МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему Особливості розробки крос-платформного мобільного застосунку для SMM з використанням нейронних мереж

Виконав: студент 2 курсу, групи 8.1210-іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Б. О. Розов
(ініціали та прізвище)
Керівник доцент, А. І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Альтер Віжен Груп»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____ другий (магістерський) _____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____ В.Г. Вербицький
"_01_" __вересня_____2021_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

__Розову Богдану Олеговичу__
(прізвище, ім`я, по батькові)

1.Тема роботи __Особливості розробки крос-платформного мобільного застосунку для SMM з використанням нейронних мереж_____

керівник роботи ___Безверхий Анатолій Ігорович, доцент_____
(прізвище, ім`я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___"30" червня 2021 року № 974-с_____

2. Строк подання студентом кваліфікаційної роботи_____30.11.2020_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розпізнавання мов та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.09.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.21	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.2021	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.20	виконано
4	Дослідження області розробки крос-платформних застосунків	18.09-24.09.20	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.20	виконано
6	Аналіз теоретичних відомостей	27.09-15.10.20	виконано
7	Проектування інтерфейсу мобільного застосунку	15.10-23.10.20	виконано
8	Узгодження інтерфейсу з науковим керівником	23.10-24.10.20	виконано
9	Розробка нейронної мережі	25.10-14.11.20	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	15.11-16.11.20	виконано
11	Інтеграція навченої моделі нейронної мережі у мобільний застосунок	16.11-23.11.20	виконано
12	Проведення аналізу можливостей розроблених програмних застосунків	24.11-09.12.21	виконано
13	Оформлення звіту	10.12-28.12.21	виконано

Студент _____ Б.О. Розов
(підпис) (прізвище та ініціали)

Керівник роботи _____ А.І. Безверхий
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 92

Рисунків: 29

Джерел: 30

Розов Б.О. Особливості розробки крос-платформного мобільного застосунку для SMM із використанням нейронних мереж : кваліфікаційна робота магістра спеціальності 121»Інженерія програмного забезпечення» / наук. керівник А.І. Безверхий. Запоріжжя : ЗНУ, 2021 92 с.

Завданням науково-дослідницької роботи є дослідження процесу розробки маркетингового застосунку для мобільних платформ. Для досягнення мети створений мобільний застосунок, що використовує нейронні мережі для обробки та класифікації світлин користувача, а також їх просування у соціальних мережах.

Мета роботи — створення мобільного крос-платформного застосунку що базується на використанні нейронних мереж класифікації зображень та використовується для просування у соціальних мережах.

Методи досліджень — синтез та аналіз.

Результатом роботи є програмний застосунок, що дозволяє користувачу просувати свої світлини у соціальній мережі.

Галузь застосування охоплює маркетологів та менеджерів з роботи у соціальних мережах.

Ключові слова: мобільний застосунок, згорткова нейронна мережа, соціальна мережа, маркетинг соціальних медіа-ресурсів, хештег, flutter, firebase

SUMMARY

Pages: 92

Pictures: 29

Sources: 30

Rozov BO Features of development of cross-platform mobile application for SMM with the use of neural networks: qualification work of the master of specialty 121 "Software Engineering" / science. head A.I. Bezverkhyi. Zaporizhzhia : ZNU, 2021, 92 p.

The task of research is to study the process of developing a marketing application for mobile platforms. To achieve this goal, a mobile application has been created that uses neural networks to process and classify user photos, as well as promote them on social networks.

The purpose of the work is creation of a mobile cross-platform application based on the use of neural networks for image classification and used to promote them in social networks.

Research methods — synthesis and analysis.

The result is a software application that allows the user to promote their photos on a social network.

The field of application covers marketers and managers of social networks.

Keywords: mobile application, convolutional neural network, social network, social media marketing, hashtag, flutter, firebase

ЗМІСТ

ВСТУП

Актуальність теми

Задача маркетингу гостро стоїть перед кожною сучасною продуктовою компанією яка зацікавлена у своєму розвитку, поширювані впливу та збільшення клієнтської бази. У цілях вирішення задач маркетингу компанії впроваджують все більший маркетинговий бюджет та наймають провідних SMM спеціалістів. Проте якість виконаної роботи залежить не лише від навичок людей та витрачених грошей, а й насамперед від використаного інструментарію.

Протягом останніх років самі маркетологи та більшість їх цільової аудиторії зробила вибір на користь мобільних пристроїв, що найбільше використовуються. С цього виходить, що як під час вибору платформи для розробки нового інструментарію, так і під час таргетування інформації розробникам та маркетологам варто звернути на них увагу у першу чергу.

Мета і завдання дослідження

Метою роботи є розробка сучасного крос-платформного мобільного застосунку, який може бути використаним користувачем задля просування власних постів у соціальних мережах.

Об’єкт дослідження

Об’єкт дослідження — побудований шлях навігації.

Предмет дослідження

Предмет дослідження — проблема просування контенту у соціальних мережах та автоматизація процесу створення постів

Наукова новизна одержаних результатів

Використання нейронних мереж для рішення таких усталених проблем як просування контенту, допоможе зекономити час який люди витрачають на створення та таргетування постів.

Практичне значення одержаних результатів

Рекламні бюджети провідних компаній зростають с кожним днем у той самий час як конкуренція за увагу користувача збільшується. Використання готового рішення для створення та таргетування контенту за допомогою нейронних мереж є найбільш практичним способом економії грошей та часу для вирішення задачі просування.

Апробація одержаних результатів

Результати дослідження були представлені на XIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2020» [1], а також на XXV науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету [2].

Глосарій

- SMM (social media marketing, Маркетинг у соціальних мережах) — це набір методів просування людей чи компаній або спосіб вирішити задачі який утилізує соціальні медіа.
- Хештег або гештег (англ. hashtag, від hash — «знак решітки», і tag — мітка) — ключове слово або фраза, яким передує символ «#» (октоторп), використовувані веб-технологіями для структурування текстових повідомлень за темою або типом. Короткі повідомлення в мікроблогах соціальних мереж, таких як: Instagram, Twitter, Facebook, можуть бути помічені хештегом, включаючи в себе як одне слово, так і більше об'єднаних слів (але без пробілів).
- Впевненість нейронної мережі (англ. Confidence) — це значення на вихідному шарі нейронної мережі, що визначає ймовірність приналежності вхідних даних до певного класу об’єкта.
- Шту́чні нейро́нні мере́жі (ШНМ, англ. artificial neural networks, ANN) — це обчислювальні системи, натхнені біологічними нейронними мережами,

що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу.

- Глибинне навчання (англ. Deep learning) — це сукупність методів машин-ного навчання (з вчителем, з частковим залучення вчителя, без вчителя та з підкріпленням), що ґрунтуються на навчанні ознак з даних, використовуючи багатошарові нейронні мережі.
- Глибока нейронна мережа (англ. Deep neural network) — це нейронна мережа з багатьма прихованими шарами.
- Згорткова нейронна мережа (англ. Convolutional neural network) — це нейронна мережа прямого поширення спеціального виду, що працює на основі фільтрів, які займаються розпізнаванням характеристик певних класів об’єктів на зображенні: прямих ліній, кривих певного типу тощо.
- Метод зворотного поширення помилки (англ. Backpropagation) — це ітеративний алгоритм, що є модифікацією методу градієнтного спуску, та використовується з метою мінімізації помилки роботи багатошарової нейронної мережі.
- Смартфо́ни (з англ. smart — розумний, і англ. phone — телефон) — окрема категорія телефонів, які — на відміну від простих стільникових телефонів — мають більше оперативної пам'яті і власний потужний, як для кишенькових пристроїв процесор, працюють під операційною системою Symbian 6.1 і вище, операційними системами платформи Windows Mobile 5 і вище або Palm OS, операційною системою iOS, Android, Tizen, Bada. Завдяки таким даним підтримують багато програм написаних на C++ та java зокрема 3D-ігри
- Фреймворк (англ. Framework, каркас, платформа, структура, інфраструктура) — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проекту та написання коду.

- Бібліотека (англ. library) — збірка об'єктів чи підпрограм для вирішення близьких за тематикою задач. У залежності від мови програмування бібліотеки містять об'єктні модулі чи сирцевий код та дані, допоміжні для задіяння та інтеграції нових можливостей в програмні рішення.
- Прикладни́й програ́мний інтерфе́йс (англ. Application Programming Interface, API) — це зв’язок між комп’ютерами або між комп’ютерними програмами. Це тип програмного інтерфейсу, що пропонує послугу для інших програм.
- SDK (від англ. Software Development Kit) — являє собою набір засобів розробки програмного забезпечення в одному встановленому пакеті
- Канвас (англ. Canvas) — це контейнер, який вміщує різні елементи малювання (лінії, фігури, текст, рамки, що містять інші елементи тощо). Свою назву бере від полотна, що використовується в образотворчому мистецтві.
- Нативна розробка — побудова застосунку з направленістю лише на одну конкретну платформу використовуючи запропонований розробником платформи SDK
- Мінімально життєздатний продукт (англ. Minimum Viable Product, MVP) — це версія продукту з достатньою кількістю функцій, яка може бути використана ранніми клієнтами, які потім можуть надати відгук для майбутньої розробки продукту.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ ТА НЕЙРОНИХ МЕРЕЖ

1.1 Аналіз стану питання розробки мобільних застосунків

Історія мобільних додатків, виходячи із статі Самуеля Аду-Гаямфі [3], починається з історії мобільних пристроїв та перших мобільних телефонів, мікро чіпи, яких вимагали найпростішого програмного забезпечення для надсилання та прийому голосових дзвінків. Але з того часу все набагато ускладнилося.

Перший публічний дзвінок на стільниковий телефон зробив Мартін Купер з Motorola 3 квітня 1973 року в рекламному акції в Нью-Йорку. Далі було ще десять років досліджень і розробок до того, як перший мобільний телефон з’явиться на ринку. DynaTAC 8000X важив близько 2 фунтів, коштував 4000 доларів і не запускав жодних програм.

Перші впізнавані програми з’явилися з асортиментом портативних комп’ютерів Psion (переважно КПК), які використовували операційну систему EPOC. Вперше випущені на початку 90-х років 16-розрядні машини (SIBO), на яких працював EPOC, дозволяли користувачам використовувати такі програми, як текстовий редактор, база даних, електронна таблиця та щоденник. Пізніші моделі цього ряду, що працюють на 32-розрядної ОС, оснащувалися оперативною пам'яттю до 2 МБ і дозволяли користувачам додавати додаткові програми за допомогою програмних пакетів (або через завантаження, якщо вам щастило володіти модемом на той час).

EPOC, який був запрограмований на OPL (відкрита мова програмування) дозволяв користувачам створювати власні програми, згодом складе основу операційної системи Symbian.

1.1.1 Розробка під платформу Android

Історія Android, спираючись на статтю Девіда Тілсона [4], починається в жовтні 2003 року — задовго до того, як термін смартфон широко використовувався, і за кілька років до того, як Apple оголосила про свої перші iPhone та iOS. Компанія Android Inc була заснована в Пало-Альто, штат Каліфорнія. Його чотирма засновниками були Річ Майнер, Нік Сірс, Кріс Уайт та Енді Рубін. С того часу, широко цитуються слова Рубіна про те, що Android Inc збирається розробляти «розумніші мобільні пристрої, які більше знають про місцезнаходження та уподобання власника».

У 2005 році розпочався наступний великий розділ в історії Android, коли оригінальну компанію придбав Google. Рубін та інші члени-засновники продовжували розробляти ОС під керівництвом своїх нових власників. Було прийнято рішення використовувати Linux як основу для ОС Android. Це означало, що операційну систему можна безкоштовно пропонувати стороннім виробникам мобільних телефонів. Google і команда Android вважали, що компанія може заробляти гроші, пропонуючи інші послуги, що використовують ОС, включаючи додатки.

Розрбка будь-якого мобільного додатку для Android складається с наступних інструментів:

- Android Studio — це офіційний інструмент розробки, IDE для Android. Розробники використовують його для створення додатків для будь-якого типу пристроїв Android. Android Studio включає інструменти для редагування коду, налагодження та продуктивності. Використовуйте ці інструменти для створення високоякісних програм, які, на вашу думку, можуть допомогти вашим потенційним клієнтам.
- Gradle — Ця програма покращує продуктивність розробників для компаній усіх розмірів. Розробники можуть кодувати вибраною мовою. Gradle готує ваш мобільний додаток до розгортання, тому ваша компанія може автоматично доставити ваше програмне забезпечення на об-

рані вами платформи. Універсальність Gradle дозволяє створювати будь-який тип мобільного додатка, який ви плануєте.

- Android Runtime (ART) — програмне забезпечення для виконання керує низькорівневими процесами, що працюють у фоновому режимі. Особливості ART включають:
  - Попередня компіляція (AOT)
  - Вивіз сміття
  - Покращення розробки та налагодження
  - Діагностична звітність

1.1.2 Розробка під платформу iOS

Як зазначив Андрій Беленко у своїй роботі [5], народження iPhone iOS, відомої тоді як iPhone OS, було зроблено Стівом Джобсом на заході Macworld на початку 2007 року. На той час iPhone мав обмежені можливості та не мав магазину додатків. Однак головною привабливістю iPhone на той час було те, що, хоча інші пристрої все ще використовували резистивні сенсорні екрани, Apple революціонізувала свій iPhone завдяки ємнісним сенсорним можливостям. Це зробило весь досвід роботи смартфона плавним та швидким для користувачів. Перший iPhone був «чарівним» продуктом, який на той час вже випереджав будь-який мобільний телефон на 5 років.

На початкових етапах iPhone не мав магазину додатків. Все, що у вас було, було вбудовано в програми від Apple. Випуск App Store в 2008 році, вбудований в ОС, означав, що користувачі можуть переглядати, а потім завантажувати програми безпосередньо на пристрій. Apple також надала функцію, яка приймає облікові записи itunes, щоб люди могли безпосередньо завантажувати програми з магазину, використовуючи той самий обліковий запис, який вони використовували для завантаження пісень з iTunes на своїх комп'ютерах. Також були додані деякі інші функції, такі як підтримка Microsoft Exchange, яка дозволила надсилати електронні листи, календарі та контакти з інших джерел та MobileMe від Apple.

Розробка мобільних застосунків для платформи iOS відбувається за домопогою Xcode.

Xcode — це інтегроване середовище розробки Apple для macOS, що використовується для розробки програмного забезпечення для macOS, iOS, iPadOS, watchOS та tvOS. Доступний через Mac App Store безкоштовно для користувачів macOS. Зареєстровані розробники можуть завантажувати попередні випуски та попередні версії набору через веб-сайт Apple Developer. Xcode включає інструменти командного рядка (CLT), які дозволяють розробляти стиль UNIX за допомогою програми Terminal у macOS. Їх також можна завантажити та встановити без основної IDE.

1.1.3 Крос-платформна розробка

Оскільки мобільні додатки стають все більш популярними, як зазначили Мануель Палм'єрі та Індержит Сінх у свої науковій праці [6], а технологія стає загальнодоступною, попит на стійкі практики та інструменти для побудови та підтримки постійної розробки стає очевидним. Аналізуючи статтю Генінга Геіткьоттеру [7], розробка крос-платформних мобільних додатків може бути досягнута кількома способами.

Перехресна компіляція. Одним із можливих підходів для вирішення задачі крос-платформної розробки є використання методу, який називається перехресною компіляцією: крос-компілятор відокремлює середовище побудови від цільового середовища, ефективно відокремлюючи джерело від цільової платформи. У контексті мобільної розробки він працює наступним чином: фреймворк забезпечує незалежний від платформи API із використанням основної мови програмування (наприклад, JavaScript, Ruby або Java). Розробники використовують API для побудови мобільного додатку, включаючи інтерфейс користувача, збереження даних та бізнес-логіку. Потім код обробляється крос-компілятором, який перетворює його на певні програми, націлені на різні платформи, на яких програма працюватиме. Артефакт програм-

ного забезпечення, згенерований у результаті цього процесу, може бути розгорнутий та виконаний на пристрої.

Перевагами цієї техніки є: продуктивність, оскільки додаток працює нативно на пристрої; покращений досвід роботи, оскільки додаток поводиться як звичайний додаток в екосистемі користувача; і повний нативний доступ до ряду можливостей, що стосуються конкретних пристроїв, таких як вбудована камера, датчики тощо. Великим недоліком є складність, оскільки крос-компілятори можуть бути важкими для написання, і їх потрібно підтримувати у відповідності з фрагментованими мобільними платформами та наявними операційними системами.

Мобільні веб застосунки. Іншим дедалі популярнішим підходом є створення додатку як мобільної веб-програми, яка працюватиме в мобільному браузері користувача. Це передбачає використання стандартних веб-технологій, таких як HTML, CSS та JavaScript, щоб побудувати додаток та дати змогу йому виглядати та поводитися як нативний додаток. Це можливо завдяки розширеним можливостям HTML 5 та CSS 3, включаючи вбудовані бази даних SQL, локальне сховище, анімацію, канвас, веб-сокети та програвання відео.

Незважаючи на те, що HTML 5 це все ще молода технологія (стандарт ще не доопрацьований), і мобільні браузери можуть реалізовувати її по-різному, її зростаюча популярність в механізмів рендерингу, як WebKit, який працює на мобільних браузерах iPhone та Android, дає змогу веб-програми виглядати і поводитись більш і більше як нативні програми.

1.1.4 Порівняння нативної та крос-платформної розробки

Нативна розробка проти крос-платформної, як можливо побачити у статі [8] — це нескінченна дискусія, яка протягом багатьох років роз’єднує технічну спільноту. Є кілька експертів, які віддають перевагу нативним додаткам перед крос-платформними. З іншого боку, такі компанії, як Uber, випус-

кають свою платформу додатків для різних платформ, щоб переписати свою програму для працівників.

Як нативні, так і міжплатформні технології розробки перебувають у постійному стані розвитку. Ця зміна характеру технологій сигналізує про те, що ці теми слід час від часу переглядати, щоб перевірити, який із цих варіантів в даний час є провідною у грі.

Розробка нативних додатків уникає складності створення стійкого продукту, який охоплює розробку кількох платформних додатків і фокусується на створенні грамотного дизайну, який залишається близьким до цільової платформи — Android, iOS тощо.

Крос-платформні фреймворки прагнуть створити програму, яка охоплює якомога більше послідовників вашого бренду, охоплюючи велику кількість кінцевих пристроїв під час процесу програмування та розробки.

Наглядне порівняння підходів наведене у таблиці 1.

