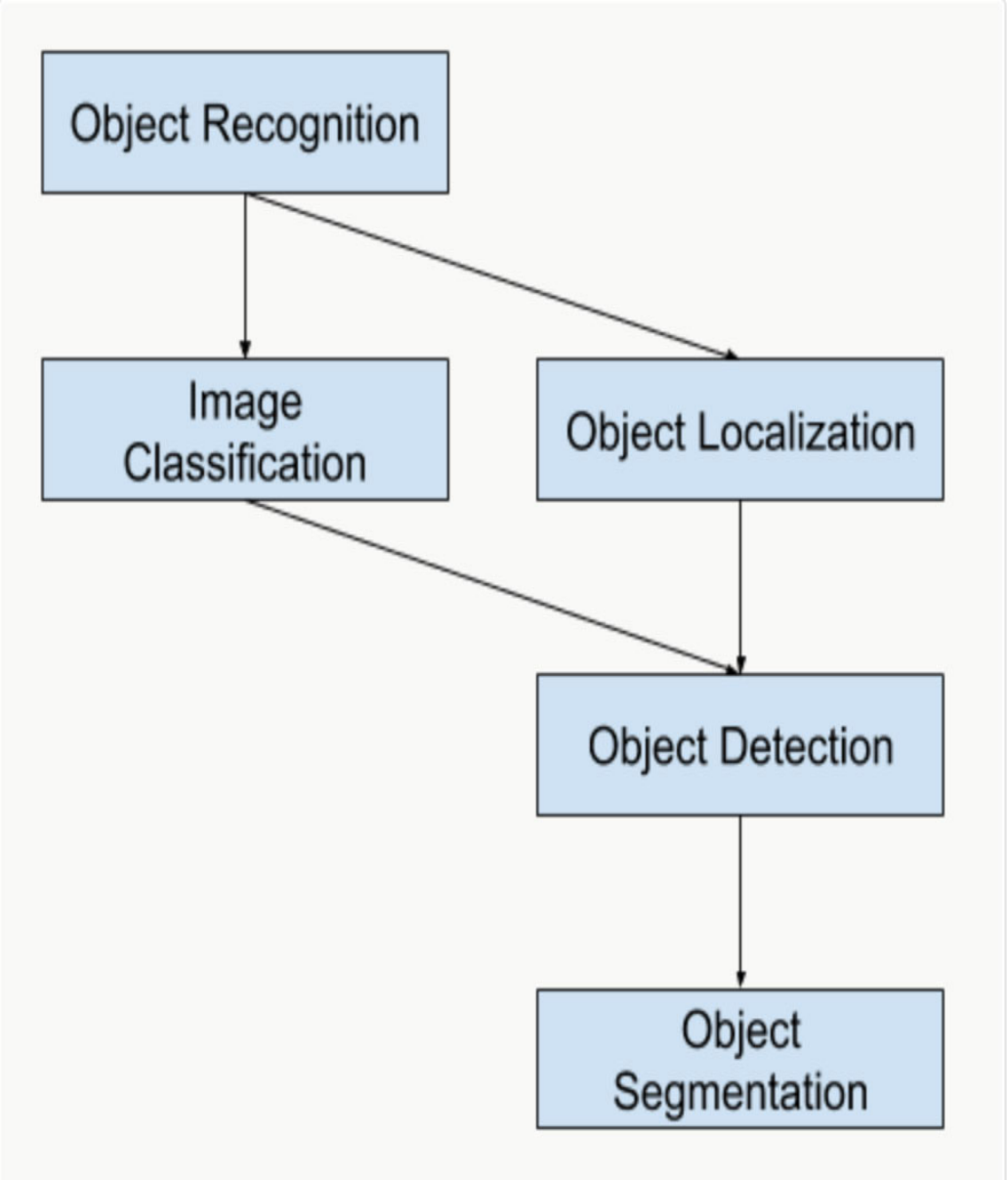

Рисунок 4.1 — Огляд завдань комп’ютерного зору розпізнавання об’єктів

Більшість останніх інновацій у проблемах розпізнавання зображень з'явилися в рамках участі в завданнях ILSVRC [24]. Це щорічний академічний конкурс з окремим завданням для кожного з цих трьох типів проблем, з метою сприяння незалежним і окремим покращенням на кожному рівні, які можна використовувати ширше. Наприклад,

перегляньте список трьох відповідних типів завдань нижче, взятий з оглядового документа ILSVRC 2015 року:

1. Класифікація зображень. Алгоритми створюють список категорій об'єктів, присутніх на зображенні.
2. Локалізація окремого об’єкта: алгоритми створюють список категорій об’єктів, присутніх на зображенні, разом із вирівняною по осі обмежувальною рамкою, що вказує положення та масштаб одного екземпляра кожної категорії об’єктів.
3. Виявлення об’єктів. Алгоритми створюють список категорій об’єктів, присутніх на зображенні, разом із вирівняною по осі обмежувальною рамкою, яка вказує положення та масштаб кожного екземпляра кожної категорії об’єктів.

Ми бачимо, що «Локалізація одного об’єкта» є простішою версією ширшого визначення «Локалізація об’єкта», яка обмежує завдання локалізації об’єктами одного типу в зображенні, що ми можемо припустити, що є легшим завданням.

Нижче наведено приклад порівняння локалізації окремого об’єкта та виявлення об’єкта, взятого з статті ILSVRC [24]. Зверніть увагу на різницю в основних очікуваннях істини в кожному випадку.

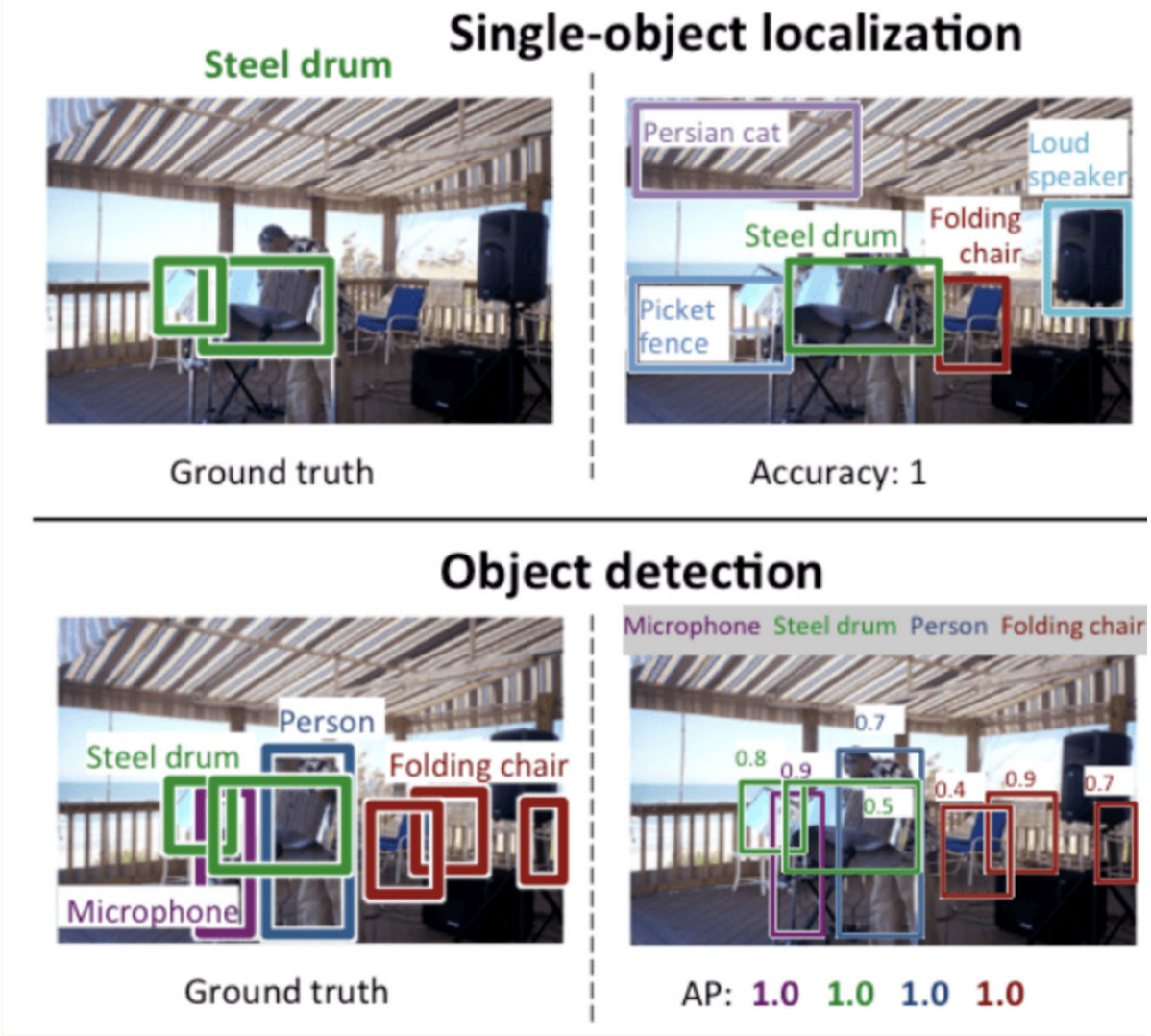

Рисунок 4.2 — Приклад порівняння локалізації та виявлення об’єкта

Ефективність моделі для класифікації зображень оцінюється за допомогою середньої помилки класифікації для передбачених міток класів. Ефективність моделі для локалізації окремого об’єкта оцінюється за допомогою відстані між очікуваним і прогнозованим обмежуючим квадратом для очікуваного класу. У той час як продуктивність моделі для розпізнавання об’єктів оцінюється за допомогою точності та запам’ятовування кожного з найкращих відповідних обмежувальних рамок для відомих об’єктів на зображенні.

Тепер, коли ми знайомі з проблемою локалізації та виявлення об’єктів, давайте подивимося на кілька останніх найефективніших моделей глибокого навчання.

4.3 Модель Inception V3

Inception v3 — це згорткова нейронна мережа для допомоги в аналізі зображень і виявленні об’єктів, яка розпочалася як модуль для Googlenet. Це третє видання початкової згорткової нейронної мережі Google, спочатку представленої під час конкурсу ImageNet Recognition Challenge. Дизайн Inceptionv3 мав на меті дозволити більш глибокі мережі, а також не допускати збільшення кількості параметрів: він має «менше 25 мільйонів параметрів», порівняно з 60 мільйонами для AlexNet.

Так само, як ImageNet можна розглядати як базу даних класифікованих візуальних об’єктів, Inception допомагає класифікувати об’єкти у світі комп’ютерного зору. Архітектура Inceptionv3 була повторно використана в багатьох різних програмах, часто використовуваних «попередньо навчених» від ImageNet. Одним із таких застосувань є наука про життя, де він допомагає у дослідженні лейкемії.