Таблиця 1

Порівняльна характеристика технологій розробки застосунків

	Нативний застосунок	**Крос-платформний застосунок**
Ціна	Висока ціна розробки	Відносно низька ціна розробки
Використання коду	Код працює лише на одній платформі	Один код можем бути використаний на кількох платформах
Використання можливостей девайсу	SDK платформи забезпечує безперешкодний доступ до API пристрою	Немає гарантованого доступу до усіх API пристрою
Послідовність інтерфейсу користувача	Відповідно до компонентів інтерфейсу користувача пристрою	Обмежена узгодженість із компонентами інтерфейсу користувача пристрою
Продуктивність	Бездоганна продуктивність, враховуючи, що додаток розроблений для пристроїв ОС	Висока продуктивність, але відставання та проблеми сумісності апаратного забезпечення не рідкість

1.1.5 Виклики крос-платформної розробки

Кілька років тому розробка крос-платформних додатків була обмежена створенням простих мобільних додатків та ігор. З часом нові технології зробили крос-платформну розробку більш пристосованою, потужною та гнучкою, ніж раніше. Однак крос-платформна розробка все ще стикається із такими проблемами:

- Збій у продуктивності через непослідовний зв'язок між нативними та не-нативними компонентами гаджетів.
- Крос-платформні розробники підтримують перехресну відповідність програм обмеженими інструментами
- Проблеми, пов'язані з продуктивністю, можуть призвести до погіршення користувацького досвіду.
- Якщо ваш діловий додаток керує більшою кількістю корпоративних даних та даних користувача, тоді перехід до міжплатформенних додатків — це не найкраща ідея відповідно до проблем безпеки.

  Але ці випробування мінімальні у порівнянні з перевагами.

1.1.6 Переваги крос-платформної розробки

З огляду на експоненціальне зростання вартості розробки платформи та необхідність швидкого виходу на ринок, крос-платформний розвиток — це найкращий шлях для бізнесу.

До переваг крос-платформної розробки можна віднести:

- Максимальний вплив на цільову аудиторію — використовуючи мобільний крос-платформний підхід, ви можете створити додаток та розгорнути його на різних платформах, включаючи Інтернет. Це означає, що, будуючи єдиним додатком, можна націлити обидві платформи — iOS і Android, таким чином, максимізуючи їх охоплення.
- Знижені витрати на розробку — розробка крос-платформного додатку базується на концепції «пиши один раз, запускай скрізь». Коди багаторазового використання та гнучка розробка додатків можуть

зменшити витрати на розробку. Тому, щоб покращити свій бізнес на декількох платформах та інструментах економічно вигідним способом, немає іншої альтернативи крос-платформним додаткам.

- Простіше обслуговування та розгортання — оскільки існує лише одна розроблена програма, яка працює на всіх платформах, її простіше підтримувати, а також розгортати код або вносити зміни. Оновлення можна оперативно синхронізувати на всіх платформах і пристроях, тим самим заощаджуючи час і гроші. Більше того, якщо помилка виявлена в загальній кодовій базі, її слід виправити один раз. Таким чином розробники можуть значно заощадити час і гроші.

1.2 Огляд існуючих технологій крос-платформної розробки

1.2.1 Ionic

Ionic — це, спираючись на книгу Кріса Гріфіта [9], один із потужних SDK HTML5, який дозволяє розробляти мобільні програми, використовуючи передові технології, такі як HTML, CSS та Javascript. Цей SDK в основному фокусується на зовнішньому вигляді та взаємодії інтерфейсу програми.

Додатки, розроблені на платформі Ionic, можуть використовуватися на різних платформах, таких як Android, iOS, стаціонарні ПК та веб сайти з єдиною базою коду. Ця відома крос-платформена технологія забезпечує заздалегідь розроблені компоненти, типографіку та чудові теми.

Цей фреймворк простий у використанні та розумінні, і він використовує API, такі як TypeScript, Virtual DOM, JSX та async, які найкраще підходять для Progressive Web App (PWA). Програми Pacifica, Nationwide, ChefSteps та ін. побудовані на основі Ionic.

Переваги:

- Ionic базується на інтерфейсі SAAS UI, розробленому спеціально для мобільних операційних систем. Він пропонує численні компоненти інтерфейсу для розробки надійних додатків.

- Від автоматизованих власних збірок до оновлення в реальному часі та CI/CD, Ionic Appflow звертається до всього життєвого циклу мобільних DevOps.
- Ionic підтримується активним співтовариством понад 5 мільйонів розробників у понад 200 країнах.

  Недоліки:
- Знання AngularJS стає майже необхідним, якщо хочеться вийти за рамки базових програм.
- Проектування навігації в програмі є складним через не дуже простий у використанні UI-маршрутизатор.
- Потужність застосунків гірша за нативну

1.2.2 NativeScript

Виходячи із книги Натаніеля Андерсона [10], це фреймворк з відкритим кодом, який створює міжплатформні програми для iOS та Android за допомогою JavaScript. Він переводить одну мову програмування на іншу, створюючи власні програми за допомогою Angular, Vue JS та TypeScript. На відміну від інших фреймворків, які використовують Cordova для рендерингу інтерфейсу додатка, керованого WebView, NativeScript має механізм рендерингу, який забезпечує нативну потужність та зручність користування.

Переваги:

- Розширюваність — надає повний та прямий доступ до всіх видів API для iOS та Android. Це забезпечує доступність і дозволяє повторно використовувати безкоштовні плагіни, Android SDK та CocoaPods.
- Інструмент CLI, зручний для розробників: NativeScript CLI дозволяє розробникам робити майже все, починаючи від додавання платформи до розгортання програм на певній платформі чи пристрої. Встановлення плагінів та налагодження додатків відбувається швидше та зручніше.

  Недоліки:

- Обмеження інтерфейсу користувача — DOM і HTML не підтримуються широко, що залишає необхідність навчитися використанню різних компонентів інтерфейсу. Це з’їдає значну частину вашого часу та бюджету.
- Неперевірені плагіни: загальна кількість перевірених плагінів значно менша. Отже, немає жодної впевненості у якості плагінів, що використовуються в цьому фреймворку.

1.2.3 React-Native

React Native, цитуючи книжку Бонні Ейсенмана [11], дозволяє розробникам створювати крос-платформні програми, які виглядають і нагадують нативні програми з можливістю доступу до нативних функцій мобільного пристрою (камери, місцезнаходження користувача тощо). Ця функція робить його гарним інструментом для Android, iOS та інших мобільних операційних систем. Це дозволяє створити швидкий і привабливий додаток за допомогою JavaScript, не жертвуючи користувацьким досвідом або швидкою продуктивністю. Більше того, цей міжплатформенний інструмент мобільної розробки має широку спільноту прихильників по всьому світу та надає вам корисні та зрозумілі документи для вирішення будь-яких питань.

Окрім Facebook, є багато компаній, які використовують React Native для своїх крос-платформних додатків — Bloomberg, Skype, Tesla, Pinterest, Walmart.

Переваги:

- До 80% кодової бази можна використовувати спільно між платформами, залежно від складності програми.
- Окрім багаторазового використання коду, це дозволяє відразу переглядати результати, крім того, що пропонує готові до застосування елементи, тим самим значно скорочуючи час розробки.

- Функція «Hot Reloading» дозволяє розробникам бачити зміни, внесені в код, протягом декількох секунд, а не хвилин під час використання нативних технологій.
- React Native фокусується на UI значною мірою, надаючи чутливий інтерфейс.
- Він також дає вам доступ до певних чудових нативних можливостей, таких як акселерометр та камера. Результатом цього є якісний користувальницький інтерфейс, схожий на нативний.

Недоліки:

- React Native не є повною мірою крос-платформним. Для використання деяких функцій, таких як камера або акселерометр, потрібно використовувати нативні компоненти, тому буде окремий код для Android та iOS.
- Оскільки фреймворк не будується спільно з iOS або Android, він часом відстає від нативних платформ. Це одна з причин, яка змусила Udacity припинити інвестувати в React Native для нових функцій.
- React Native не має послідовності щодо випуску оновлень.
- React Native покращує швидкість розробки, але також збільшує тривалість процесу налагодження, особливо на Android.

1.2.4 Flutter

Один з кращих крос-платформний мобільний інструмент розробки. Цей фреймворк з відкритим кодом, випущений Google у 2017 році. Flutter створює нативні програми, які можуть підтримуватися багатьма операційними системами, такими як iOS, Android та інші. Основною мовою програмування фреймворку є Dart, що, виходячи із статті Микити Кузьміна [12], забезпечує збільшення продуктивності до 15%.

З оновленням в 2019 році, Google додав корисну функцію — вас підтримають при інтеграції Flutter у вашу існуючу програму iOS або Android, представлену в магазині.

Серед послідовників Flutter ви можете знайти Abbey Road Studios, Alibaba, BMW, Groupon, Google, Tencent та інших.

Переваги:

- Функція «Hot Reloading» дозволяє розробникам бачити зміни, внесені в код, протягом декількох секунд, а не хвилин, як при використанні нативних технологій.
- Це ідеальна основа для розробки MVP. Замість того, щоб витрачати зайві гроші та час на дві окремі програми, ви можете швидко створити мобільний додаток Flutter, який виглядає нативним як на Android, так і на iOS.
- Flutter базується на Dart, об’єктно-орієнтованій мові програмування, для якої розробникам було досить легко набути навичок.
- Flutter має повний набір віджетів у Material Design від Google та в стилі Apple із пакетом Cupertino.
- Багато готових рішень для власних додатків для Android та iOS дозволяють працювати з платформами безперервної інтеграції, такими як Тревіс та Дженкінс.

Недоліки:

- Існує обмежена підтримка телевізора з програмами, побудованими на платформі Flutter, тобто Flutter не підтримує Android TV та Apple TV.
- Хоча внаслідок розробки Google, існує декілька бібліотек з готовими до впровадження функціоналами, Flutter все ще бракує функціоналу у порівнянні з нативною розробкою.
- Оскільки додатки з підтримкою Flutter використовують вбудовані віджети, а не віджети платформи, розмір програми, як правило, більший. В даний час найменша з можливих програм, створених за допомогою Flutter, може важити не менше 4 Мб.

	Descriptives				Analysis against native		
Framework	Mean	SD	Max	Min	ANOVA p	ω^2	Tukey p
Native	17.50	8.24	49.60	5.90	–	–	–
React Native	23.17	13.57	66.60	1.80	$< .001$	.059	$= .000$
MAML/MD_2	16.11	8.12	45.90	5.90	$= .001$	.006	$= .101$
Ionic	22.35	11.11	59.93	0.09	$< .001$	.057	$= .000$
Flutter	19.60	9.64	56.75	0.00	$< .001$	.013	$= .001$
NativeScript	15.71	8.38	35.73	5.00	$= .001$	.010	$= .060$

Рис. *1 Огляд завантаження процесора смартфона*

1.3 Висновки з порівняльного аналізу технологій крос-платформної розробки

Аналізуючи різні підходи до розробки та, базуючись на роботі Ваєву Ву [13], можна зробити висновок що кожен має свої переваги. Оскільки для вирішення поставленої задачі є зацікавленість у продуктивній праці застосунку, раціональним рішенням буде відмовитися від технології розробки що базуються на побудові веб застосунків. Порівнюючи інші варіанти технологій було вирішено зупинитися на Flutter як на оптимальному інструменті розробки, адже від запроваджую зручність розробки та продуктивність застосунку на платформах для яких розробляє застосунок.

1.4 Аналіз стану питання нейронних мереж

Ідея нейронних мереж, як не дивно почалася як модель того, як нейрони в мозку функціонують, називаючись «конекціонізмом» і використовуючи з'єднані схеми для імітації розумної поведінки. Згідно зі статтею Джона Габуссі [14], у 1943 році вона була зображена як проста електрична схема нейрофізіологом Уорреном Маккалоком і математиком Уолтером Піттсом. Дональд Хебб розвив цю ідею далі у своїй книзі "Організація поведінки" (1949), пропонуючи підсилювати нервові шляхи при кожному наступному

застосуванні, особливо між нейронами, які мають тенденцію працювати одночасно, тим самим починаючи довгий шлях до кількісної оцінки складних процесів мозку.

Дві основні концепції, які є попередниками нейронних мереж:

- «Порогова логіка» — перетворення безперервного введення в дискретний вихід
- «Геббіанське навчання» — модель навчання, заснована на нейронній пластичності, запропонована Дональдом Хеббом у своїй книзі «Організація поведінки», яка часто узагальнюється фразою: «Клітини, які працюють разом, встановлюють зв’язок між собою».

Обидві запропоновані в 1940-х. У 1950-х роках, коли дослідники почали намагатися перевести ці мережі на обчислювальні системи, перша мережа Геббі була успішно реалізована на MIT в 1954 році.

Приблизно в цей час Френк Розенблатт, психолог з Корнелла, працював над розумінням порівняно простіших систем прийняття рішень, наявних в оці мухи, які лежать в основі та визначають реакцію втечі. Намагаючись зрозуміти і кількісно визначити цей процес, він запропонував ідею Персептрона в 1958 році, назвавши його Марк I Персептроном. Це була система з простим співвідношенням вихідних даних, змодельована на нейроні Маккалока-Піттса, запропонована в 1943 р. Уорреном С. Маккаллохом, неврологом, та Уолтером Піттсом, логіком для пояснення складних процесів прийняття рішень у мозку за допомогою «linear threshold gate». Нейрон Маккалока-Піттса бере вхідні дані, приймає вагу і повертає «0», якщо результат нижче порогового значення, а «1» — в іншому випадку.

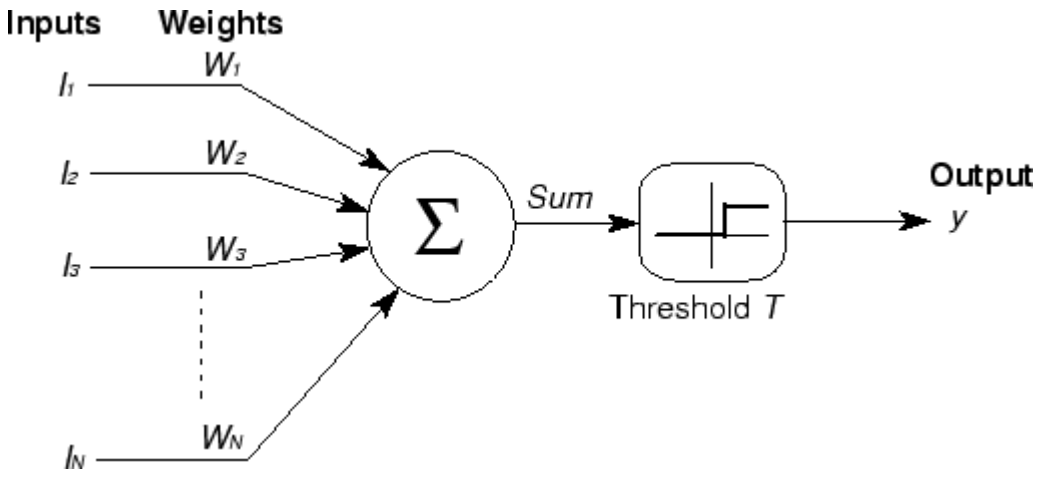

Рис. *2 Нейрон Маккалока-Піттса*

Перевага Марка I Персептрона полягала в тому, що його ваги можна було отримати шляхом послідовно переданих вхідних даних, мінімізуючи різницю між бажаними та фактичними вихідними даними.

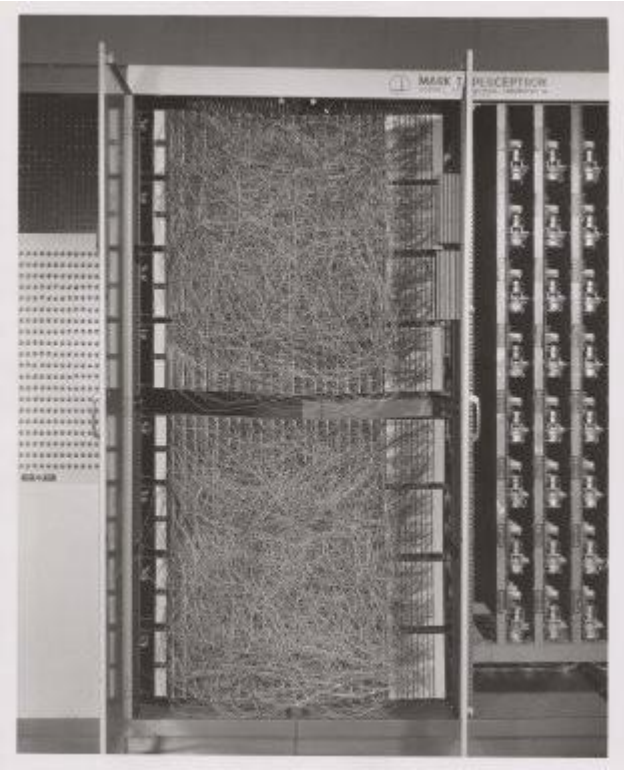

Рис. *3 Перша відома реалізація перцептрона Mark I*

Основний недолік? Цей персептрон міг лише навчитися відокремлювати лінійно відокремлювані класи, роблячи простий, але нелінійну XOR схему нездоланним бар'єром.

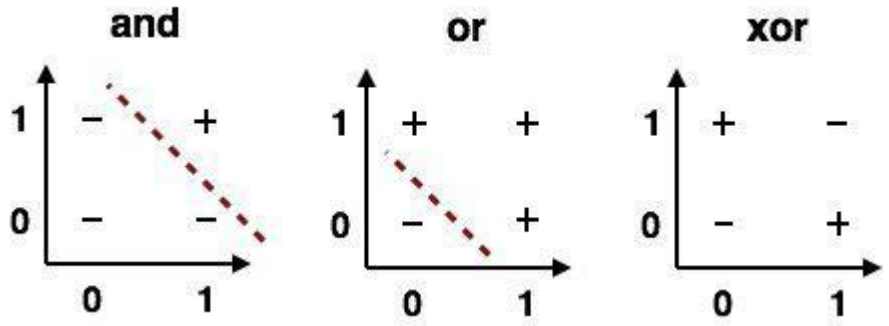

Рис. *4 Наглядна демонстрація недоліку персептрона*

Незважаючи на безладний і дещо незадовільний випадок використання машинного навчання для кількісного визначення систем прийняття рішень, крім мозку, сучасні штучні нейронні мережі — це не що інше, як кілька шарів цих персептронів.

1.5 Нейронна мережа класифікації зображень

Convolutional neural network (CNN) — це клас нейронних мереж глибокого навчання. CNN представляють величезний прорив у розпізнаванні зображень. Вони найчастіше використовуються для аналізу візуальних зображень і часто працюють за кадром при класифікації зображень. Їх можна знайти в основі всього: від позначення фотографіями Facebook до самокерованих автомобілів. Вони працюють у всьому, від охорони здоров’я до безпеки. Вони швидкі та ефективні. Але як вони працюють?

Класифікація зображень — це процес отримання вхідних даних (як зображення) та виведення класу (наприклад, "кішка") або ймовірність того, що вхідні дані є певним класом ("існує 90% ймовірності, що цей вхід є котом").