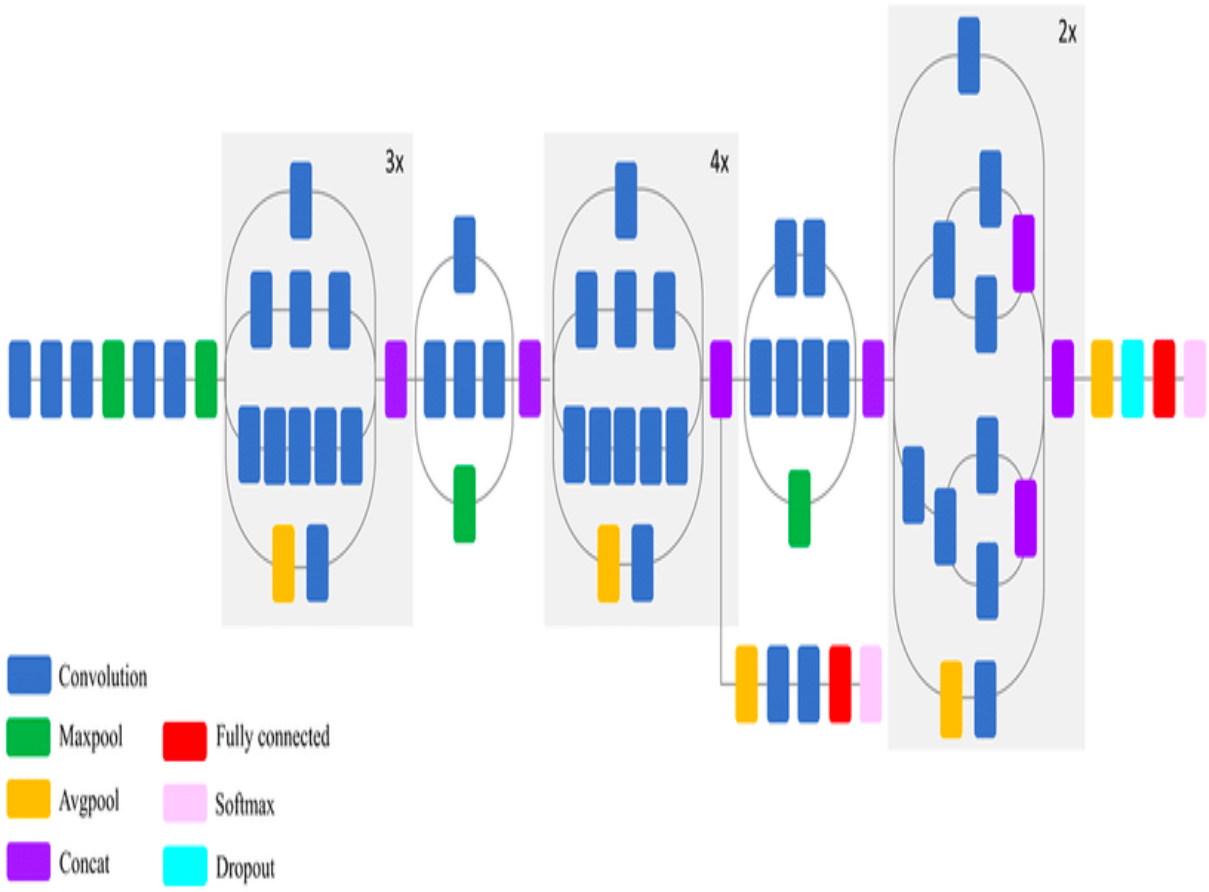

Рисунок 4.6 — Схематична діаграма роботи моделі Inception V3

Оригінальна назва (Inception) отримала кодову назву після того, як популярний інтернет-мем "we need to go deeper" став вірусним, цитуючи фразу з фільму Inception Крістофера Нолана.

Саме цю модель буде використано для розробки додатку, через простоту у використанні, підтримкою Google та хороший відсоток точності при її використанні.

4.6 Виявлення акторів системи та формування варіантів використання

Даний мобільний додаток спроектовано для користування однією людиною. Система буде другим актором додатку.

Користувач даної системи матиме можливість обрати зображення для аналізу, та переглянути результати. Варіант використання вибір зображення включатиме в себе такі під варіанти: зняти зображення з використанням камери, вибрати зображення з бібліотеки зображень чи зображень попередньо зроблених камерою, а також дістати зображення з сайту, що забезпечує зображення Lorem Picsum.

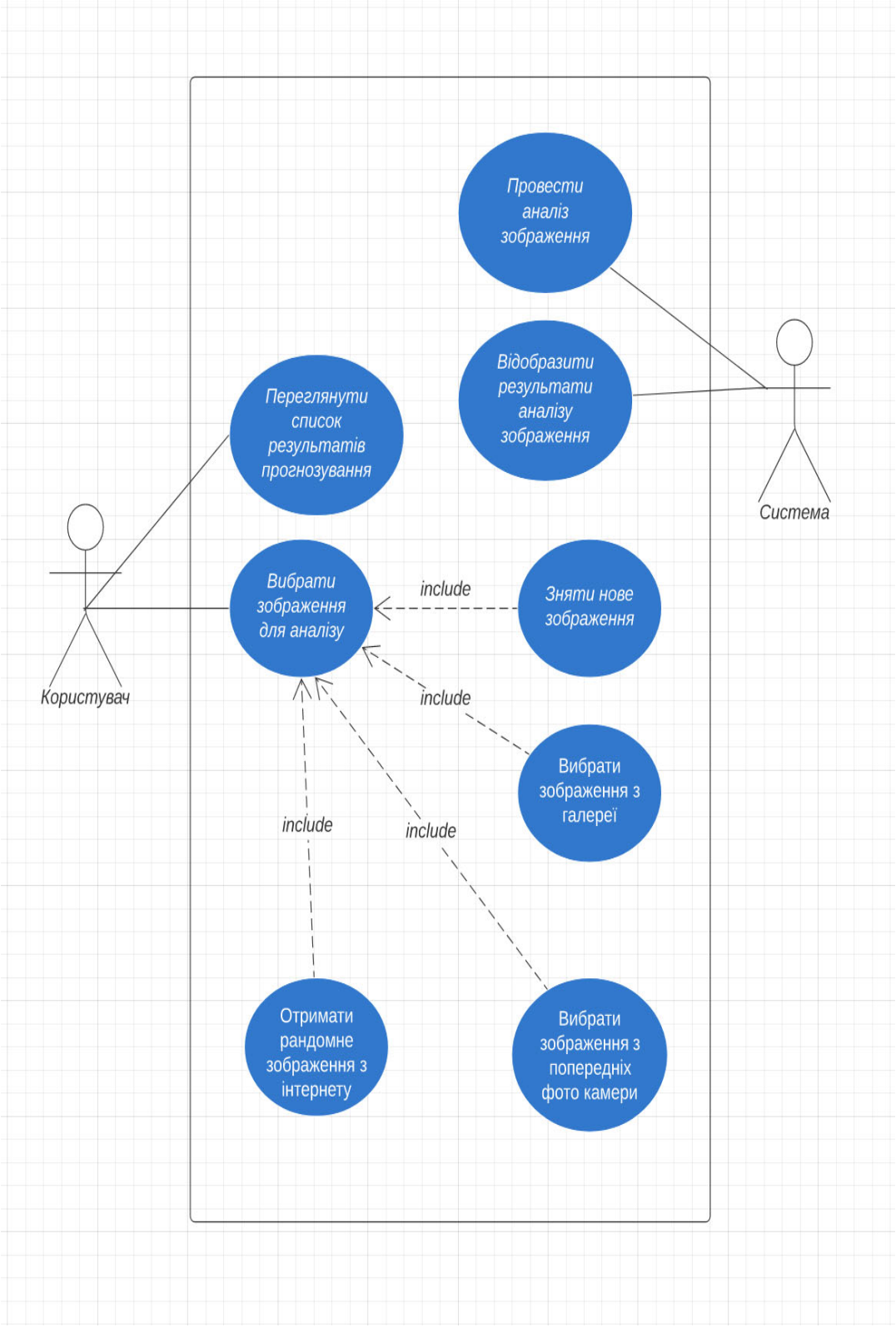

Рисунок 4.7 — ULM діаграма варіантів використання проектованого додатку

4.7 Вибір архітектури додатку

Вибір правильної архітектури iOS додатку є важливим для створення високофункціонального додатка для iPhone. Архітектура програмного забезпечення зосереджується на тому, як основні елементи та компоненти в додатку iPhone використовуються та взаємодіють з іншими ключовими компонентами та елементами. Це також допомагає керувати алгоритмом та деталями реалізації кожного компонента.

Вибір архітектури залежить від складності вашої програми. Крім того, складність програми залежить від ряду функцій, інтегрованих у програму. Кожна функція містить набір компонентів, класу, сутності та елементів.

Без відповідної архітектури програмного забезпечення досить важко керувати та оновлювати код програми на ходу. Особливо, коли розробникам додатків для iPhone потрібно виправити будь-яку помилку або вирішити вимоги щодо покращення функцій