CNN має:

- Convolutional (згорткові) шари
- ReLU шари (Шар зрізаних лінійних вузлів)
- Pooling шари (Шари агрегування)
- a Fully connected шар (Шар повного з’єднання)

  Класична архітектура CNN виглядала б приблизно так:

  Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->
  ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected

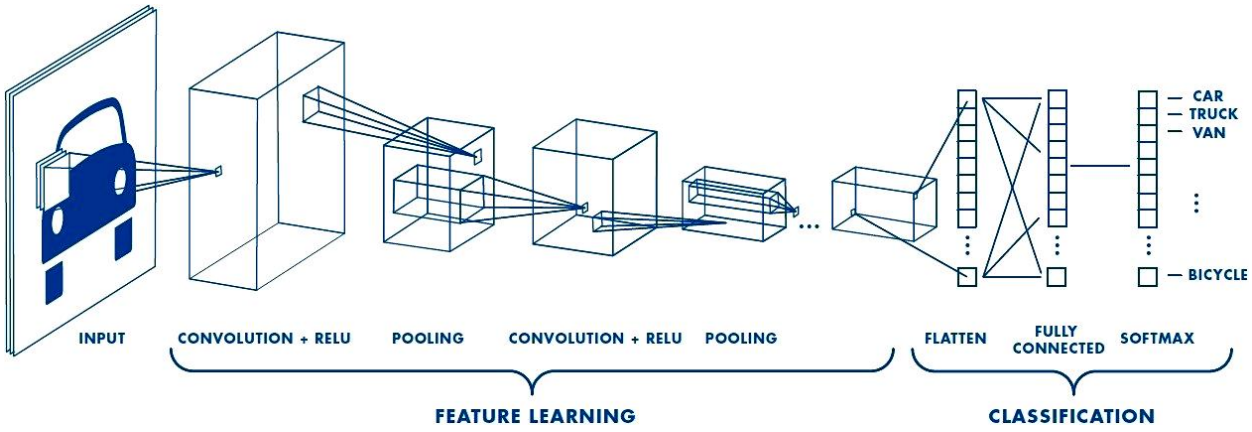

Рис. *5 Приклад архітектури CNN*

CNN, базуючись на книжці Зангу та Жу [15], згортає вивчені функції з вхідними даними і використовує 2D згорткові шари. Це означає, що цей тип мережі ідеально підходить для обробки 2D-зображень. Порівняно з іншими алгоритмами класифікації зображень, CNN насправді використовують дуже мало попередньої обробки. Це означає, що вони можуть вивчити фільтри, які повинні бути виготовлені вручну в інших алгоритмах. CNN можуть бути використані в багатьох програмах — від розпізнавання зображень та відео, класифікації зображень та систем рекомендацій до обробки природної мови та медичного аналізу зображень.

CNN натхненні біологічними процесами. Вони базуються на дослідженнях, проведених Хубелем та Візелем у 60-х роках щодо зору у котів та мавп. Шаблон зв’язку в CNN походить від їхніх досліджень щодо організації

зорової кори. В оці ссавця окремі нейрони реагують на зорові подразники лише в рецептивному полі, яке є обмеженою областю. Сприйнятливі поля різних регіонів частково перекриваються так, що охоплюється все поле зору. Це спосіб роботи CNN.

CNN мають вхідний шар, вихідний шар і приховані шари. Приховані шари зазвичай складаються із згорткових шарів, шарів ReLU, шарів об'єднання та повністю з'єднаних шарів.

- Згорткові шари застосовують операцію згортки до входу. Це передає інформацію наступному шару.
- Об’єднання з’єднує виходи кластерів нейронів в єдиний нейрон наступного шару.
- Повністю з'єднані шари з'єднують кожен нейрон в одному шарі з кожним нейроном у наступному шарі.

У згортковому шарі нейрони отримують вхід лише з підрайону попереднього шару. У повністю зв’язаному шарі кожен нейрон отримує вхідні дані від кожного елемента попереднього шару.

CNN працює, витягуючи збудження із зображень. Це позбавляє від необхідності ручного вилучення функцій. Збудження не навчені. Їх вивчають, поки мережа тренується на наборі зображень. Це робить моделі глибокого навчання надзвичайно точними для завдань комп’ютерного зору. CNN вивчають виявлення збудження через десятки або сотні прихованих шарів. Кожен шар збільшує складність вивчених збуджень.

1.5.1 Алгоритм роботи CNN

Виразити принцип роботи згорткової нейронної мережі, як його зазначили Кріжевський та Сутскевер у своїй книжці [16], можна у наступну послідовність дій:

1. Починається з вхідного зображення
2. Застосовує до нього багато різних фільтрів для створення карти збудження

3. Застосовує функцію ReLU для збільшення нелінійності.
4. Застосовує шар об’єднання до кожної карти збудження.
5. Згладжує об’єднані зображення в один довгий вектор.
6. Вводить вектор у повністю з’єднану штучну нейронну мережу.
7. Обробляє особливості через мережу. Остаточний повністю зв’язаний рівень забезпечує “голосування” класів, які нам необхідні.
8. Тренується через пряме поширення та зворотне поширення протягом багатьох епох. Це повторюється, поки не буде отримано чітко визначену нейронну мережу з навченими вагами та функціональними детекторами.

На самому початку цього процесу вхідне зображення розбивається на пікселі. Для чорно-білого зображення ці пікселі інтерпретуються як 2D-масив (наприклад, 2x2 пікселі). Кожен піксель має значення від 0 до 255. (Нуль повністю чорний, а 255 — повністю білий. Між цими цифрами існує відтінок сірого). На основі цієї інформації комп’ютер може почати працювати з даними. Для кольорового зображення це тривимірний масив із синім шаром, зеленим шаром і червоним шаром. Кожен з цих кольорів має своє значення від 0 до 255. Колір можна знайти, поєднавши значення в кожному з трьох шарів.

1.6 Архітектура CNN

1.6.1 Convolution (Згортання)

Основною метою етапу згортки є вилучення об’єктів із вхідного зображення. Згорнутий шар завжди є першим кроком в CNN.

У вас є вхідне зображення, детектор збудження та карта збудження. Ви берете фільтр і застосовуєте його піксельний блок за піксельним блоком до вхідного зображення. Ви робите це шляхом множення матриць.

Скажімо, у вас є ліхтарик і лист обгортки з бульбашками. Ваш ліхтарик висвітлює область розміром 5 бульбашок x 5 бульбашок. Щоб переглянути

весь аркуш, ви б провели ліхтариком по кожному квадрату 5x5, поки не побачите всі бульбашки.

Світло від ліхтарика тут — це ваш фільтр, а область, по якій ви ковзаєте — поле сприйняття. Світло, що ковзає по полях сприйняття, — це ваш ліхтарик, що крутиться. Ваш фільтр — це масив чисел (також званих вагами або параметрами). Відстань, яку світло від вашого ліхтарика ковзає під час проходження, Називається кроком. Наприклад, крок один означає, що ви переміщуєте фільтр за один піксель за раз. Конвенція — це крок у два пікселя.

Глибина фільтра, як зазначили Жегеді та Ліу у свої роботі [17], повинна бути такою ж, як і глибина вводу, тому, якби розглядалось кольорове зображення, глибина була б 3. Це робить розміри цього фільтра 5x5x3. У кожному положенні фільтр множить значення у фільтрі з початковими значеннями в пікселі. Це елементарне множення. Множення підсумовуються, створюючи єдине число. Якщо почати з верхнього лівого кута обгортки з бульбашками, це число відповідає лівому верхньому куту. Тепер ви переміщуєте фільтр у наступне положення і повторюєте процес через усю обгортку з бульбашками. Масив, який отримаємо, називається картою збудження або картою активації. Можна використовувати більше одного фільтра, який допоможе краще зберегти просторові відносини.

Потрібно вказати такі параметри, як кількість фільтрів, розмір фільтру, архітектура мережі тощо. CNN вивчає значення фільтрів самостійно під час навчального процесу. Є безліч варіантів, з якими можна працювати, щоб зробити найкращий класифікатор зображень для вашого завдання. Можна вибрати матрицю заповнену нулями (нульове заповнення), щоб застосувати фільтр до граничних елементів матриці вхідного зображення. Це також дозволяє контролювати розмір карт збудження. Додавання нульового заповнення — це широке згортання. Якщо не додавати нульового заповнення, це вузьке згортання.

Результатом цього є згорнена карта об’єктів. Вона менше вихідного вхідного зображення. Це полегшує та пришвидшує працю. Втрачається інфо-

рмація? Деяка, так. Але в той же час метою детектора збудження є виявлення особливостей, що саме це і робить.

1.6.2 ReLU layer (Шар зрізаних лінійних вузлів)

Шар ReLU — ще один крок до нашого шару згортки. Ви застосовуєте функцію активації на своїх картах збудження, щоб збільшити нелінійність у мережі. Це тому, що самі зображення вкрай нелінійні! Він видаляє негативні значення з карти активації, встановлюючи їх на нуль.

Згортання — це лінійна операція з такими речами, як помноження та додавання матриць. Дані в реальному світі, які потрібно дізнатись від CNN, будуть нелінійними. Це може використовуватися за допомогою такої операції, як ReLU. Ви можете використовувати інші операції, такі як гіперболічний тангенс або сигмоїдна функція. Однак ReLU є популярним вибором, оскільки він може швидше навчити мережу без суттєвих покарань за точність узагальнення.

1.6.3 Pooling (агрегування)

Агрегування поступово зменшує розмір вхідного подання. Це дозволяє виявляти об’єкти на зображенні незалежно від того, де вони знаходяться. Агрегування допомагає зменшити кількість необхідних параметрів і обсяг необхідних обчислень. Це також допомагає контролювати перенавчання.

Перенавчанні може бути подібним до того, як ви запам’ятовуєте надто детальні деталі перед тестом, не розуміючи інформації. Запам’ятовуючи деталі, ви можете чудово справитись із своїми картками вдома. Однак ви не справитесь зі справжнім тестом, якщо вам буде представлена нова інформація.

Ще один приклад: якщо у всіх собак у ваших даних про дресирування є плями та темні очі, ваша мережа вважатиме, що для того, щоб зображення було класифіковано як собака, воно повинно мати плями та темні очі. Якщо ви перевіряєте свої дані з тими самими даними навчання, це зробить дивови-

жну роботу з правильної класифікації собак. Але якщо ваші результати — лише "собака" і "кішка", а у вашій мережі представлені нові зображення, що містять, скажімо, ротвейлера та хаскі, це, ймовірно, буде класифікуючи як котів ротвейлера, так і хаскі.

Без різноманіття ваша мережа буде марною із зображеннями, які точно не відповідають навчальним даним. Завжди дані про навчання та тестування окремо. Якщо ви тестуєте з даними, на яких тренувались, інформація у вашій мережі запам’ятовується. Це зробить жахливу роботу, коли буде представлено будь-які нові дані.

Отже, для цього кроку ви берете карту збуджень, застосовуєте шар агрегування, і результатом є агрегована карта збуджень.

Найпоширенішим прикладом агрегування є максимізаційне агрегування. При максимізаційному агрегуванні вихідне зображення розподіляється на набір областей, які не перекриваються. Вихідні дані кожної області — це максимальне значення в кожній області. Це дає нам менший розмір з меншою кількістю параметрів.

Максимізаційне агрегування — це отримання максимального значення в кожному місці зображення. Це позбавляє нас від 75% інформації, яка не є збудженням. Беручи максимальне значення пікселів, ви враховуєте деформацію. Якщо функція обертається трохи вліво або вправо або що завгодно, агреговане збудження буде таким самим. Ви зменшуєте розмір і параметри. Це чудово, оскільки це означає, що модель не буде перенавчена цією інформацію.

Ви можете використовувати усереднювальне агрегування або агрегування сум, але вони не є загальним вибором. Максимізаційне агрегування, як правило, працює ефективніше, ніж обидва на практиці. При максимізаційному агрегуванні ви берете найбільші значення пікселів. При усереднювальному агрегуванні ви берете середнє значення всіх значень пікселів у цьому місці зображення.

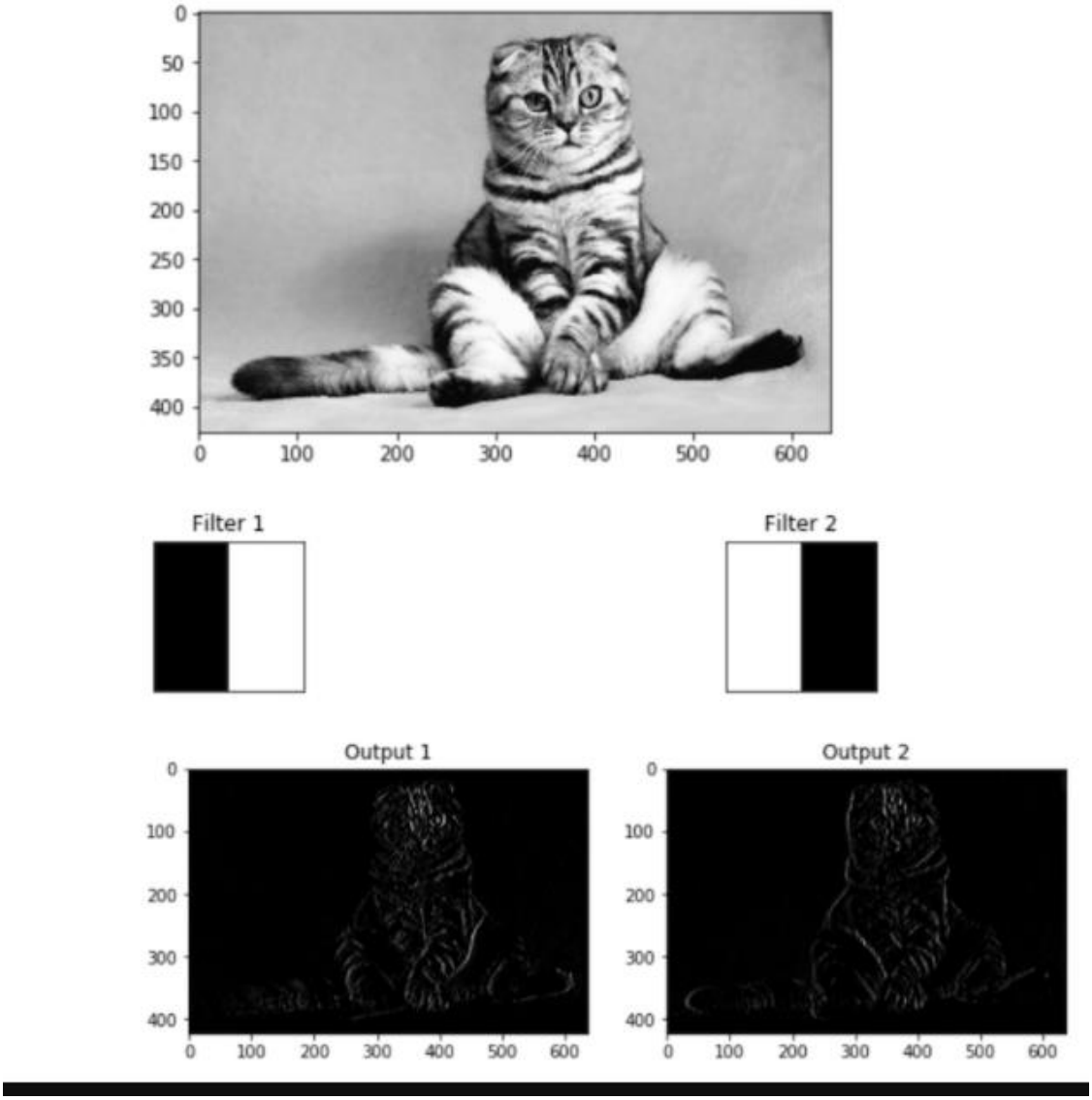

Рис. *6 Візуалізація роботи шару агрегації*

1.6.4 Flattening (згладження)

Це досить простий крок. Ви згладжуєте агреговану карту збуджень у послідовний стовпець чисел (довгий вектор). Це дозволяє цій інформації стати вхідним шаром штучної нейронної мережі для подальшої обробки.

1.6.5 Повноз'єднаний шар

На цьому кроці додається штучна нейронна мережа до нашої згорткової нейронної мережі.

Основною метою штучної нейронної мережі є поєднання наших збуджень у більшу кількість атрибутів. Вони прогнозуватимуть класи з більшою

точністю. Це поєднує збудження та атрибути, які можуть краще передбачити класи.

На цьому кроці помилка обчислюється, а потім зворотньо розмножується. Детектори ваги та збуджень налаштовані для оптимізації роботи моделі. Потім процес відбувається знову і знову.

Як працюють вихідні нейрони, коли їх більше одного?

По-перше, необхідно зрозуміти, які ваги застосовувати до синапсів, які підключаються до виходу. Потрібно знати, які з попередніх нейронів важливі для виходу.

Якщо, наприклад, у вас є два вихідних класи, один для кота і один для собаки, нейрон, який читає 0, абсолютно не впевнений, що збудження належить коту. Нейрон, який читає 1, абсолютно впевнений, що ознака належить коту. У останньому повністю зв’язаному шарі нейрони зчитують значення від 0 до 1. Це означає різні рівні достовірності. Значення 0,9 означало б достовірність 90%.

Нейрони кота, які визначені при визначенні ознаки, знають, що зображення — це кішка. Кажуть, математичний еквівалент: «Це мої нейрони. Мене слід спрацьовувати» Якщо це трапляється багато разів, мережа дізнається, що коли деякі функції запускаються, зображення стає кішкою.

Через безліч ітерацій котячий нейрон дізнається, що коли деякі збудження викликаються, зображення стає кішкою. Нейрон собаки (наприклад) дізнається, що коли загоряються деякі інші функції, зображення стає собакою. Нейрон собаки дізнається, що, наприклад, знову нейрон "великого мокрого носа" і нейрон "гнучкого вуха" вносять з великою кількістю впевненості в нейрон собаки. Це надає більшу вагу нейрону "великого мокрого носа" та нейрону "гнучкого вуха". Нейрон собаки вчиться більш-менш ігнорувати нейрон "вусів" та нейрон "котячої райдужки". Котячий нейрон вчиться надавати більшу вагу таким нейронам, як "вуса" та "котяча райдужка".

Після того, як мережа пройде навчання, ви зможете передати зображення, і нейронна мережа зможе визначити ймовірність класу зображення для цього зображення з великою кількістю впевненості.

Повноз'єднаний шар — це традиційний багатошаровий персептрон. Він використовує класифікатор у вихідному рівні. Класифікатор, як правило, є функцією активації softmax. Повноз'єднаний означає, що кожен нейрон попереднього шару з’єднується з кожним нейроном наступного шару. Яка мета цього шару? Використовувати функції вихідних даних попереднього рівня для класифікації вхідного зображення на основі навчальних даних.