Для реалізації IOS додатку найпопулярнішими архітектурами є: MVP, MVC та MVVM.

4.7.1 MVC

Шаблон проектування Model-View-Controller (MVC) призначає об'єктам у програмі одну з трьох ролей: модель, подання або контролер. Шаблон визначає не тільки ролі, які відіграють об’єкти в додатку, він визначає спосіб, яким об’єкти спілкуються один з одним. Кожен з трьох типів об’єктів відокремлений від інших абстрактними кордонами і

взаємодіє з об’єктами інших типів через ці межі. Колекція об’єктів певного типу MVC у програмі іноді називають шаром — наприклад, шаром моделі [26].

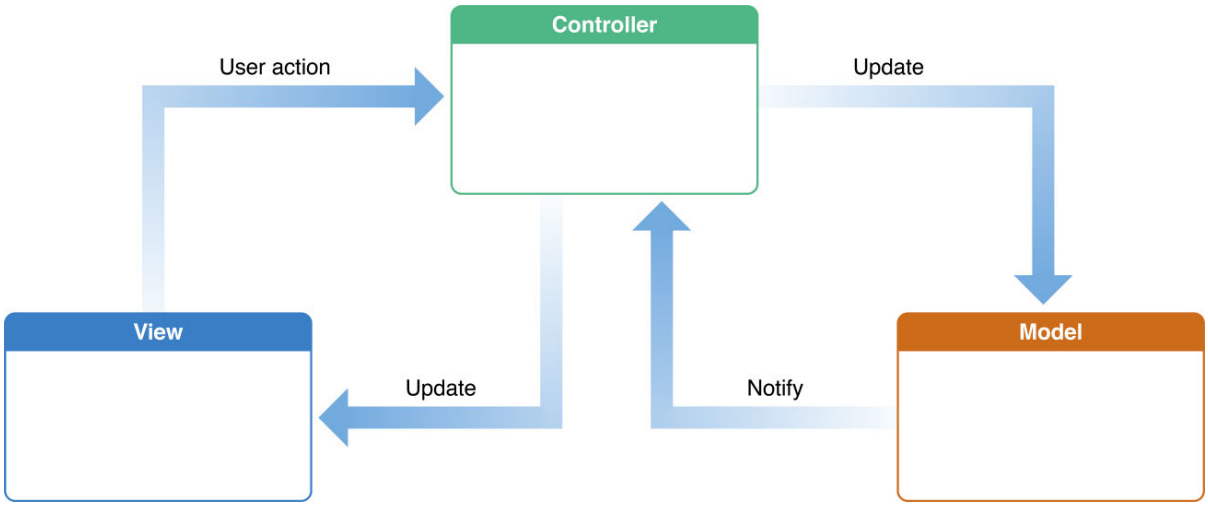

Рисунок 4.8 — Apple MVC

З початку розробки для IPhone це був архітектурний підхід запропонований Apple. Дана архітектура є простою у використанні, але при розширенні додатків може виникнути проблема нагромаджень коду, що підвищує складність у відлаштовугоджувані і збільшується кількість помилок.