1.7 Порівняння інструментів розробки та навчання нейронних мереж

1.7.1 Найпопулярніші мови програмування нейронних мереж

Python — популярна мова з високоякісними бібліотеками машинного навчання та аналізу даних. Мова загального призначення, яка, як зазначено у книжці Роуссума [18], віддає перевагу своїй читабельності, хорошій структурі та відносно м'якій кривій навчання, продовжує набирати популярності. Згідно з щорічним опитуванням розробників, проведеним Stack Overflow, проведеним у січні, Python можна назвати найбільш швидкозростаючою основною мовою програмування. Вона посідає сьоме місце за популярністю (38,8%), і зараз на крок випереджає C # (34,4%).

C ++ — мова середнього рівня, що використовується для паралельних обчислень на CUDA. C ++ це, як булу сказано Едісоном Вілсі у його праці [19] гнучка, об'єктно-орієнтована, статично типізована мова, заснована на мові програмування C. Мова залишається популярною серед розробників завдяки надійності, продуктивності та великій кількості доменів програм, які вона підтримує. C ++ має як високий, так і низький рівні мови, тому він вважається мовою програмування середнього рівня. Ще однією перевагою цієї мови є розробка драйверів та програмного забезпечення, які можуть безпосе-

редньо взаємодіяти з апаратним забезпеченням у режимі реального часу. І оскільки C ++ досить чистий для пояснення основних понять, його використовують для досліджень та викладання.

R — мова для статистичних обчислень та графіки. Сільвія Тіпман у свої статі писав [20], що це мова та середовище для статистики, візуалізації та аналізу даних, є найкращим вибором для науковців. Це ще одна реалізація мови програмування S. R та бібліотеки, написані в ньому, надають численні графічні та статистичні методи, такі як класичні статистичні тести, лінійне та нелінійне моделювання, аналіз часових рядів, класифікація, кластеризація тощо. Ви можете легко розширити мову за допомогою R-пакетів машинного навчання. Мова дозволяє створювати високоякісні графіки, включаючи формули та математичні символи.

1.7.2 Фреймворки для загального машинного навчання

NumPy — це пакет розширень для виконання числових обчислень за допомогою Python, який, цитуючи книжку Тревіса Оліфанта [21], замінив NumArray та Numeric. Він підтримує багатовимірні масиви (таблиці) та матриці. Дані ML представлені в масивах. А матриця — це двовимірний масив чисел. NumPy містить функції трансляції як інструменти для інтеграції C / C ++ та коду Fortran. Його функціональність також включає перетворення Фур'є, лінійну алгебру та можливості випадкових чисел. Фахівці в галузі даних можуть використовувати NumPy як ефективний контейнер для зберігання багатовимірних загальних даних. Завдяки можливості визначати довільні типи даних, NumPy легко та швидко інтегрується з численними видами баз даних.

scikit-learn — проста у використанні система машинного навчання для багатьох галузей. scikit-learn — це бібліотека машинного навчання Python з відкритим кодом, побудована поверх SciPy (Науковий Python), NumPy та matplotlib. Бібліотека призначена для виробничого використання. Простота, якісний код, варіанти співпраці, продуктивність та велика документація, на-

писана зрозумілою мовою, сприяють його популярності серед різних фахівців. scikit-learn надає користувачам ряд усталених алгоритмів для контрольованого та неконтрольованого навчання.

1.7.3 ML фреймворки для моделювання нейронних мереж

TensorFlow — гнучкий фреймворк для масштабного машинного навчання. Це, як її описали у статі Діллон та Лангмор [22], бібліотека програмного забезпечення з відкритим кодом для машинного навчання та глибоких досліджень нейронних мереж, розроблена та випущена Google Brain Team в рамках організації AI від Google у 2015 році.

Суттєвою особливістю цієї бібліотеки є те, що числові обчислення виконуються за допомогою графіків потоків даних, що складаються з вузлів і ребер. Вузли представляють математичні операції, а ребра — це багатовимірні масиви даних або тензори, на яких виконуються ці операції.

TensorFlow є гнучким і може використовуватися на різних обчислювальних платформах (центральних процесорах, графічних процесорах та TPU) та пристроях, від робочих столів до кластерів серверів до мобільних та граничних систем. Він працює на Mac, Windows і Linux.

PyTorch — простий у використанні інструмент для дослідження. Це фреймворк машинного навчання з відкритим кодом для глибоких нейронних мереж, які підтримуються та прискорюються за допомогою графічних процесорів. Бібліотека, розроблена командою Facebook спільно з інженерами з Twitter, SalesForce, NRIA, ENS, ParisTech, Nvidia, Digital Reasoning та INRIA, була вперше випущена в жовтні 2016 року. PyTorch був розроблений з ідеєю надати якомога швидший та гнучкий досвід моделювання. Варто згадати, що робочий процес у PyTorch схожий на робочий процес у NumPy, науковій обчислювальній бібліотеці на базі Python.

Keras — легка, проста у використанні бібліотека для швидкого створення прототипів. Це бібліотека глибокого навчання на Python, яка може працювати поверх Theano, TensorFlow або CNTK. Член команди Google Brain

Франсуа Шолле розробив його, щоб надати вченим даних можливість швидко проводити експерименти з машинного навчання. Бібліотека може працювати на GPU і CPU і підтримувати як періодичні, так і згорткові мережі, а також їх комбінації. Швидке створення прототипів можливо за допомогою зрозумілого інтерфейсу бібліотеки високого рівня, поділу мереж на послідовності окремих модулів, які легко створювати та додавати.

1.8 Огляд датасетів для навчання моделі

CIFAR-100 — він є позначеними підмножинами набору даних із 80 мільйонів крихітних зображень. він має 100 класів, що містять по 600 зображень у кожному. У класі є 500 навчальних зображень та 100 тестових зображень. 100 класів у CIFAR-100 об’єднані у 20 суперкласів. Кожне зображення має "м’яку" мітку (клас, до якого воно належить) та "грубу" мітку (суперклас, до якого воно належить).

ImageNet — це база даних зображень, організована відповідно до ієрархії WordNet (наразі лише іменники), в якій кожен вузол ієрархії зображений сотнями та тисячами зображень. Проект сприяв розвитку комп'ютерного зору та дослідженням глибокого навчання. Дані доступні для дослідників безкоштовно для некомерційного використання.

1.9 Огляд існуючих рішень задачі просування контенту

1.9.1 Phototag — Hashtag Generato

Цей застосунок вирішую поставлену нами задачу, а саме:

- Підбирає теги до зображення.
- Дозволяє створювати пости.

Проте можна виділити суттєві недоліки застосунку:

- Не може підібрати доречні теги з інстаграму додатково до тих що були розпізнані нами.

- Редагування тегів обмежене.

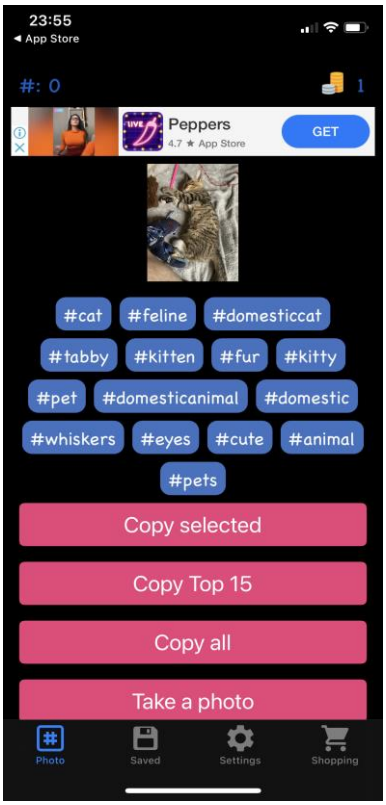

Рис. *7 Користувацький інтерфейс застосунку Phototag — Hashtag Generator*

1.9.2 keywordtool.io

Сайт keywordtool.io вирішує лише одну із поставлених задач: допомагає знаходити доречні теги до підораного вами ключового слова. Натомість такі задач застосунок вирішити не може:

- Підібрати теги до знімку.
- Створити пост.

Як висновок можна сказати що використання цього сайту є невиправданим у нашому випадку.

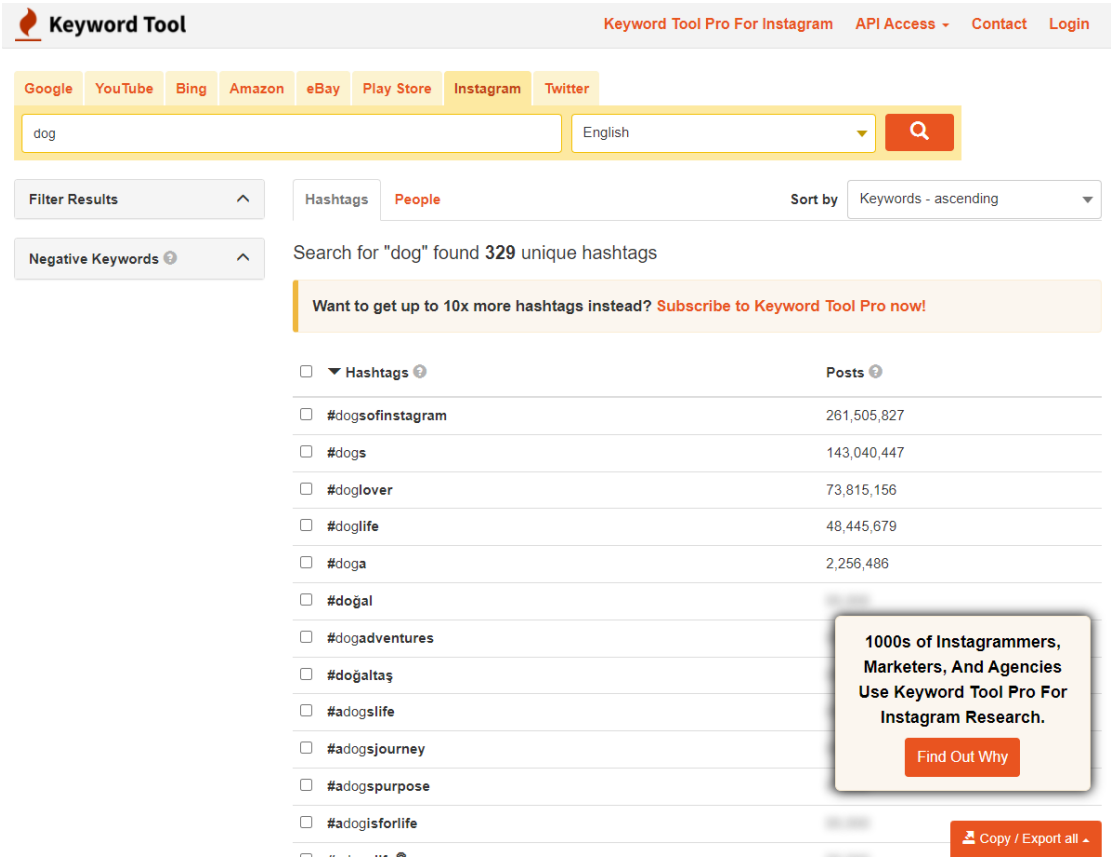

Рис. *8 Користувацький інтерфейс keywordtool.io*

1.9.3 Brandfolder

Функціонал сайту Brandfolder є наступним:

1. Звантажити зображення
2. Знайти на зображенні теги

Цей функціонал дозволяє вирішити одну із поставлених перед нами задач, проте інші не в змозі, такі як

- Створи пост у соціальній мережі
- Підібрати доречні хетеги

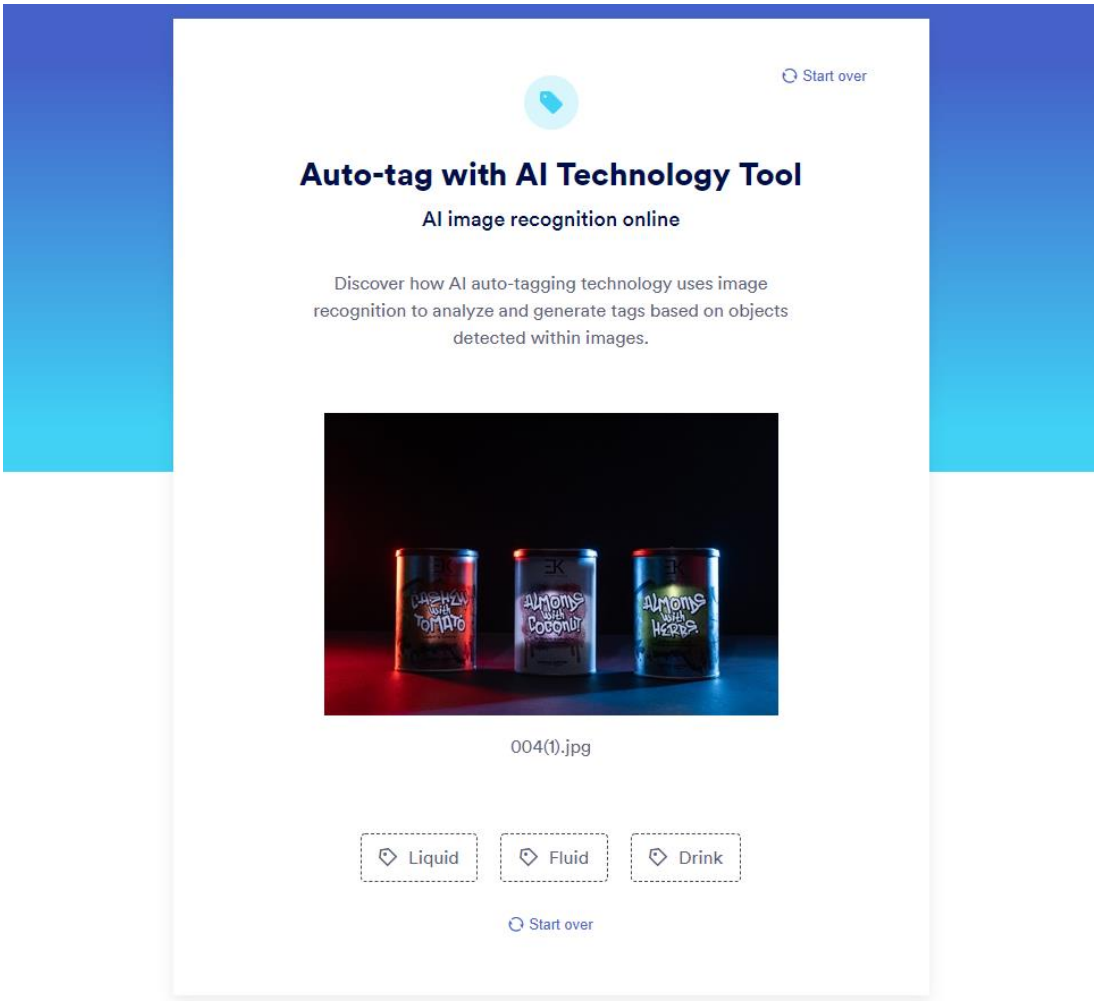

Рис. *9 Користувацький інтерфейс Brandfolder*

1.9.4 Висновки із аналізу конкуруючих рішень

На поточний час нема єдиного застосунку який би міг одночасно виконувати наступні вимоги:

1. Обробляти зображення та знаходити теги.
2. Підібрати до оброблених зображень хештеги.

Це дає змогу зробити висновок що застосунок, який заплановано розробити, є актуальним та не буде загублений серед конкурентів.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО РОЗРОБКУ МОБІЛЬНИХ ЗАСТОСУНКІВ ТА НЕЙРОНИХ МЕРЕЖ

2.1 Процес розробки мобільного застосунку за допомогою Flutter

2.1.1 Декларативний підхід до розробки користувацького інтерфейсу

Більшість розробників звикла до того що зазвичай раніше більшість фреймворків від Win32 до Android та iOS використовували імперативний підхід до розробки UI. Імперативний стиль може бути саме тим до чого ви найбільш звикли: ви власноруч створюєте повноцінний UI об’єкт та згодом маніпулюєте його зовнішнім виглядом за допомогою різних методів із коду.

Як відбувається маніпулювання UI у декларативному фреймворку:

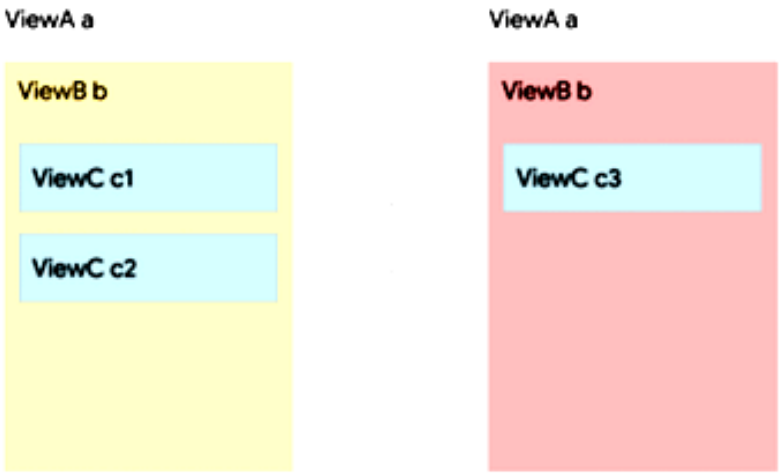

Рис. *10 Приклад зміни UI*

Уявіть собі що вам необхідно прийти від стану застосунку зліва, до стану справа (змінити колір фону. У імперативному стилі це потребую напряму звернутися до ViewFirst та змінити його вручну:

Лістинг*2.1 Приклад коду у імперативному програмуванні*

```
first.setColor(red)
first.clearChildren()
ViewThird third = new ViewThird(...)
first.add(third)
```

Як було описано Андерсоном та Алексоном у їх спільній книжці [23], у декларативному стилі вам непотрібно звертатися до представлень аби змінити зовнішні вигляду. Ваше зовнішнє представлення є незмінним, екран може перебудуватися лише тоді коли ви зміните зовнішні змінні які представлення використовує для побудови UI

Лістинг *2.2 Приклад коду у декларативному програмуванні*

```
return ViewSecond
(
  color: someColorVar,
  child: someChildVar,
)
```

2.1.1 Застосунок у Flutter — це дерево віджетів

Користувацький інтерфейс застосунку на Flutter будується з віджетів. Це сучасний підхід який був натхнений React. Віджети описують як повинно виглядати представлення відповідно до поточної конфігурації та стану застосунку. Коли стан віджету змунюється, віджет перебудовується на екрані наново. Фреймворк порівнює зміни що відбулися аби мінімізувати затрату потужності девайсу для рендеру.

Віджет може бути кінцевим та відображати інформацію (наприклад віджет що відображає текст), або включати в себе інші віджети.

Віджет може бути кінцевим та відображати інформацію (наприклад віджет що відображає текст), або включати в себе інші віджети.