4.7.2 MVVM

Model-View-ViewModel (MVVM) — це структурний шаблон програмування, що розділяє код на моделі, вигляди та моделі виглядів. У даному шаблоні моделі зберігають дані додатку. Вигляди відповідають лише за відображення даних і є зазвичай нащадками класу UIView в IOS інфраструктурі. Моделі виглядів є проміжним шаром між класами моделей

та класами виглядів і відповідають за перетворення даних у форму потрібну користувацькому інтерфейсу [27].

Даний шаблон необхідно використовувати при розробці великих додатків, оскільки він є складнішим за MVC чи MVP. Також даний архітектурний підхід дозволяє відносно легке тестування додатку, але потребує значно більшого часу для розробки.

4.7.3 MVP

Шаблон Modle-View-Presenter (MVP) є похідним від архітектурного шаблону Model-View-Controller (MVC) і використовується в основному для створення інтерфейсів користувача. У MVP презентер бере на себе функціональність посередника. У MVP вся логіка попереднього налаштування передається саме презентеру [27].

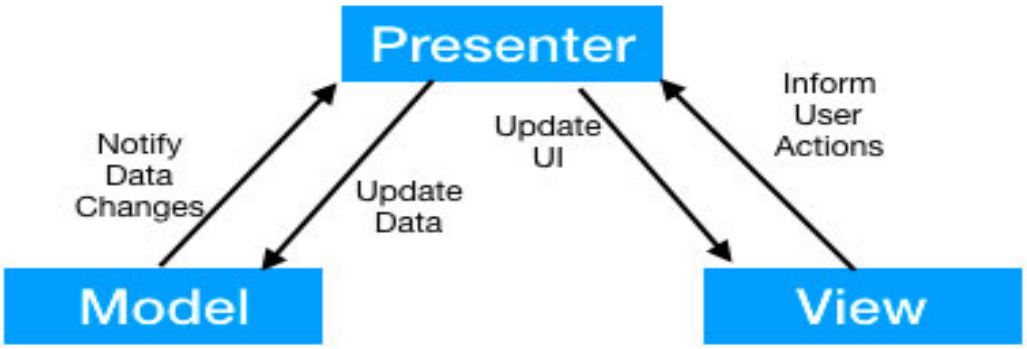

Рисунок 4.10 — MVP

Даний архітектурний підхід має такі переваги:

1. користувацький інтерфейс не взаємодіє безпосередньо з моделями даних, що ізолює реалізацію користувацького інтерфейсу та реалізацію взаємодії з даними;
2. є можливість змінювати користувацького інтерфейсу без змін у шарі взаємодії даних;
3. спрощує підтримку додатку.

4.8 Постановка вимог

Для перевіряння якості програмного продукту необхідно перевірити чи він відповідає поставленим вимогам. Саме тому на етапі проектування було виділено функціональні та нефункціональні вимоги.

Функціональними вимогами для додатку розпізнавання об’єктів є:

1. даний додаток повинний забезпечувати безпеку користувача і не поширювати особисті дані ні з яким іншим сервісом;
2. додаток повинний виконувати усі функції, що були перераховані у варіантах використання системи.

Нефункціональними вимогами системи є:

1. час відгуку на будь яку команду користувача не повиний перевищувати 0.5 секунди;
2. додаток повинний не вилітати;
3. даний додаток повинен запускатися на усіх IPhone з встановленою версією IOS 13, або вище.

5 РОЗРОБКА ТА ТЕСТУВАННЯ ДОДАТКУ

Розробка додатку була проведена використовуючи Xcode 13, а відлагодження менеджменту пам’яті було проведено за допомогою Xcode Leak Manager. Дані програмні продукти можна завантажити скориставшись офіційним сайтом для розробників від Apple.

Для тестування даного додатку використовувався девайс Apple IPhone 11 з встановленою версію IOS 15.1.

Після запуску програми відкривається основне вікно роботи програми, де зверху екрану ми можемо побачити вікно зображення, нижче - таблицю результатів передбачення можливості наявності об’єктів на ньому, а ще нижче кнопку для зміни зображення.

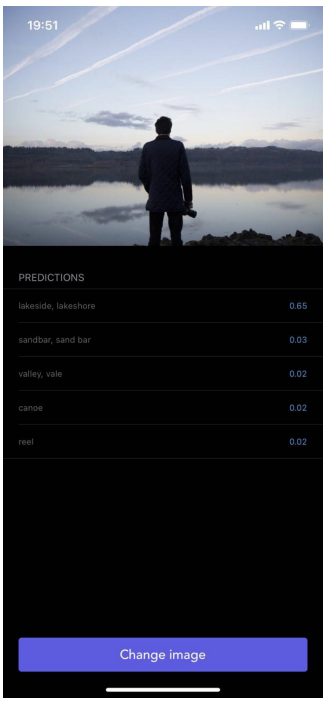

Рисунок 5.1 — Основне вікно програми

Після натиснення на кнопку зміни фото відкривається вікно вибору джерела зображення, де можна вибрати одну з 5 опцій: “Зняти фото”, “Вибрати з зображень зроблених камерою”, “Вибрати з галереї”, “Додати рандомне фото”, “Скасувати”.

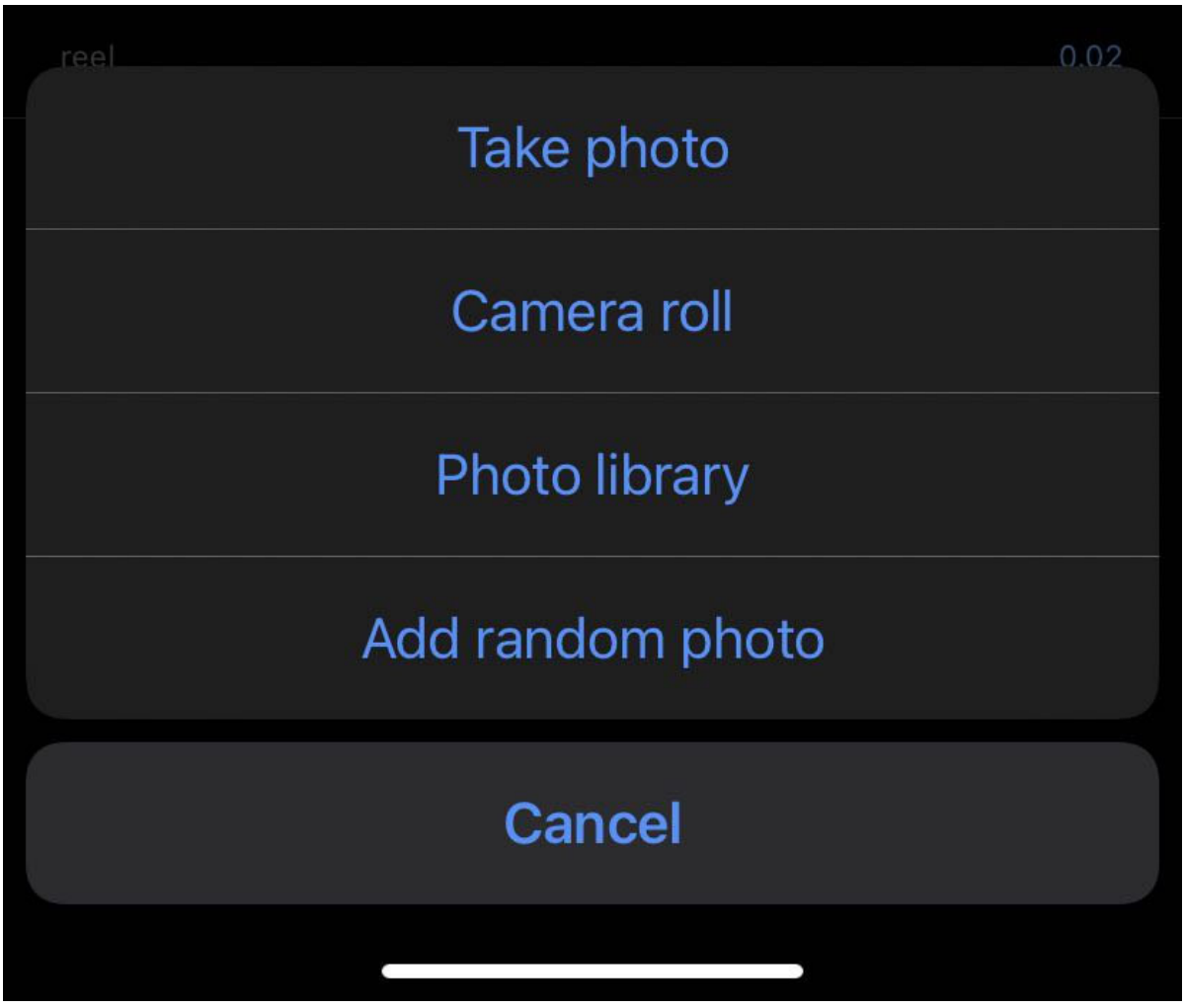

Рисунок 5.2 — Вікно вибору джерела фото

Також при тестуванні додатку було проведено перевірку на дотримання нефункціональних вимог. Додаток повністю відповідає усім поставленим вимогам.