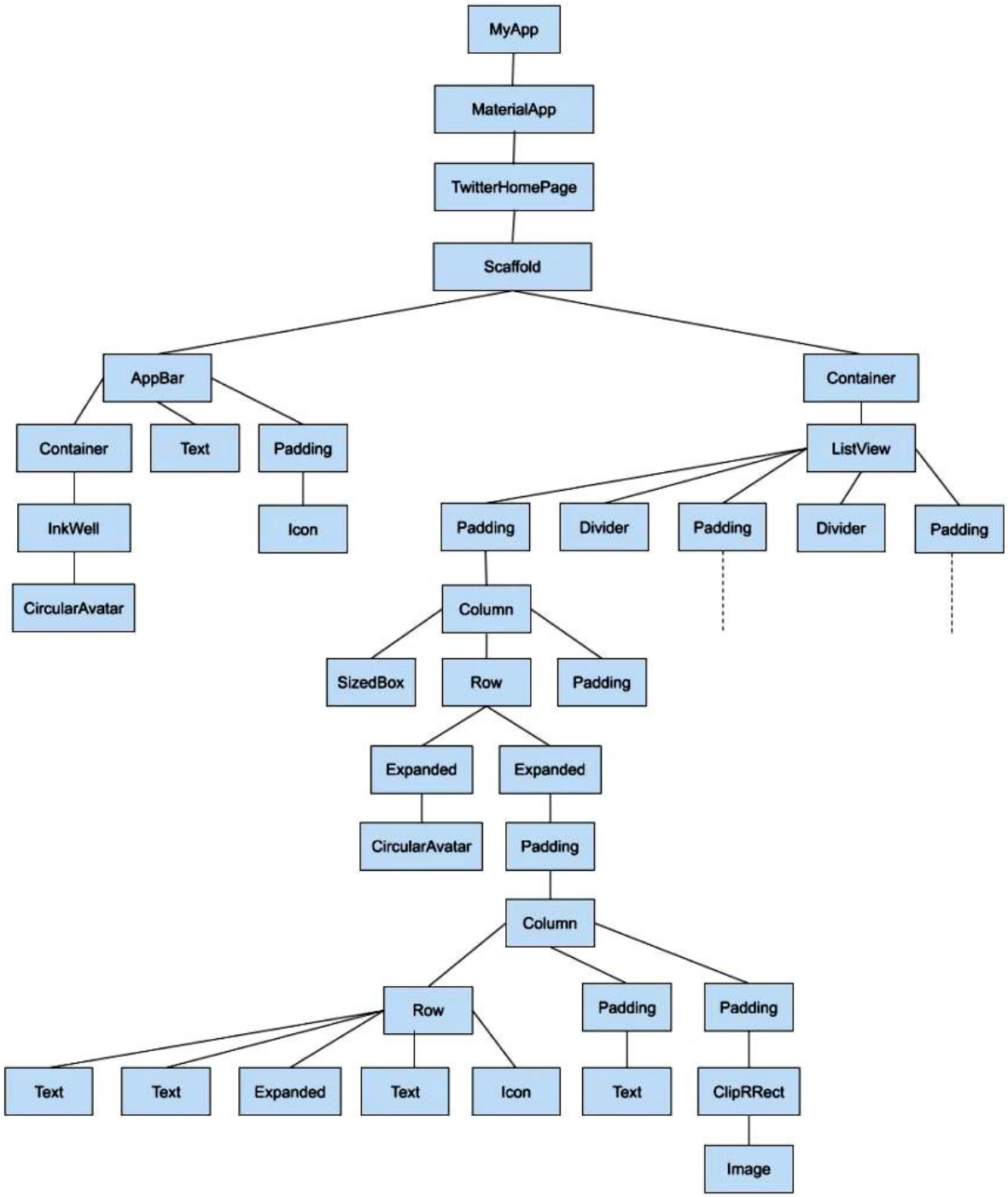

Рис. *11 Приклад дерева віджетів*

2.1.2 Менеджмент станів застосунку

Виходячи із праці Дмитра Слепнева [24], стани застосунку можна поділити на два типи:

1. Локальні стані.
2. Глобальний стан.

Локальні стани. Суть локального стану полягає у тому що він зберігається лише у одному віджету застосунку. Можна навести декілька прикладів:

- Обраний номер сторінки у віджету перегляду сторінок.
- Поточний стан якоїсь комплексної анімації.
- Обраний номер вкладки навігації.

Стан є самодостатнім, інші віджети у дереву не мають необхідності у доступі до цих даних. Вони не повинні зберігатись або якось змінюватись.

Глобальний стан. Суть глобального стану (або стану застосунку) полягає у тому що доступ до цього стани бажається мати із декількох різних частин застосунку, а також бажаємо зберегти упродовж декількох запусків застосунку.

Деякі приклади глобальних станів:

- Налаштування застосунку.
- Інформація про акаунт користувача.
- Обрані товари у онлайн магазині.
- Обрані новини у інформаційному застосунку.

2.2 Процес розробки нейронних мереж

Основні концепції. Перед тим як починати розробку нейронної мережі потрібно розглянути основні концепції нейронних мереж, які наглядно були продемонстровані у праці Джейн та Мао [25].

Нейрон. Нейрон є базовою структурною одиницею у побудові нейронних мереж.

Вибір алгоритму навчання. Нейронні мережі можна поділити на два основних типи за принципом того, чи має алгоритм у процесі навчання зворотній зв’язок:

- Навчання з учителем — алгоритму надається приклад того, які дані подаються на вхід та який бажаємо отримати вихід, які задає так названий «вчитель».

- Навчання без учителя — ніяк не класифікуємо вхід алгоритму, натомість він сам має знайти закономірність даних у вході. З цього виходить що метою розробки таких алгоритмів може бути й як саме навчання (аби виявляти скриті залежності даних входу) або для досягнення іншої мети.

2.2.1 Збір та класифікація даних

Першим кроком для реалізації алгоритму навчання із вчителем є збір та класифікація даних. У випадку згорткової нейронної мережі потрібно визначити категорії та знайти відповідні до них зображення. Є необхідним зробити два набору зображень:

1. Тренувальні зображення
2. Контрольні (тестові) зображення

Перший набір використовується для навчання нашої нейронної мережі, а другий для того щоб контролювати результат навчання.

2.3 Штучне збільшення даних для навчання

Це технологія яка широко використовується у машинному навчанні яка, як її описували Дук та Менг [26], дозволяє нам підвищити кількість даних для навчання додаючи до оригінального набору даних видозмінені копії даних або штучно створені дані на основі поточних даних.

2.3.1 Штучне збільшення даних для розпізнавання зображень

Штучно збільшити дані для розпізнавання зображення можна за допомогою двох наступних підходів:

Видозміна картинок. До цього виду штучного змінення можна віднести:

- Видозміна форми та розміру.

- Повороти зображення.
- Зміни кольорів.
- Обрізання зображення.
- Перевороти зображення.
- Додавання шуму до зображення.
- Видалення випадкових частин зображення.

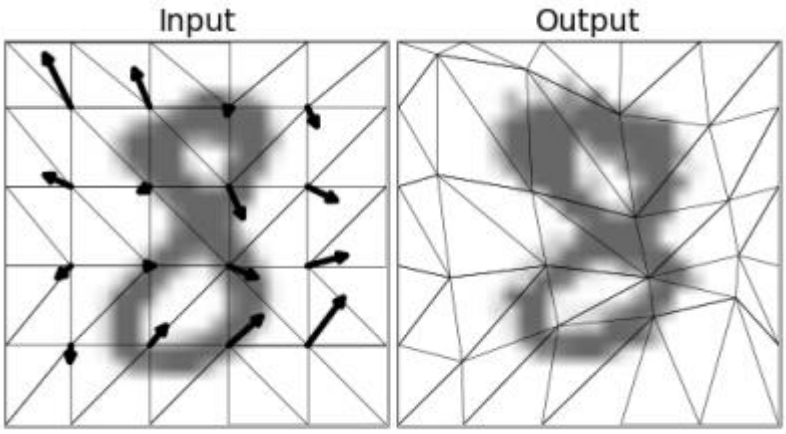

Рис. *12 Приклад того як можна відозмінити зображення*

Додавання нових згенерованих зображень. Якщо наш датасет замалий для того щоб вирішита поставлену задачу, то технологія видозмінення може представити на просте та ефективне вирішення проблеми нехватки даних. Проте якщо датасет усе одно замалий

2.4 Розробка алгоритму навчання

Наступним кроком є розробка алгоритму нашої нейронної мережі. Розробник може обирати.

Теоретичні відомості про згорткові нейронні мережі. Згорткові нейронні мережі базуються на концепціях істотного біологічного процесу — зору (тобто те за якою схемою у тварин з’єднуються нейрони зорової кори). У

головній корі окремі нейрони області зору тільки в обмеженій області реагують на імпульси. Уся область зору частково покрита полями відчуттів різних нейронів.

Порівнюючи CNN та інші алгоритми обробки зображень можна зробити висновок що у CNN менше попередньої обробки даних. Якщо у інших підходах була необхідність розробити вручну фільтри до зображень, то у нашому алгоритми мережа сама цьому навчається. Це дає нам більшу переваг бо наша мережа є незалежною від людського фактору або попередніх знань.

Convolutional neural network (згорткова нейронна мережа) є алгоритмом який навчається із вчителем.

2.5 Архітектура згорткової нейронної мережі

Є декілька підходів за допомогою яких ви можете розробляти згорткову нейронну мережу

LeNet. Згідно зі статтею Ел-Саві та Хазема р[27], одна з перших реалізації згорткової нейронної мережі. Складається усього із 5 шарів із параметрами що навчаються (із цього пішла назва мережі).

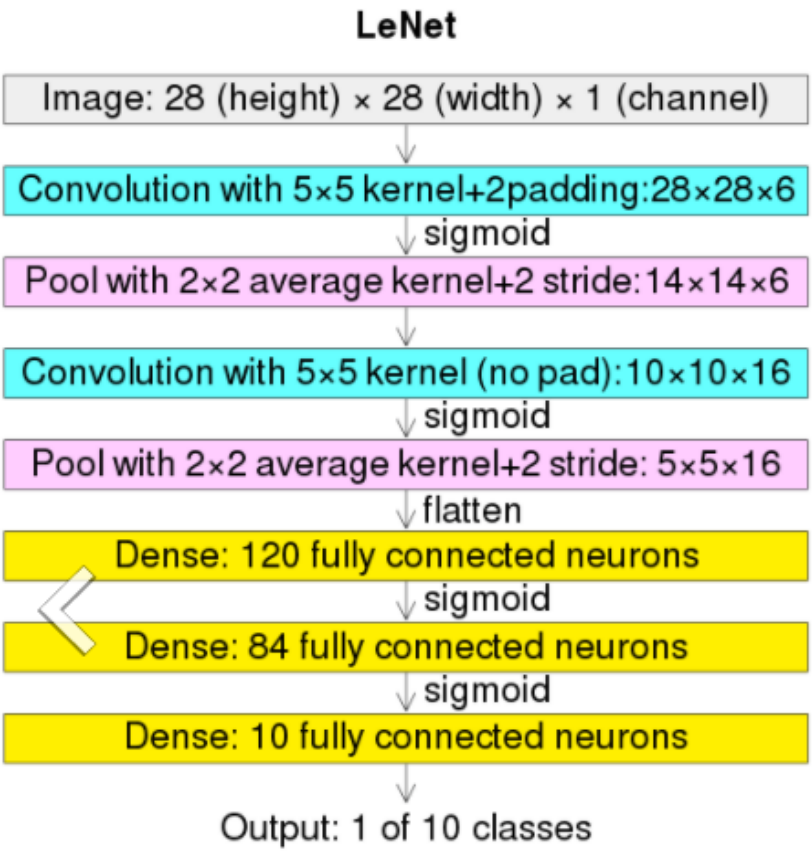

Рис.*13 Класична LeNet архітектура*

AlexNet. Розроблена у 2012. Базується, виходячи зі статті Ю, Янга та Баі [28], на п’яти згорткових шарах які в свою чергу починаються із ядра розміром 11 на 11. Була розроблена із орієнтацією на 1000 можливих класів зображень.

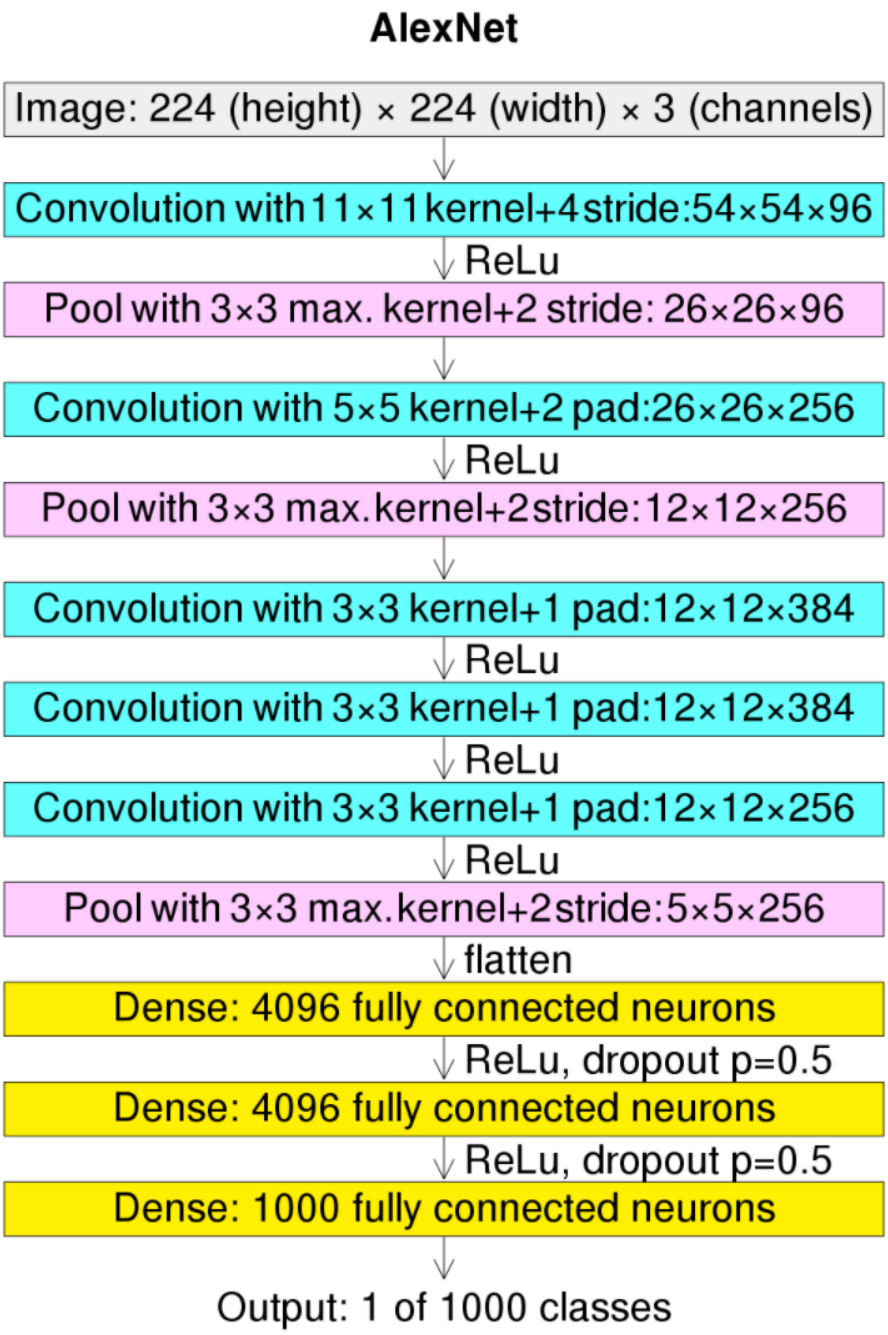

Рис. *14 Класична AlexNet архітектура*

VGG. Вперше архітектура з'явилася у статі «Very Deep Convolutional Networks for Large-Scale Image Recognition» у 2014 році [29]. Там були вперше представлені дослідження які підтвердили що додавання більшої кількості шарів підвищує ефективнім згорткової нейронної мережі.

Архітектура має дві реалізації:

1. VGG-16 — складається із 16 вагових шарів.
2. VGG-19 — складається із 19 шарів.

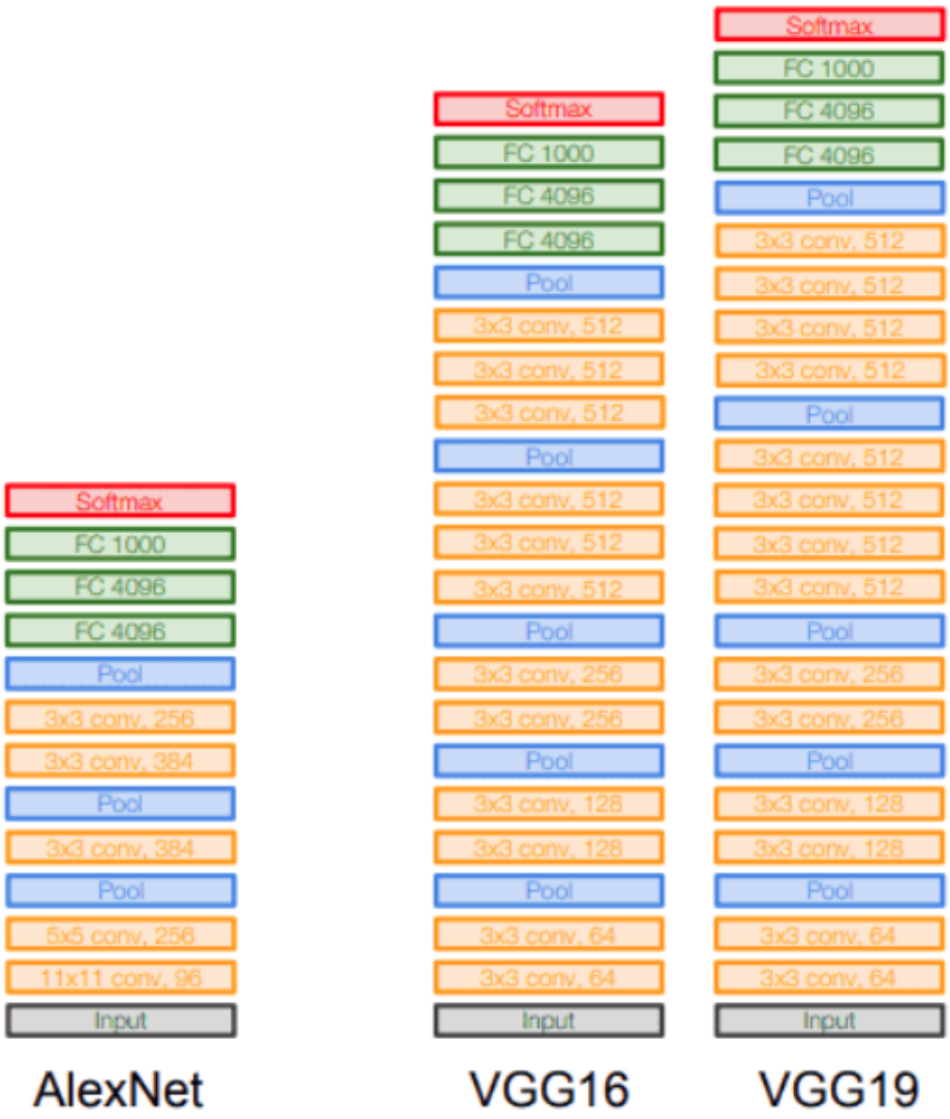

Рис. *15 Візуальне порівняння архітектури AlexNet та VGG*

Inception and GoogLeNet (2014 рік). Згідно із статтею Камардіна та Сяріфа [30], перше презентували шари згортання розміром 1x1. Їх наміром є досягнути розширення мережі не тільки у глибину за рахунок росту кількості шарів, а й у ширину.

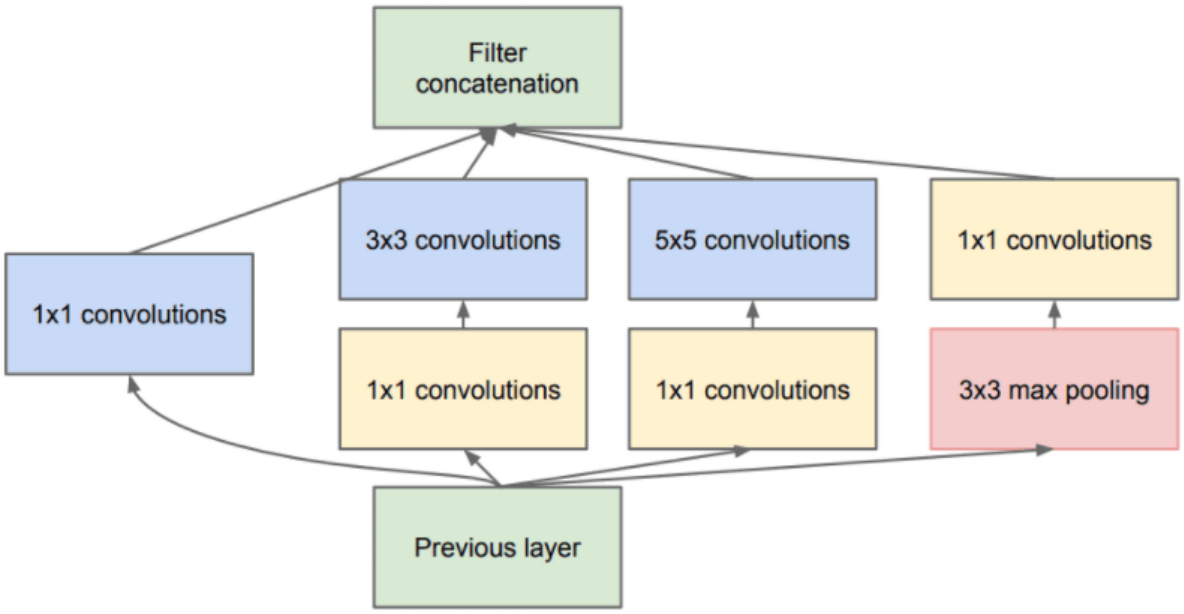

Рис.*16 Схема архітектури GoogLeNet/InceptionNet*

Inception версії 2 та 3. У 2015 році автори GoogLeNet запропонували певний перелік покращень до архітектури:

- Замінти згортковий шар 5x5 двома шарами 3x3.
- Розбити ядро 3x3 на два ядра 1x3 та 3x1.
- Була додана пакетна нормалізація.

ResNet (2015). Архітектура що може динамічно змінювати кількість шарів від 18 до 150.

2.6 Висновки про розробку застосунків та нейронних мереж

Аналіз теоретичних відомостей показав, що для реалізації нейронної мережі недостатньо самого алгоритму. Не менш важливим є зібрати та правильно підготувати навчальні та тренувальні дані. Були з’ясовані основні принципи й підходи сучайсної мобільної розробки з використанням Flutter.

Наступним кроком є вибір інструмента для побудови нейронної мережі та застосування ключових принципів Flutter для розробки необхідного застосунку.

РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Архітектура системи

Архітекутра системи складається із двох головних частин:

1. Мобільного застосунку.
2. Нейронної мережі.

У мобільному застосунку обираємо зображення яким хочемо поділитися, далі передаємо його у нейронну мережу на обробку, яка повертає нам список тегів. Далі тегами разом із зображенням можем поділитися у соціальній мережі.

Рис.*17 Архітектура системи*

3.1.1 Архітектура мобільного застосунку

Мобільний застосунок є проектом на Flatter який складається з наступних частин:

Нативні проекти (іос та андроїд) — потрібні для використання фреймворку на кожній з платформ. Також вони виконують дві суттєві функції у нашому проекті:

1. Дозволяють зробити індивідуальні для кожної з платформ налаштування проекту, наприклад налаштування компіляції проекту, вихідного застосунку для цієї платформи і т. д.
2. Дозволяють нам розробити модулі на нативному рівні, наприклад для того щоб використовувати API системи до яких нема доступу через Flatter.

Вихідний код — ключова складна проекту, наш код на мові Dart. Кожен Flatter проект є деревом віджетів, У випадку цього проекту можна виділити наступні структуру проекту:

1. Вступний екран.
2. Головний екран.
3. Екран зворотного зв’язку.

Pubspec налаштування — це перелік налаштувань, який включає до себе:

- Обгорненні спільні нативні налаштування платформ, а саме версію та назву проекту.
- Перелік бібліотек Flatter які використовуються у проекті.

Ассети — це папка що включає до себе зображення які використовуються у застосунку, а також текст який використовується та його переклад.

Тести — тести для нашого застосунку (Unit тести тощо).

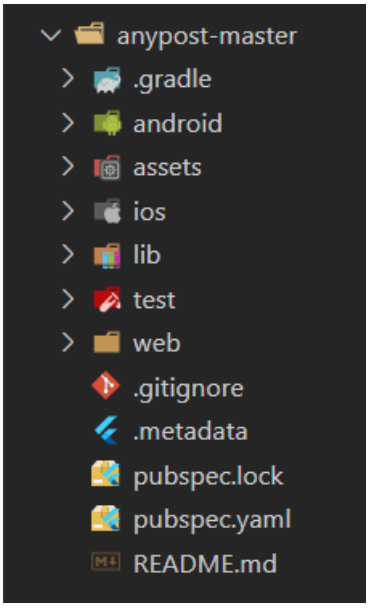

Рис. *18 Структура застосунку*

3.1.2 Архітектура проекту нейронної мережи

Проект розробки нейронної мережі класифікації зображень складається з наступних частин:

Датасет — за допомогою обраного нами датасету наша нейронна мережа має змогу навчитися поділяти зображення на попередньо обрані класи. У випадку нашого проекту використовується датасет CIFAR-100.

Python проект — проект на мові програмування пайтон, де за допомогою бібліотек з навчання можна навчити модель.

Вихідна модель — навчена нами модель класифікації зображень. Є універсальною та не пов’язаною з нашим мобільним застосунком, може бути використана де забажаємо.

3.1.3 Архітектура розробленої моделі

Архітектура нейронної мережі має наступний вигляд:

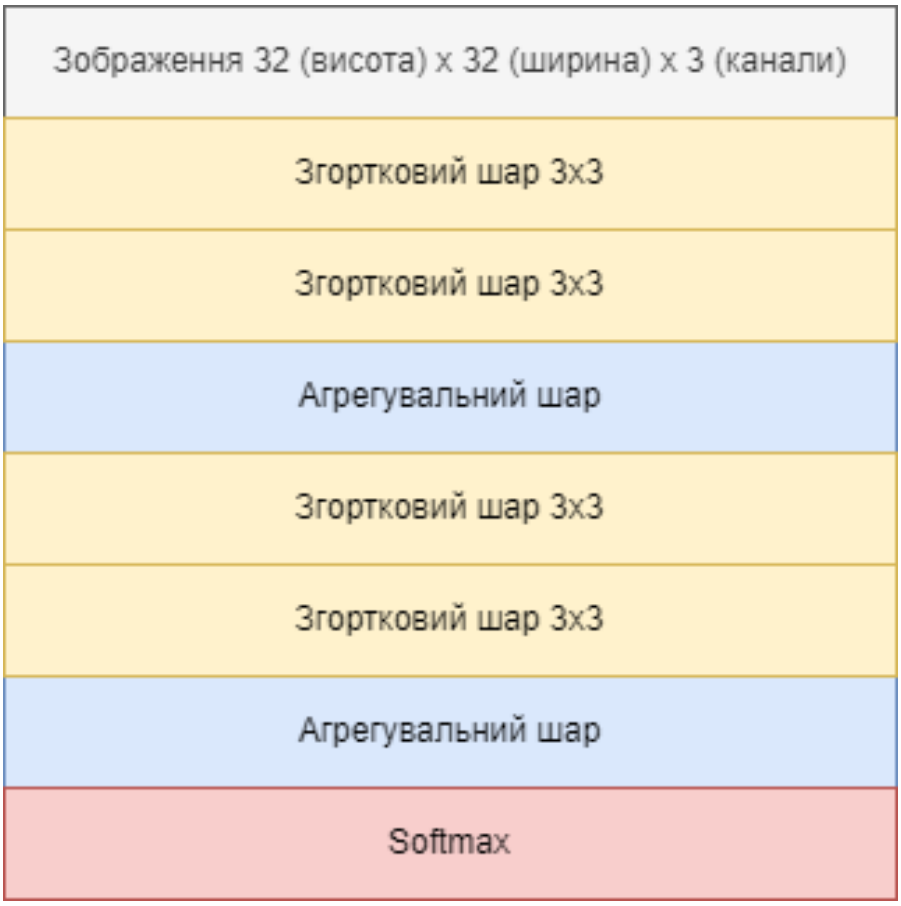

Рис.*19 Архітектура нейронної мережі*

Побудову архітектури нейронної мережі можна представити наступними кроками:

1 крок: Завантажуємо наш датасет для навчання

На першому кроці підготовляємо завантажену із інтернету модель до використання.

Функція за допомогою якої десереалізується завантажена із Інтернету модель.

Лістинг *3.1 Функція завантаження датасету*

```
def unpckl(some_file):
    import pickle
    with open(some_file, 'rb') as fo:
        some_dct = pickle.load(fo, encoding='bytes')
    return some_dct
```

2 крок: Ділимо зображення на відповідні категорії

1. Meta.
2. Train.
3. Test.

3 крок: Загружаємо класи для розпізнавання

Classes = pd.DataFrame(meta[b'some_lbl_names'],columns = ['clsss'])

4 Збільшення даних

За допомогою бібліотеки збільшуємо дані видозміненими зображеннями. У нашому випадку збільшуємо кількість зображень у 10 раз

Із одного датасету робимо 10

Лістинг *3.2 Функція розширення датасету*

```
sequnces = iaa.Seqntial([
    iaa.Fliplr(0.5),
    iaa.CrpAndPad(px=(-2,
2),sample_independently=True,pad_mode=["constant",
"edge"]),
    iaa.Affne(shear=(-10, 10),mode =
['symmetric','wrap']),#48
    iaa.Add((-5, 5)),
    iaa.Multiply((0.8, 1.2)),
],random_order=True)
Ourdata1 = sequnces.augment_images(ourdata)
Ourdata2 = sequnces.augment_images(ourdata)
Ourdata3 = sequnces.augment_images(ourdata)
Ourdata4 = sequnces.augment_images(ourdata)
Ourdata5 = sequnces.augment_images(ourdata)
Ourdata6 = sequnces.augment_images(ourdata)
Ourdata7 = sequnces.augment_images(ourdata)
Ourdata8 = sequnces.augment_images(ourdata)
Ourdata9 = sequnces.augment_images(ourdata)
Ourdata10 = sequnces.augment_images(ourdata)
```

5 крок: Змішуємо дані у купу

6 крок: Рандомізуємо дані

7 крок: Загружаємо дані у модель

8 крок: Навчання моделі

Модель навчається базується на VGG архітектурі згорткової нейронної мережі.

Лістинг *3.3 Модель навчання нейронної мережі*

```
cnv_1 = cnv _layer(x,shape=[3,3,3,32])
cnv_2 = cnv _layer(cnv _1,shape=[3,3,32,64])
cnv _2_pooling = max_pool_2by2(cnv _2)
cnv _3 = cnv _layer(cnv _2_pooling,shape=[3,3,64,128])
cnv _4 = cnv _layer(cnv _3,shape=[3,3,128,256])
cnv _4_pooling = max_pool_2by2(cnv _4)
cnv _2_flat = tf.reshape(cnv _4_pooling,[-1,8*8*256])
full_layer_one = tf.nn.relu(normal_full_layer(cnv
_2_flat,1024))
full_one_dropout =
tf.nn.dropot(full_layer_one,hold_prob)
y_pred = normal_full_layer(full_one_dropot,num_cls)
```

Повторюємо процес навчання доти доки точність не перевісить 53 відсотки (у нашому випадку знадобилося близько 13200 повторень)

9 крок: Порівнюємо справжні результати із передбаченими моделлю

10 крок: Імпортування моделі у мобільний застосунок

Навчена модель у сирому вигляді ще не може використовуватися у мобільному застосунку. У застосунку для роботи з нейронною мережею використовується бібліотека Google ML Kit.

3.2 Апаратні та програмні вимоги

3.2.1 Вимоги до мобільного пристрою

Android платформа — мінімальною версією на якій можна запустити наш застосунок Android 5.0, більш старі версії не підтримуються.

iOS платформа — мінімальною версією є iOS 13

3.2.2 Вимоги до комп'ютера у проекті нейронної мережі

Комп'ютер на якому планується запустити наш проект мусить мати наступні характеристики:

- Python 3.7–3.9
    - Для підтримки Python 3.9 потрібен TensorFlow 2.5 або пізнішої версії.
    - Для підтримки Python 3.8 потрібен TensorFlow 2.2 або пізнішої версії.
- pip 19.0 або пізнішої версії (потрібна підтримка багатьох Linux 2010)
- Операційна система:
    - Ubuntu 16.04 або пізнішої версії (64-розрядна версія).
    - macOS 10.12.6 (Sierra) або пізнішої версії (64-розрядна версія). (без підтримки графічного процесора).
    - Для macOS потрібен pip 20.3 або пізнішої версії.
    - Windows 7 або пізнішої версії (64-розрядна версія).
- Microsoft Visual C++, що розповсюджується для Visual Studio 2015, 2017 та 2019.
- Для підтримки графічного процесора потрібна карта з підтримкою CUDA ® (Ubuntu і Windows).

Вимоги до обладнання:

- Починаючи з TensorFlow 1.6, двійкові файли використовують інструкції AVX, які можуть не виконуватися на старих процесорах.

- Прочитайте посібник з підтримки GPU, щоб налаштувати відеокарту з підтримкою CUDA ® в Ubuntu або Windows.

3.2.3 Вимоги до надійності

Відсутність програмних збоїв, можливість скористатися усіма функціональними можливостями застосунку використовуючи будь-яку фотографію.

3.2.4 Вимоги до продуктивності

Повинні виконуватися наступні умови продуктивності:

- Обробка зображення та виділення хештегів не може займати більше 10 секунд
- Розмір застосунку +не може перевищувати 100 мегабайт
- Частота кадрів з якою працює застосунок не може бути нижчою за 30 кадрів на секунду
- Застосунок мусить працювати коректно без подключення до мережі

3.3 Засоби розробки

3.3.1 Засоби розробки мобільного застосунку

image_picker — бібліотека для обирання зображення користувачем, яке у подальшому буде використовуватися та оброблюватися. Наразі використовується лише на платформі андроїд бо має суттєвий недолік на платформі іос. Проблема полягає у тому, щоб аби надалі була змога поділитися у інстаграм обраним за допомогою цієї бібліотеки зображенням на іос, нам потрібно зробити копію зображення. Це є суттєвою проблемою з позиції користувацького досвіду, тому було зроблене рішення розробити кастомний swift модуль обирання зображення на іос, що використовує більш сучасний підхід до обирання зображення та дозволяє уникнути створення копій.

Лістинг *3.4 Кастомний модуль вибору зображення на ioc*

```
extension AppDelegate: PHPickerViewControllerDelegate {
    // TODO: Handle error cases
    func picker(_ picker: PHPickerViewController, did-
FinishPicking results: [PHPickerResult]) {
        picker.dismiss(animated: true, completion: nil)

        guard let result = self.result else {
            return;
        }

        for imageResult in results {
            if imageRe-
sult.itemProvider.canLoadObject(ofClass: UIImage.self)  {
                imageRe-
sult.itemProvider.loadObject(ofClass: UIImage.self) {
(newImage, error) in
                    if let error = error {

print(error.localizedDescription)
                    } else {
                        do {
                            guard let imageURL =
NSURL(fileURLWithPath: NSTemporaryDirecto-
ry()).appendingPathComponent("temp_image.png"), let image =
newImage as? UIImage, let data = image.png() else {
                                return
                            }
                            try data.write(to: im-
ageURL)
                            let pickerResult = Im-
agePickerResult(assetId: imageResult.assetIdentifier, im-
ageUrl: imageURL.path)
```

```
                                    let resultJsonData = try
JSONEncoder().encode(pickerResult)
                                    let resultJsonString =
String(data: resultJsonData, encoding: .utf8)
                                    result(resultJsonString)
                                } catch { }
                            }
                        }
                    } else {
                        print("Loaded Assest is not a Image")
                    }
                }

                self.result = nil
            }
        }
```

Лістинг *3.5 Функції що відповідають за обирання фотографії у затосунку, за допомогою бібліотеки або кастомного модулю відповідно*

```
      void _pickImageIOS() async {
        final resultJSON = await plat-
form.invokeMethod('pickImage');
        if (resultJSON != null) {
          final result = PickerRe-
sult.fromJson(jsonDecode(resultJSON));
          setState(() {
            imageCache?.clear();
            _pickerResult = result;
            _imageLabeling();
          });
        }
      }
```

```
      void _pickImageAndroid() async {
         final pickedFile = await _picker.getImage(source:
ImageSource.gallery);
        if (pickedFile != null) {
          setState(() {
             _pickerResult = PickerResult(imageUrl: picked-
File.path);
            _imageLabeling();
          });
        }
      }
```

easy_localization — це бібліотека за допомогою якої реалізується функція перекладу нашого застосунку.

Лістинг *3.6 Віджет який реалізує функцію перемикання мови застосунку*

```
  class LanguagePicker extends StatelessWidget {
    const LanguagePicker({
      Key? key,
    }) : super(key: key);

    @override
    Widget build(BuildContext context) {
      return DropdownButtonHideUnderline(
        child: DropdownButton(
          dropdownColor: K.COLOR_WEB_TEXT,
          icon: Image.asset(
            "assets/images/dropdown_icon.png",
            width: 10,
            height: 5,
          ),
          value: context.locale.languageCode,
          items:
```

```
                <String>['en',
'ru'].map<DropdownMenuItem<String>>((String value) {
                return DropdownMenuItem<String>(
                  value: value,
                  child: PrimaryText(
                    text: value.toUpperCase(),
                    shouldTranslate: false,
                  ),
                );
              }).toList(),
              onChanged: (String? newValue) {
                if (newValue != null) {
                  context.setLocale(Locale(newValue));
                }
              },
            ),
          );
        }
      }
```

`Fastlane` – бібліотека для автоматизації створення та відправки на тестування нових версій застосунку.

`google_ml_kit` — бібліотека-обгортка для Flatter, що надає доступ до нативних реалізацій бібліотек Google's ML Kit на платформа андроід та іос. Дозволяє нам вирішити задачу класифікації зображень за допомогою вбудованої моделі.

Лістинг *3.7 Метод що використовує google_ml_kit та виконує класифікацію зображень*

```
      void _imageLabeling() async {
        if (_pickerResult != null) {
          setState(() {
            _isLoading = true;
```

```
          });

          final InputImage inputImage = In-
putImage.fromFilePath(_pickerResult!.imageUrl);
          final imageLabeler = Goog-
leMlKit.vision.imageLabeler();
          final List<ImageLabel> labels = await imageLa-
beler.processImage(inputImage);

          setState(() {
            _groups = labels;
            _fetchAllTags();
          });
          imageLabeler.close();
        }
      }
```

http — бібліотека для використання мережевих запитів. Є потреба у цій бібліотеці аби за знайденими категоріями зображення знайти популярні хештеги у соціальній мережі.

Лістинг *3.8 Функція у якій використовуючи знайдені категорії робимо запит до соціальної мережі*

```
  void _fetchTag(String tag) async {
    setState(() {
      _additionalIsLoading = true;
    });
    final hashtag = (tag[0] == '#' ? tag :
'#$tag').toLowerCase();
    final queryParameters = {'query': hashtag};
    final uri = Uri.https(K.INST_BASE_URL,
K.INST_SEARCH_REQ, queryParameters);
```

```
        try {
          final response = await http.get(uri);
          final relevantTags =
              Instagram-
TagsList.getHashtagsList(jsonDecode(response.body), tag);
          setState(() {
            _groupsTags[tag] = Tuple2(relevantTags, true);
            _additionalIsLoading = false;
          });
        } catch (err) {
          _showFetchErrorDialog([tag]);
          setState(() {
            _groupsTags[tag] = Tuple2([], true);
            _additionalIsLoading = false;
          });
        }
      }
```

Firebase Crashlytics — інструмент за допомогою якого можна аналізувати проблеми у роботі нашого застосунку. Принцип роботи наступний: якщо у використанні користувачем трапилась помилка у робі застосунку та воно непередбачено закрилося, тоді цей сервіс автоматично посилає дані про збій до меню, де можема подивитися надану нам інформацію, а саме:

- Час збою.
- Назва та модель пристрою.
- Власно помилка у коді.

Якщо збій не є випадковістю, а присутній як серйозний баг який часто зустрічають користувачі при використанні застосунку, то отримуємо повідомлення на електронну пошту що нам варто звернути увагу на збій який трапляється регулярно та заважає нормальній роботі застосунку.

Google Analytics — це перелік метрик які збираються із користувачів нашого застосунку. За допомогою їх збору та аналізу можна звернути увагу на слабкі місця нашого застосунку. Ось декілька із найбільш корисних метрик для використання:

- Скільки часи у середньому користувач проводить у застосунку (час утримання користувача)
- На якому екрану користувач закриває застосунок
- Країна та мова користувача
- Яку версію застосунку використовую користувач

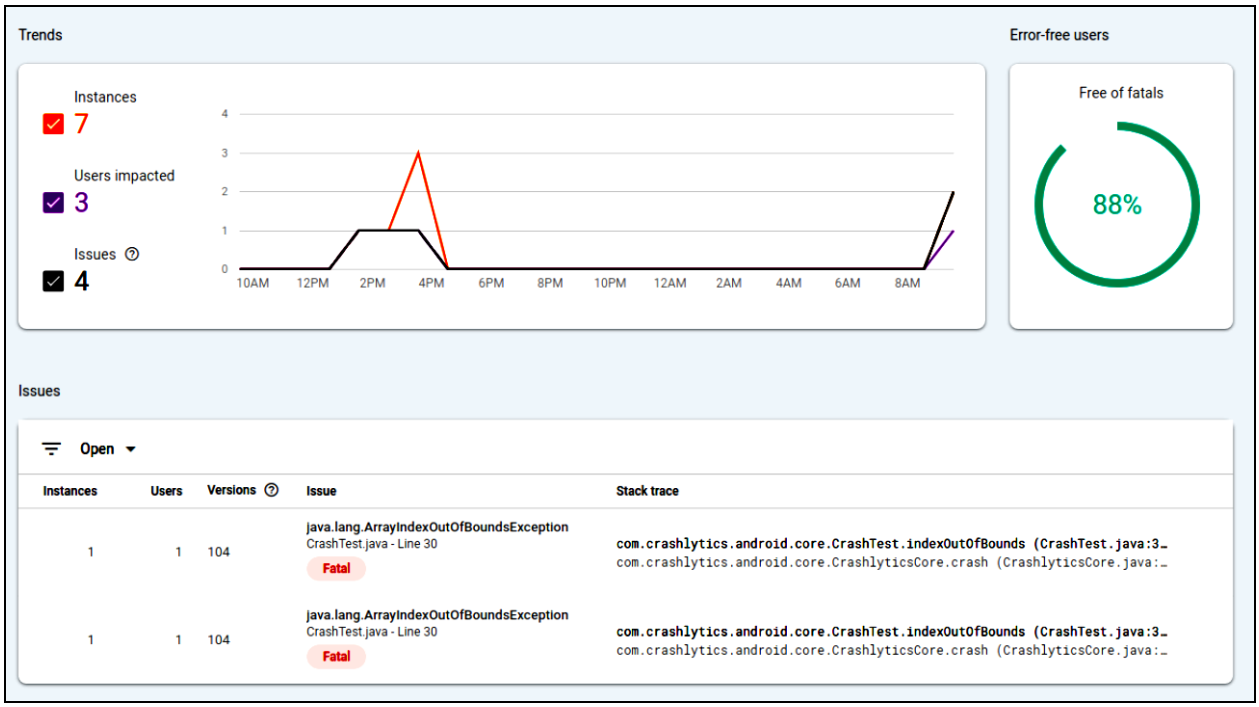

Рис. *20 Панель використання Firebase Crashlytics*

Firebase App Distribution — засіб для надання нових версій застосунку тестовим користувачам

3.3.2 Засоби розробки нейронної мережі

Jupyter Notebook — це сучасна середа розробки на Python створена спеціально для дата-саєенс. У цій середі ви відразу можете бачити результат виконання коду або його окремих фрагментів.

TensorFlow — це платформа для машинного навчання із відкритим вихідним кодом. Вона має гнучку екосистему різних бібліотек та ресурсів що дозволяють значно прискорити процес написання нейронних мереж.

Tensorflow

Placeholders are tensorflow's variables that do not require initial values

In [36]:
```
x = tf.placeholder(tf.float32,shape=[None,32,32,3])
y_true = tf.placeholder(tf.float32,shape=[None,num_class])
```

In [37]:
```
hold_prob = tf.placeholder(tf.float32)
```

Helper functions for initializing layers

In [38]:
```
def init_weights(shape):
    init_random_dist = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(init_random_dist)

def init_bias(shape):
    init_bias_vals = tf.constant(0.1, shape=shape)
    return tf.Variable(init_bias_vals)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2by2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

def convolutional_layer(input_x, shape):
    W = init_weights(shape)
    b = init_bias([shape[3]])
    return tf.nn.relu(conv2d(input_x, W) + b)

def normal_full_layer(input_layer, size):
    input_size = int(input_layer.get_shape()[1])
    W = init_weights([input_size, size])
    b = init_bias([size])
    return tf.matmul(input_layer, W) + b
```

In [39]:
```
num_class = 100
```

Рис. *21 Використання Tensorflow для ініціалізації шарів нейронної мережі*

CIFAR-100 — датасет для навчання який складається зі 100 класів кожен з яких містить 600 картинок (500 тренувальних та 100 тестувальних).

Рис. *22 Зображення із датасету із відповідними категоріями*

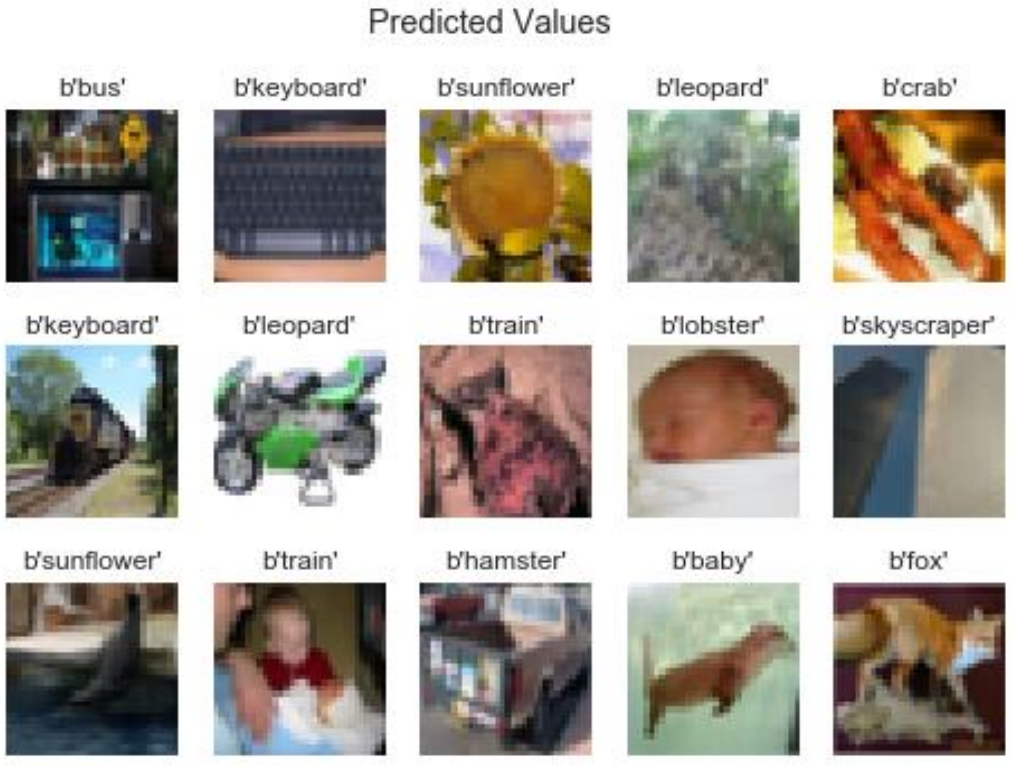

Рис. *23 Зображення із датасету із категоріями, які передбачає нейронна мережа*

Imgaug — це бібліотека яка дозволяє нам збільшити кількість даних застосунку за допомогою технології розширення кількості зображень.

За допомогою цієї бібліотеки можливо сотворити наступні маніпуляції із зображенням:

- Підвищення рівню контрасту та точності зображення.
- Афінне перетворення зображення .
- Обрізання та додавання відступів до зображення.
- Діагональний злом зображення.

Ці маніпуляції можна провести із:

- Оригінального зображання.
- Теплової мапи.
- Мапи сегментів.
- Ключових точок.
- Границь да полігонів.

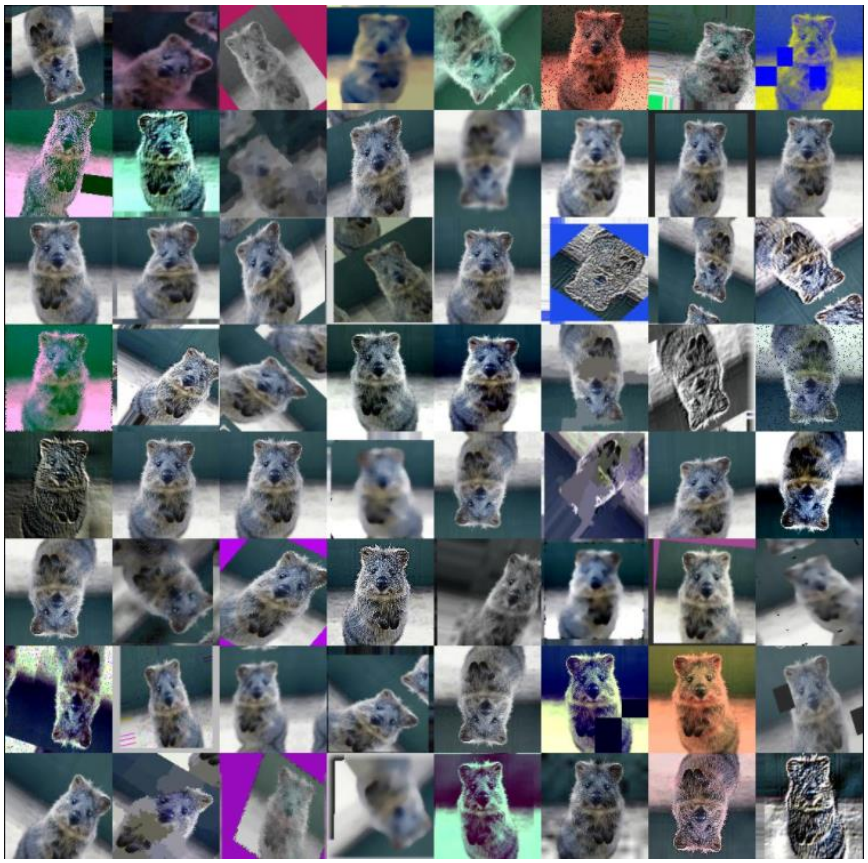

Рис. *24 Приклад відозміненого зображення за допомогою бібліотеки Imgaug*

3.4 Функціональні можливості

Функціональні можливості розробленого нами застосунку є наступними:

1. На першому етапу користувач обирає зображення яке він планує використовувати у своєму пості у соціальній мережі.
2. Далі застосунок проводить аналіз зображення та за допомогою навченої моделі нейронної мережі виділяє з нього категорії.
3. За відділеними категоріями знаходимо хештеги які є найбільш актуальним на даний час у соціальній мережі. Ці хештеги далі використаємо у нашому пості.
4. Користувач має можливість змінити набір хештегів (видалити категорії які він вважає некоректними) або змінити порядок у якому йдуть обрані хетеги
5. На фінальному етапі користувач поширює створений застосунком пост у соціальній мережі

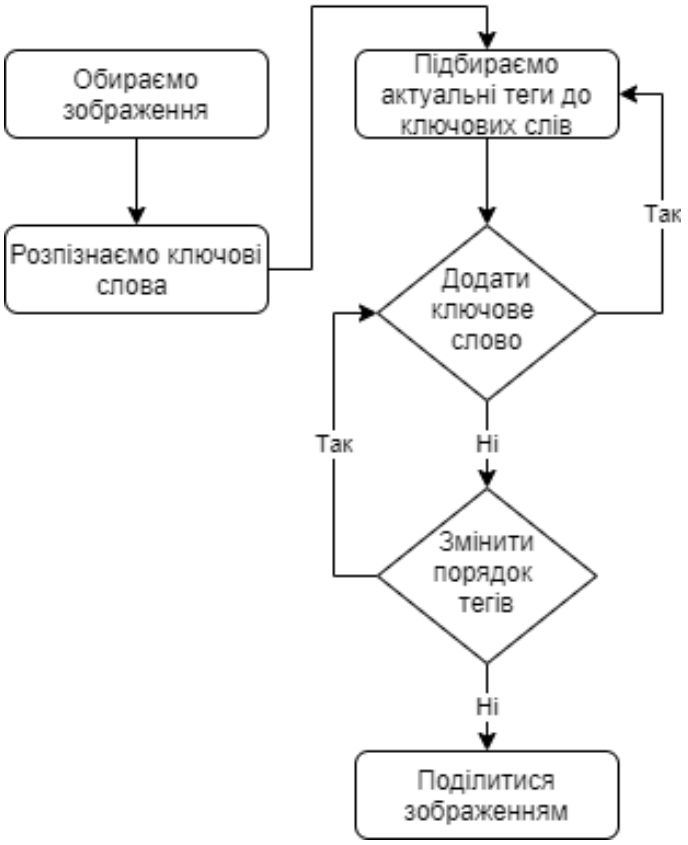

Рис. *25 Представлення можливостей застосунку у вигляді схеми*

3.5 Результати розробки програмного забезпечення

1. Був розроблений мобільний застосунок який дає змогу користувачу автоматизувати задачі SMM та спростити просування свого контенту у соціальній мережі.
2. Була розроблена та навчена нейронна мережа класифікації зображення.
3. Розроблений застосунок був доданий до Apple Store на платформі iOS та до Play Market на платформі Android.

РОЗДІЛ 4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Розробка методики дослідження проекту

Методика дослідження включає в себе:

1. Дослідження архітектури нейронної мережі.
2. Дослідження ефективності мобільного застосунку.

Нейронна мережа написана за допомогою мови Python та платформи TensorFlow та середовища розробки Jupyter Notebook із використанням датасету CIFAR-100. Мобільний застосунок буде розроблений за допомогою середовищ розробки Visual Studio Code, Android Studio, XCode. Розроблена та навчена модель нейронної мережі була інтегрована у мобільний застосунок.

Кінцевим етапом дослідження є викладення розробленого застосунку у магазини застосунків Play Market на платформі Android та App Store на платформі iOS.

Існує ряд правил, яких необхідно дотримуватися, розробляючи мобільні застосунку:

1. Знай свого користувача. Лише звертаючи увагу на цільову аудиторію свого застосунку можна досягнути кінцевого стану застосунку яке не буде перевантажене функціями які не використовуються кінцевим користувачем. Виходячи с цього правила, можна використовувати наступний підхід:
    a. Спочатку зроби перелік запланованого функціоналу.
    b. Проведи опитування серед користувачів.
    c. Реалізуй відвідні варіанти у тестовій версії.
    d. Проаналізуй статистичні метрики із застосунку.
    e. Якщо якісь із функцій не використовуються, поміркуй над тим чи варто залишати їх у кінцевому варіанті застосунку.

    Також існує аналогічний ряд правил для нейронних мереж.

- Перед розробкою мережи пройди наступні кроки:

a. Аналіз поточних процесів. Проаналізуй поточну поставлену задачу.
b. Аналіз виправданості використання нейронних мереж. Нейронні мережі стають модним трендом сьогодні, через це їх можуть використовувати для вирішення задач, де їх використання є не раціональним.
c. Аналіз питання використання апаратного забезпечення. Чи може справитися із завданням навчання робочий комп'ютер, або варто застосувати хмарну інфраструктуру для навчання.

- Сфокусуйся на кількості та якості оброблюваної інформації
- За можливості використовуй розподілене навчання:
    a. Розподілення даних — завдяки цьому можна розділити інформацію для навчання у окремі набори, які будуть використовуватися розподіленою системою у копіях нейронної мережі для пришвидшення навчання.
    b. Розподілення моделей — це намір переглянути класичну схему побудови нейронної мережі, та використовувати декілька моделей. Найбільш раціонально використання цього підходу у великих нейронних мережах.

4.2 Результати дослідження нейронної мережі

4.2.1 Розробка нейронної мережі

Нейронна мережа дозволяє класифікувати зображення на більш ніж 100 класів.

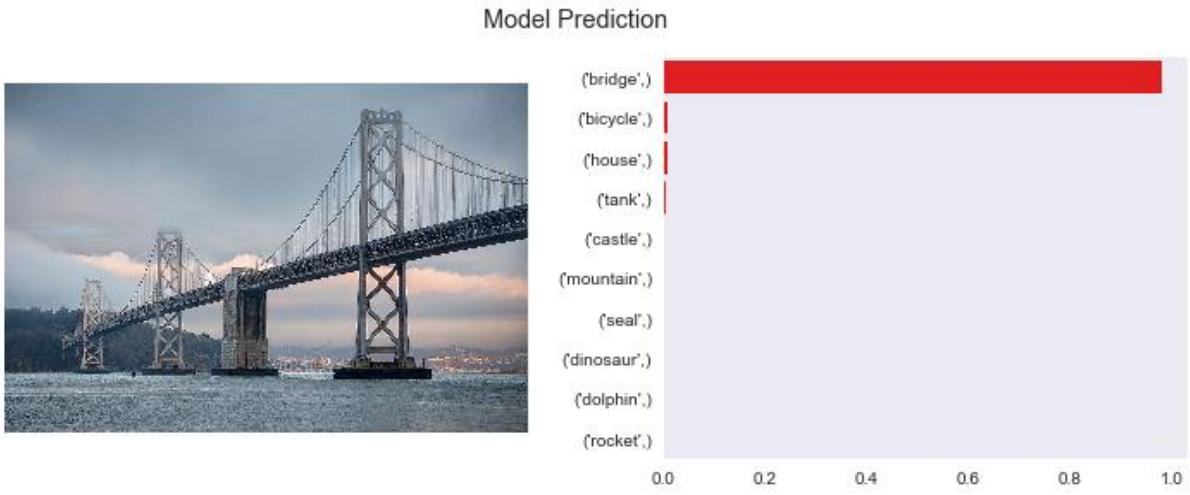

Рис. *26 Передбачення розбиття зображення на класи*

4.2.2 Порівняльний аналіз побудованої нейронної мережі із конкурентними рішеннями

Аналіз проводитися у наступному порядку:

- Обираємо три зображення.
- Аналізуємо за допомогою навченої моделі.
- Порівнюємо результати та їх точність.

Зображення 1

Зображення 2

Зображення 3

Рис. *27 Приклад вибраних зображень для порівняння методів*

4.2.3 Розроблена нейронна мережа

Мережа, яка розроблена нами проаналізувала зображення та видала наступні результати:

Таблиця *2*

Зображення 1

Bridge	97%
Bicycle	0.2%
House	0.2%
Tank	0.1%
Castle, mountain, seal, dinosaur, dolphin, rocket	<0.1%

Таблиця *3*

Зображення 2

Bicycle	99%
Motorcycle	0.1%
Snake, keyboard, crab, plate, worm, lizard, clock, telephone	<0.1%

Таблиця *4*

Зображення 3

Pine tree	99%
Beetle, bottle, palm tree, lamp, butterfly, man, boy, spider, otter	<0.1%

4.2.4 Нейронна мережа Google

Це нейронна мережа яка розроблена компанією Google та є вбудуваною за замовчення у Google ML Kit.

Таблиця *5*

Зображення 1

Bridge	99%
Dog, sky, river, vehicle, boat, pet, metal	<0.1%

Таблиця *6*

Зображення 2

Bicycle	99%
Vehicle	10%
Wheel	1%
Metal	0.2%
Tire	0.1%

Таблиця *7*

Зображення 3

Branch	60%
Plant	29%
Christmas	10%
Forest, Sky, Twig	<0.1%

4.2.5 Нейронна мережа brandfolder

Brandfolder це сайт що допомагає спеціалістам з маркетингу та креативу керувати їх робочими об’єктам.

Таблиця *8*

Зображення 1

Cloud	98%
Water	97%
Sky	93%
Gider bridge	88%
Horizon	14%

Таблиця *9*

Зображення 2

Bicycle	100%
Wheel	100%
Tire	74%
Crankset	29%
Sports Equipment	25%

Таблиця *10*

Зображення 3

Plant	100%
Larch	100%
Tree	100%
Terrestrial plant	98%
Evergreen	97%
Liquid	3%

4.2.6 Нейронна мережа imagga

Це серів який надає бізнесу який до нього звертається послуги доступу до навчених нейронних мереж

Таблиця *11*

Зображення 1

Pier	100%
Bridge	100%
Support	100%
Device	89%
City	53%
Landmark	51%

Architecture	48%
Structure	47%
River	45%
Water	40%

Таблиця *12*

Зображення 2

Cyclist	100%
Bicycle	70%
Bike	66%
Ride	35%
Sport	34%
Cycle	33%
Cycling	32%
Wheel	31%
Transport	28%
Transportation	25%

Таблиця *13*

Зображення 3

Evergreen	100%
Tree	56%
Plant	45%
Fir	40%
Branch	39%
Leaf	34%
Season	30%
Natural	26%
Environment	25%
Forest	24%

4.2.7 Висновки з аналізу нейронних мереж

Розроблена нами згорткова мережа була розроблена за допомогою платформи Tensorflow та мови програмування Python. Обраним датасетом був CIFAR-100 через те що він пропоную широку та різноманітну вибірку категорій та зображень до них. Нейронна мережа навчалася впродовж десяти годин поки не досягнула точності у 73%.

Можна зробити наступні висновки для кожної із нейронної мереж:

Розроблена нами мережа — велика точність попадання за одним ключовим словом, проте у деяких випадках точність другорядних тегів не є великою.

Нейронна мережа Google — аналогічно із розробленою нами мережею, точність за однією із категорій є великою, проте точність другорядних може піддаватися сумнівам.

Нейронна мережа brandfolder — мережа де точність другорядних тегів є високою, проте не завжди тег з найбільшою точністю є для картинки ключовим.

Нейронна мережа imagga — у випадку обраних картинок є найбільш точною мережею. Пропонує найбільше варіантів другорядних тегів які є досить великими за точністю.

Виходячи із аналізу нейронних мереж можна стверджувати, що кожна із мереж якісно продемонструвала свою роботу та може використовуватися для задачі класифікації зображень.

4.2.8 Дослідження розробленого мобільного застосунку

Створений нами застосунок дозволяє вирішити завдання прискорення праці SMM-спеціалістів. Був проведений ряд тестів що дозволяє нам зробити висновок про ефективність розробленого застосунку.

Контрольна група. До тесту були залучені чотири людини

1. Молодший спеціаліст із SMM із IT компанії.
2. Людина яка не пов’язана із маркетингом.

3. Фриланс-спеціаліст з просування контенту.
4. Розробник програмного забезпечення.

Методика тестування. Перед тестуванням були поставлені наступні задачі людям на яких проводиться тестування:

1. Створити пост у інстаграм звичайним шляхом (а саме відкрити застосунок, обрати зображення та підібрати до нього таку кількість хештегів, яка здається їм доречною для просування посту у мережі)
2. Створити пост за допомогою розробленого нами застосунку (перед тестом групі був наданий час на тестові запуски застосунку, аби ознайомитися із його функціональними можливостями).
3. Надати метрики про просування постів у мережі через три дня після викладення. Ці метрики будуть використовуватися у аналізу ефективності просування.

Таблиця *14*

Тестування створення поста у Instagram вручну

Кількість підобраних тегів до зображження	**Витрачений час на створення посту**
3	2 хв. 10 сек.
6	2 хв. 32 сек.
5	3 хв. 05 сек.
4	2 хв. 16 сек.

Таблиця *15*

Тестування створення поста у Instagram за допомогою застосунку

Кількість підобраних тегів до зображення	**Витрачений час на створення посту**
10	1 хв. 00 сек.
10	1 хв. 06 сек.
10	1 хв. 11 сек.
10	1 хв. 01 сек.

Таблиця *16*

Порівняльний аналіз кожного із постів

Номер тесту	Кількість переглядів	Кількість лайків
Інстаграм — 1	111	23
Інстаграм — 2	142	25
Інстаграм — 3	123	22
Інстаграм — 4	150	30
ANYPOST — 1	140	38
ANYPOST — 2	203	60
ANYPOST — 3	184	47
ANYPOST — 4	177	53

Виходячи із проведених тестів можна зробити висновки, що розробки досягнута та прискорено процес створення поста у соціальних мережах.

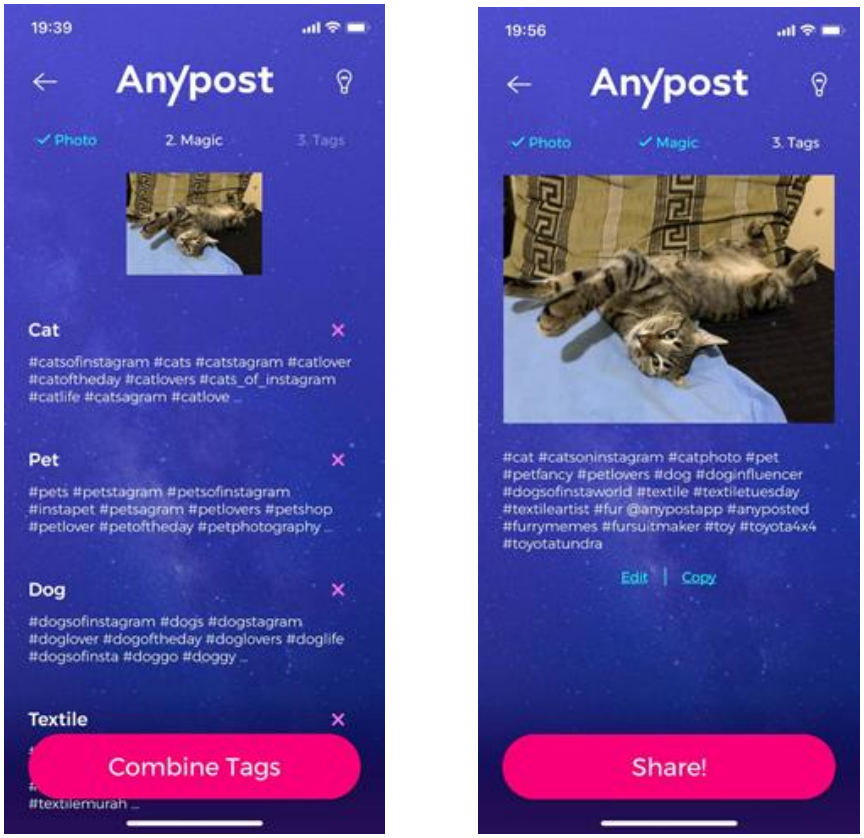

Рис. *28 Приклад генерації хештегів під час проведення дослідження*

4.3 Висновки із проведених досліджень

У ході дослідження була перевірена ідея автоматизації процесу створення та просування постів у соціальних мережах. У якості соціальної мережі для дослідження був обраний Instagram. Із результатів тестів виходить, що у середньому пости створені застосунком просувалися на 35% краще, а на їх створення було затрачено у 2,5 рази менше часу.

Як висновок можна підтвердити справедливість ідеї автоматизації задач SMM шляхом створення мобільних застосунків. Після вдалого проходження досліджень можливо перейти до використання нашого застосунку у інших соціальних мережах, наприклад Facebook та Twitter.

Рис. *29 Приклад створеного під час проведення дослідження посту*

ВИСНОВКИ

1. Встановлена і доведена актуальність розробки мобільного застосунку для SMM з використанням нейронних мереж.
2. Розроблена архітектура нейронної мережі і виконана її побудова із використанням бібліотеки Tensorflow та навчена на основі датасету CIFAR-100.
3. Для розробки архітектури мобільного застосунку обраний Flutter у сполученні із нейромережними технологіями. Для інтеграції нейронної мережі у мобільний застосунок використовувалася бібліотек Google ML Kit.
4. Проведені дослідження ефективності використання мобільного застосунку у умовах приближених до справжніх робочих задач на основі часу, який затрачувався на створення посту у соціальній мережі Instagram. Дослідження підтвердило раціональність використання, у подальшому застосунок може використовуватися у інших соціальних мережах.
5. Розроблена система додана до магазинів застосунків на кожній з платформ (AppStore на iOS та Play Market на Android).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розов Б.О., Безверхий А.І. Особливості розробки крос-платформного мобільного застосунку для SMM із використанням нейронних мереж: Матеріали І Всеукраїнської науково-практичної конференції здобувачів вищої освіти, аспірантів та молодих вчених «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України»19-21 жовтня 2021 р. Запоріжжя : ЗНУ, 2021. С. 342–343.

2. Розов Б., магістрант, Безверхий А. І., доцент — науковий керівник. Особливості розробки крос-платформного мобільного застосунку для SMM використанням нейронних мереж. *Молода наука-2021* : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2021. Т. 5. С. 96–97.

3. S Adu-Gyamfi, From Motorola Dynatac to Apple iPhone 10 plus: responses on the use of mobile telephony technology from a University community, 2017. URL: https://repository.mruni.eu/bitstream/handle/007/15209/4754-10589-1-SM.pdf?sequence=1 (дата звернення 01.05.2021).

4. D Tilson, C Sorensen, K Lyytinen. Change and control paradoxes in mobile infrastructure innovation: the Android and iOS mobile operating systems cases, 2012. URL: https://ieeexplore.ieee.org/abstract/document/6148683 (дата звернення 05.05.2021).

5. A Belenko, D Sklyarov. Evolution of iOS Data Protection and iPhone Forensics: from iPhone OS to iOS 5, 2011. URL: https://papers.put.as/papers/ios/2011/bh-ad-11-Belenko-iOS_Data_Protection.pdf (дата звернення 10.05.2021).

6. M Palmieri, I Singh, A Cicchetti. Comparison of cross-platform mobile development tools, 2012. URL: https://www.researchgate.net/profile/Inderjeet-Singh-30/publication/261315380_Comparison_of_cross-plat-

form_mobile_development_tools/links/54ae4b600cf2828b29fcce0d/Comparison-of-cross-platform-mobile-development-tools.pdf (дата звернення 15.05.2021).

7. Heitkötter H., Hanschke S., Majchrzak T.A. Evaluating cross-platform development approaches for mobile applications, 2012. URL: https://www.scitepress.org/Papers/2012/39045/39045.pdf (дата звернення 20.05.2021).

8. Xanthopoulos S., Xinogalos S. A comparative analysis of cross-platform development approaches for mobile applications, 2013. URL: https://dl.acm.org/doi/abs/10.1145/2490257.2490292 (дата звернення 25.05.2021).

9. Griffith C. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova. Sebastopol, CA: O'Reilly Media, 2017, P. 1–23.

10. Anderson N.J. Getting started with NativeScript, Birmingham : Packt Publishing Ltd., 2016. ISBN 978-1-78588-865-6, P. 1–15

11. Eisenman B. Learning react native: Building native mobile apps with JavaScript, Sebastopol, CA: O'Reilly Media, 2015, P. 1–66.

12. Kuzmin N., Ignatiev K., Grafov D. Experience of Developing a Mobile Application Using Flutter, 2019. URL: https://link.springer.com/chapter/10.1007/978-981-15-1465-4_56 (дата звернення 01.06.2021).

13. Wenhao Wu. React Native vs Flutter, Cross-platforms mobile application frameworks, Metropolia University of Applied Sciences, 2018. URL: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1 (дата звернення 06.06.2021).

14. Ghaboussi J., Garrett Jr. H., Wu X. Knowledge-based modeling of material behavior with neural networks, 1991. URL: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%290733-9399%281991%29117%3A1%28132%29 (дата звернення 11.06.2021).

15. Zhang X., Zhou X., Lin M., Sun J. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, P. 6848–6856.

16. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), P. 84–90.

17. Szegedy C., Liu, W., Jia Y., Sermanet P., Reed S., Anguelov D., Rabinovich A. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, P. 1–9.

18. Van Rossum, Guido, and Fred L. Drake Jr. Python tutorial. Vol. 620. Amsterdam: Centrum voor Wiskunde en Informatica, 1995, P. 1–12.

19. Alexandrescu, Andrei. Modern C++ design: generic programming and design patterns applied. Addison-Wesley, 2001. P. 1–10.

20. Tippmann S. Programming tools: Adventures with R. Nature News, 2015, 517.7532: 109. URL: https://www.nature.com/articles/517109 (дата звернення 11.06.2021).

21. Oliphant Travis E. A guide to NumPy. Vol. 1. USA: Trelgol Publishing, 2006. P. 12–45.

22. Dillon J. V., Langmore I., Tran D., Brevdo E., Vasudevan S., Moore D., ... & Saurous, R. A. Tensorflow distributions. arXiv preprint arXiv:1711.10604, 2017. P. 1–13.

23. Andersson D., & Axelsson A. Evaluation of The Software Development Process for A Multi-Platform Solution in Flutter, 2021. P. 1–57.

24. Slepnev D. State management approaches in Flutter, 2020. P. 4–98.

25. Jain A. K., Mao J., & Mohiuddin K. M. Artificial neural networks: A tutorial. Computer, 29(3), 1996, P. 31–44.

26. Van Dyk D. A., & Meng X. L. The art of data augmentation. Journal of Computational and Graphical Statistics, 10(1), 2001. P. 1–50.

27. El-Sawy A., Hazem E. B., & Loey, M. (2016, October). CNN for handwritten arabic digits recognition based on LeNet-5. In International conference on advanced intelligent systems and informatics. Springer, Cham. P. 566–575.

28. Yu W., Yang K., Bai Y., Xiao T., Yao H., & Rui Y. (2016, June). Visualizing and comparing AlexNet and VGG using deconvolutional layers. In Proceedings of the 33 rd International Conference on Machine Learning.

29. Simonyan K., & Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. URL: https://arxiv.org/pdf/1409.1556.pdf(2014.pdf (дата звернення 11.06.2021).

30. Sam S. M., Kamardin K., Sjarif N. N. A., & Mohamed N. Offline signature verification using deep learning convolutional neural network (CNN) architectures GoogLeNet Inception-v1 and Inception-v3. Procedia Computer Science, 161, 2019, P. 475–483.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Розов Богдан Олегович, студент _2_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти bohdan.rozov@gmail.com, — підтверджую, що написана мною кваліфікаційна робота на тему «Особливості розробки крос-платформного мобільного застосунку для SMM з використанням нейронних мереж» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2021 Підпис _____ Розов Богдан Олегович

(студент)

Дата 30.11.2021 Підпис _____ Безверхий Анатолій Ігорович

(науковий керівник)