ВИСНОВКИ

Під час виконання даної магістерської роботи було проведено дослідження сучасного стану машинного навчання, було проведено аналіз засобів розробки під мобільні платформи. Проаналізовано можливості використання засобів машинного навчання для розробки додатків під мобільні платформи. Було спроектовано та розроблено додаток для платформи IOS з використанням засобів машинного навчання для визначення що саме зображено на малюнку.

Було проаналізовано класифікацію засобів машинного навчання. Було описано математичну основу, що лежить в основі “навчання з учителем”.

Було розглянуто засоби розробки мобільних додатків. Було доведено необхідність мобільних додатків в цілому. Було проведено порівняння між нативними та кросплатформленими засобами розробків додатів для платформи IOS та Android. Було проведено порівняння між найпопулярнішими засобами розробки. У ході дослідження було виявлено переваги нативної розробки при використанні машинних засобів навчання через кращу взаємодію з девайсом користувача.

Було проаналізовано найпопулярніші та найпоширеніші моделі, що використовуються на практиці. При реалізації додатку було використано модель Inception V3, через простоту в її використані та хорошу продуктивність. Додаток було розроблено з використанням мови програмування Swift та XCode. Було протестовано розроблений демонстраційний додаток, додаток відповідає усім поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How many phones are in the world? [Електронний ресурс] – Режим доступу до ресурсу: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.
2. The Majority of Americans' Mobile Time Spent Takes Place in Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.emarketer.com/content/the-majority-of-americans-mobile-time-spent-takes-place-in-apps.
3. Burkov A. The Hundred-Page Machine Learning Book / Andriy Burkov., 2019. – 3 с.
4. Theobald O. Machine Learning for Absolute Beginners / Oliver Theobald., 2017. – (Third edition). – 19 с.
5. In depth machine learning for enterprice [Електронний ресурс] – Режим доступу до ресурсу: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML.
6. Mueller J. Machine Learning For Dummies / J. Mueller, L. Massaron., 2016. – 14 с.
7. Machine Learning vs. Traditional Programming [Електронний ресурс] – Режим доступу до ресурсу: https://www.logianalytics.com/predictive-analytics/machine-learning-vs-traditional-programming/.
8. Burkov A. The Hundred-Page Machine Learning Book / Andriy Burkov., 2019. – 6 с.
9. Godfellow I. Deep learning / I. Godfellow, Y. Bengio, A. Courville., 2015.

10. Number of apps available in leading app stores as of 1st quarter 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.
11. Mobile App Or Website? 10 Reasons Why Apps Are Better [Електронний ресурс] – Режим доступу до ресурсу: https://vwo.com/blog/10-reasons-mobile-apps-are-better/.
12. McWherter J. Professional Mobile Application Development / Jeff McWherter., 2012.
13. Programming with Objective-C [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html.
14. Swift vs Objective-C: Which One to Consider For Your Next iOS App? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ideamotive.co/blog/swift-vs-objective-c-which-should-you-pick-for-your-next-ios-mobile-app.
15. About Swift [Електронний ресурс] – Режим доступу до ресурсу: https://docs.swift.org/swift-book/.
16. Kotlin vs. Java: Which is the Better Option for Android App Development? [Електронний ресурс] – Режим доступу до ресурсу: https://clearbridgemobile.com/java-vs-kotlin-which-is-the-better-option-for-android-app-development/.
17. Kotlin overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/kotlin.

18. Best Cross-Platform Mobile Development Tools in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://litslink.com/blog/best-cross-platform-mobile-development-tools-in-2021.
19. Native Vs Cross-Platform Development: Pros & Cons Revealed [Електронний ресурс] – Режим доступу до ресурсу: https://www.uptech.team/blog/native-vs-cross-platform-app-development.
20. Tensorflow lite owerview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/lite.
21. Core ML Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreml.
22. Core ML, Vision tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://swiftbook.ru/post/tutorials/guide-coreml-and-vision/.
23. Amazon Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/machine-learning/.
24. A Gentle Introduction to Object Recognition With Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/object-recognition-with-deep-learning/.
25. Tang J. Intelligent Mobile Projects with TensorFlow / Jeff Tang., 2018. - 25 с.
26. Cocoa Core Competencies [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.
27. Design Patterns by Tutorials: MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm.