Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка мобільного додатку-помічника по приготуванню їжі з використанням технологій комп‘ютерного бачення

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи
Старший викладач
Борозенний С.О

_____

(Підпис)

“___” _____ 2021 року

Виконала студентка ІПЗ-3
Сорокопуд Ю.М.
“___” _____ 2021 року

Київ 2021

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студентки Сорокопуд Юлії Миклаївни факультету інформатики 3-го курсу
ТЕМА: Розробка мобільного додатку-помічника по приготуванню їжі з використанням технологій комп‘ютерного бачення

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

Частина 1: Аналіз предметної області. Постановка завдання курсової роботи

Частина 2: Теоретичні відомості

Частина 3: Опис реалізації програмного продукту

Висновки

Список літератури

Додатки

Дата видачі „___” _____ 2021 р. Борозенний С.О.
_____(підпис)

Завдання отримала _____
(підпис)

Календарний план виконання роботи:

Тема: **Розробка мобільного додатку-помічника по приготуванню їжі з використанням технологій комп‘ютерного бачення**

No	Назва етапу	Термін виконання	Примітка
1	Отримання завдання на курсову роботу	02.10.2020	
2	Пошук тематичної літератури	14.10.2020	
3	Огляд літератури за темою роботи	16.10.2020	
4	Проведення дослідження	17.02.2021	
5	Аналіз отриманих результатів	01.03.2021	
6	Проектування архітектури додатку	02.03.2021	
7	Створення інтерфейсу додатку	19.04.2021	
8	Створення функціоналу класифікації зображень	21.04.2021	
9	Написання текстової частити курсової роботи	11.05.2021	
10	Створення презентації	16.05.2021	
11	Здача курсової роботи	17.05.2021	

Студентка Сорокопуд Ю.М.

Керівник Борозенний С.О.

“____” _____

ЗМІСТ

Анотація

Робота була присвячена розробці iOS додатку з використанням технологій комп’ютерного бачення для полегшення користувачу пошуку рецептів за його складовими. Для розробки додатку були використані мова Swift, фреймворк Core ML, Vision, інструмент Create ML.

Ключові слова: комп’ютерне бачення, iOS

Вступ

Актуальність теми та її практичне значення

Смартфон сьогодні уже річ, без якої важко уявити своє життя. Якщо в 2012 вони були лише у 15% населення, уже в 2019 році, кожна третя людина в світі мала смартфон[1]. З кожним днем ця цифра лиш росте. Люди користуються гаджетами щодня, кожен для своїх цілей та потреб. Але в середньому, середня кількість піднять телефону за день становить 63 рази. (додаток А)

Смартфон – це рішення багатьох проблем користувача в його ж долоні. Для того щоб знайти інформацію, яка його цікавить, замовити їжу, чи поспілкуватися з людиною, що знаходиться в іншому кінці планети, достатньо просто торкнутися екрану свого телефону.

Як росте кількість користувачів, так і їх вибагливість. Тепер користувач не просто шукає рішення своєї проблеми, а вимагає її зручне та швидке вирішення. Зайві рухи, кліки, програми, сторінки, надто повільна робота чи просто неприємний інтерфейс – причини, чому сьогодні споживач може з легкістю відмовитись від користування продуктом.

Завдяки цьому сфера розробки додатків розвивається колосальними темпами. Ринок насичений різними додатками, а нові з’являються в Google Play та AppStore щодня. Так, щомісяця в реліз виходять в середньому 100 тисяч Android додатків та 40 тисяч iOS додатків. Статистика випущених Android та iOS додатків за останні два роки наведена в додатках Б та В.

Через COVID-19 та локдауни починаючи з 2020 року, люди по всьому світу стали більше проводити часу вдома. Більшість людей почали працювати віддалено, а учні перейшли на дистанційне навчання. Різко технології стали ще важливішими у нашому повсякденному житті. Тепер соціальні мережі та месенджери не просто додаткова можливість спілкування, а основна. Всі дзвінки, зустрічі, робочі наради відбуваються онлайн.

Також, люди почали активніше вести свої соціальні мережі, робити фотографії та знімати відео. Так , за даними Domo в 2020 щохвилини у додатку Facebook користувачі викладали 147 000 фотографій, в Instagram 347 222 інстаграм-історій і завантажували загальною кількість 500 годин відео на YouTube.(додаток Г)

Люди «заповнюють» світ фотографіями та відео. Їх збільшення та стрімкий розвиток машинного навчання став поштовхом до розвитку іншої технології – комп'ютерного бачення.

Ця технологія дуже сильно розвилася протягом останніх років, і уже багато де використовується. В перекладачі Google уже давно є функція розпізнання тексту, вам потрібно тільки навести на нього, а переклад одразу виведеться на екран. У нових смартфонах зазвичай функція розпізнавання обличчя використовується для розблокування телефону. В той час як в Китаї вже масово відбувається розпізнавання обличчя на вулицях, в закладах, аеропортах, а зібрані дані використовує в роботі поліція.

В сільському господарстві, медицині, спорті, виробництві та інших сферах технологія комп'ютерного бачення використовується для безпечнішої та ефективнішої роботи, адже часто комп'ютер може виявити речі, які пропустила або на які не встигла зреагувати людина. Словом, ця технологія ще молода, але уже активно застосовується в розробці різних програм. А впровадження технологій в додатки відбувається все частіше та стає легшим.

В цей самий час, через COVID-19 за останні півтора року більше половини людей стали готувати вдома частіше, ніж раніше (додаток Ґ). І у кожного з них щодня виникає питання: «що приготувати?» .

Виходячи з даної потреби користувачів та актуальності і можливості легко впроваджувати технології комп'ютерного бачення в застосунки, за мету була поставлена реалізація iOS додатку-помічника для приготування їжі з використанням цих технологій.

Структура роботи

Робота складається з вступу та трьох основних розділів, висновків, списку джерел та додатків.

У першому розділі описаний аналіз предметної області.

У другому розділі описані основні технології, які використовувались для розробки додатку, теоретичні відомості про них.

У третьому розділі описані етапи розробки та принципи роботи готового застосунку.

Постановка задачі

1. Проаналізувати додатків-аналогів даної предметної області
2. Проаналізувати принципи роботи машинного навчання та комп’ютерного бачення
3. Дослідити технології для створення моделей машинного навчання, роботи з ними та роботи з комп’ютерним бачення а межах операційної системи iOS
4. Використовуючи мову Swift та досліджені технології, створити iOS-додаток для приготування їжі

Розділ 1. Аналіз предметної області та постановка завдання

1.1. Аналіз сучасного стану питання та обґрунтування теми

В сьогоденних умовах кожен зіштовхується з необхідністю приготувати щось поїсти на вечерю, навіть якщо раніше цього ніколи не робив. У 2020 році категорія «рецепти» увійшла у список найпопулярніших запитів в Google. [2] Часто люди не знають, що можна приготувати з продуктів, які у них є в холодильнику.

Також може виникнути необхідність готувати страву з нових продуктів, і тоді виникає проблема пошуку рецепту навіть за назвою інгредієнту. В цьому випадку буде зручно, якщо б програма сама розпізнала інгредієнт та знайшла рецепти, складовою яких він є.

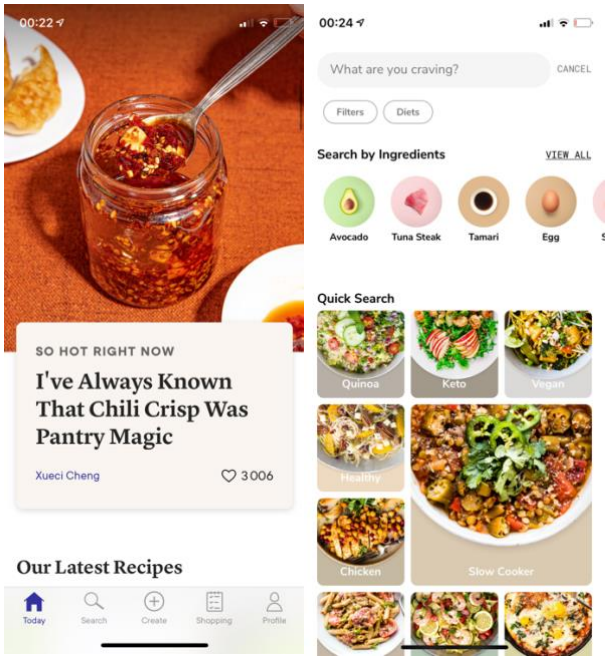
Є різні реалізовані додатки, які вирішують окремо кожне з цих питань, проте для цього користувачу прийдеться користуватися різними сервісами. Тому як вирішення, було обрано написати додаток, в якому можна було б знайти смачний рецепт просто завантажуючи фото наявних інгредієнтів.

1.2. Аналіз існуючих додатків для приготування їжі

Перш ніж приступати до розробки самого додатку, був проведений аналіз конкурентів, щоб проаналізувати переваги та недоліки їх застосунків.

Лідерами серед додатків для приготування їжі є SideChef, Tasty та Kitchen Stories. Всі три з них мають задумку кухарського блога, чимось схоже соціальні мережі для кулінарів.

Загалом як візуально, так і функціонально всі три застосунки між собою схожі. Окремо можна визначити зручний пошук за інгредієнтами в додатку SideChef, кожен з них відображений разом з фотографією, щоправда в цей ж самий час це є і незручністю, адже список інгредієнтів обмежений, а ввести інші, не з цього списку, немає можливості.



Русинок 1.1. Додатки-аналоги SafeChef та Kitchen Stories

Ще два додатки, які заслуговують за увагу – це “Go Fruit” та “See Food”, адже в них використовуються технології комп’ютерного бачення.

Суть першого застосунку полягає в класифікації лише екзотичних фруктів, також є каталог цих фруктів в окремому вікні. Другий додаток є щоденником харчування, його використовують коли хочуть виявити поживну цінність страви, яка зараз перед користувачем. В додатку немає можливості завантажити фотографію з галереї, що є суттєвим мінусом.

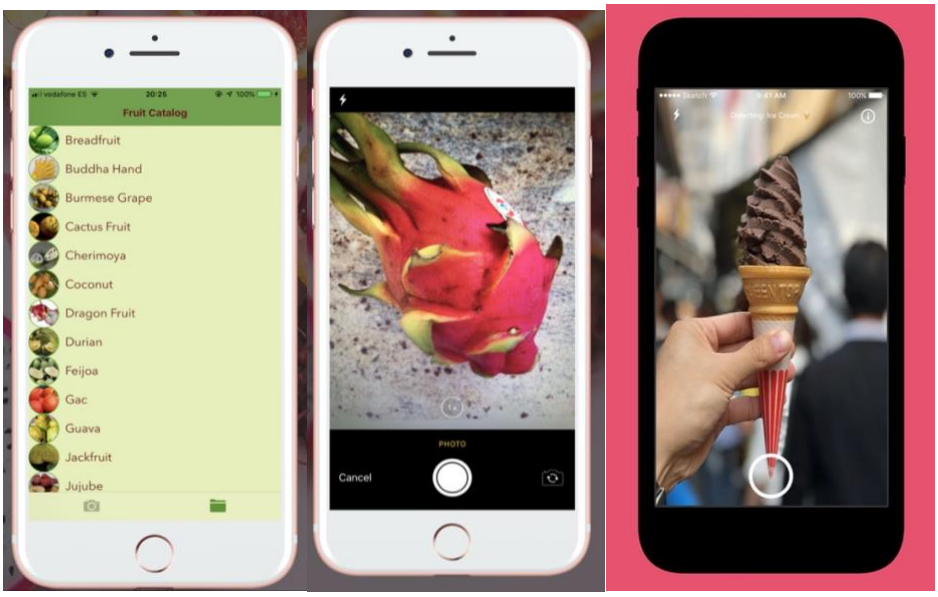


Рисунок 1.2. Додатки-аналоги з використанням комп’ютерного бачення

Проаналізувавши додатки конкурентів, було виявлено недоліки, які будуть враховані при розробці програмного продукту, а саме: не обмежувати набір інгредієнтів для пошуку рецептів, а у разі надання швидкого вибору інгредієнтів, у користувача має бути можливість ввести в рядку пошуку інгредієнт, якого немає в запропонованому списку; користувач повинен мати можливість, як зробити нову фотографію, так і завантажити з галереї зображення для класифікації.

1.3. Основні поняття технологій машинного навчання

Артур Самуель визначив машинне навчання як область навчання, яка дає комп'ютерам можливість вчитися без явного програмування ще в 1959 році. Величезна кількість даних - запорука гарної роботи машинного навчання. Саме тому сьогодні воно на піку своєї популярності, адже завдяки Інтернету і кількості його користувачів, даних зараз більше, ніж будь-коли раніше.

Сьогодні машинне навчання використовують для вирішення реальних проблем, які раніше вважалися надто важкими, а мобільні пристрої тепер досить потужні, щоб запускати алгоритми машинного навчання прямо в них.

Написання коду має на увазі під собою створення послідовного алгоритму - списку інструкцій, які потім програма має виконувати. Це є чудовим варіантом вирішення для більшості задач, однак є такі задачі, де такий евристичний підхід не задовольняє результатом. Це саме той тип задач, для яких застосовується машинне навчання.

Машинне навчання передбачає самостійне виявлення комп’ютером того, як він може виконувати завдання. Алгоритми машинного навчання будують модель на основі вхідних даних, відомих як "тренувальний набір", для того, щоб робити прогнози або приймати рішення, не будучи явно запрограмованими для цього.

Одне з основних понять машинного навчання - це модель.

Модель - це алгоритм, який вивчив комп’ютер для виконання певного завдання разом з даними, необхідними для запуску цього алгоритму.

Створити модель означає навчити її: показати величезну кількість прикладів проблеми, яку треба вирішити. Успішно навчена модель містить в собі знання про проблему. Ключова мета машинного навчання - навчання моделей, щоб вони могли робити хороші прогнози щодо нових даних.

Є декілька основних типів навчання моделі: навчання з учителем, або кероване навчання (з англ. supervised learning), навчання без учителя (з англ. unsupervised learning) та навчання з підкріпленням (англ. reinforcement learning).

Найпоширенішим є *навчання з учителем*. Це такий тип навчання, коли навчальний процес моделі керується людиною, вона вказує комп’ютеру що саме і як йому треба вивчити. Тобто людина надає моделі тренувальний набір для навчання, заздалегідь відмічає всі потрібні дані, щоб машина навчалася на конкретних прикладах.

Під час контрольованого навчання створюються моделі класифікації та регресії.

Зазвичай, щоб показати або передбачити взаємозв’язок між процесом та що може бути ним спричинено, використовуються моделі регресії. Результатом регресійної моделі є одне або кілька чисел з плаваючою крапкою. Щоб виявити існування та розташування обличчя на фотографії, ви використовуєте модель регресії, яка виводить чотири цифри, що описують прямокутник на зображенні, що містить обличчя.

Методи класифікації дозволяють розділити дані на завчасно зазначені класи. Наприклад, чи є електронне повідомлення спамом чи ні, чи це фото кота чи собаки. Саме тип моделі класифікації буде використовуватися в практичній частині курсовій роботи.

Під час навчання без учителя, людина ніяк не залучена до процесу навчання. Кластеризація - типовий приклад такого типу навчання. Моделі подається велика кількість ніяк не маркованих даних і її задача полягає у знаходженні закономірностей в цих даних. Програми пошуку подібних зображень використовують саме цей спосіб навчання.

Останній вид навчання - *навчання з підкріпленням*. Воно схоже на реальне навчання людей: машину карають за помилки і винагороджують за правильні вчинки. Цей тип навчання використовується для таких завдань, як програмування роботів.

1.4. Основні поняття технологій комп'ютерного бачення

Комп'ютерне бачення є підрозділом штучного інтелекту та машинного навчання, його мета: відтворити потужні можливості людського зору.

Комп'ютерним баченням називається область інформатики, яка зосереджена на наданні комп'ютерам можливості обробляти, аналізувати та розуміти візуальні дані (зображення та відео) за прикладом того, як це роблять люди. Тобто отримавши лише зображення, комп'ютерне бачення повинно надати нам якомога повніший опис того, що на ньому є: кольори, форми, розміри, розташування та інші характеристики всіх наявних на фотографії об'єктів.

Є декілька основних типових завдань, які виконуються комп'ютерним баченням:

Класифікація об'єктів - аналізується вміст зображення і класифікується як такий, що належить, до однієї з заздалегідь визначених категорій. Наприклад, чи знаходиться на зображенні квітка, чи ні.

Ідентифікація об'єкта - аналізується візуальний вміст зображення та виявляється конкретний об'єкт на фото або відео. Відслідковується, чи

з’являється об’єкт на зображенні чи ні. Прикладом може бути моніторинг зображень з камер безпеки для ідентифікації обличчя конкретної людини.

Відстеження об’єктів - система обробляє відео, знаходить об'єкт (або об'єкти), що відповідає критеріям пошуку, і відстежує його рух. Наприклад відстежується руху транспорту.

Розпізнавання об’єктів - спочатку може використовуватись класифікація, щоб віднести зображення чи відео до певного класу, а потім на ньому виявляється об’єкт.

Сегментація об'єкту - може розглядатись як наступний крок після розпізнавання об'єкта. Але тепер відбувається не лише пошук об’єктів на зображенні, а й для них створюється якомога точніші маски.

1.5. Висновки до розділу 1

В першому розділі були знайдені та проаналізовані існуючі додатки-аналоги для приготування їжі з та без використання технологій комп’ютерного бачення. Розглянуті основні поняття технологій машинного навчання та комп’ютерного бачення. Описані принципи їх роботи та приклади використання.

Розділ 2. Теоретичні відомості

2.1. Інформація про Swift як мову програмування

Спочатку основною мовою розробки програмного забезпечення iOS та Mac OS додатків була Objective-C. Це об’єктно-орієнтована мова програмування на основі таких мов як C та Smalltalk. І хоча вона до сих пір використовується, спеціально на заміну застарілої Objective-C, Apple Inc розробила і в 2014 році представила на щорічній конференції WWDC (Worldwide Developers Conference) нову мову програмування – Swift[3].

Swift – розроблена з нуля нова сучасна об'єктно-орієнтована мова, завдяки якій програмування стає простішим та швидшим. Основними перевагами є легкий та зрозумілий синтаксис, висока продуктивність та ефективна обробка помилок. Згідно з даними Apple, звичайний алгоритм пошуку виконується в Swift до 2,6 разів швидше, ніж на Objective-C та до 8,4 разів швидше, ніж на Python 2.7.[4]

Вагомою причиною розробки мови з самого початку було зробити її максимально безпечною. Так, в Swift змінні мають бути завжди проініціалізованими до їх використання, відбувається перевірка масивів та цілих чисел на переповнення, а пам’ять керується автоматично.

Для керування пам’яттю використовується механізм ARC (Automatic Reference Counting – з англ. «Автоматичний підрахунок посилань»). Він працює так: для кожного новоствореного екземпляру класу виділяється шматок пам’яті, де зберігається вся інформація про нього. ARC відслідковує, скільки в даний момент часу існує посилань на кожен уже збережений екземпляр класу. Якщо видаляються усі посилання на екземпляр, ARC вивільняє його з пам’яті.

Також об’єкти не можуть мати порожнього значення (nil) – компілятор одразу видаватиме помилку. Однак, в деяких ситуаціях є необхідність використати nil. Для таких випадків, в Swift представлений додатковий тип

– Опціонали (Optional), які обробляють відсутність значення. Сам по собі опціонал – це просто перелік із двома випадками: .none та .some. У випадку Optional.none, він не має жодного пов'язаного значення, і дорівнюватиме nil. В іншому випадку – Optional.some, міститиме обгорнуте пов'язане з ним значення.[5]

Крім того, Swift є статично та строго типізованою мовою, ви не можете передати аргумент з невідповідним типом даних, а його правильність перевіряється на етапі компіляції. Вказати тип або надати початкове значення змінній потрібно одразу під час її оголошення.

В 2019 році Apple представила SwiftUI. Фреймворк, який допомагає дуже швидко і просто побудувати користувацький інтерфейс. Він підтримується на будь-якому пристрої виробництва Apple.

SwiftUI використовує декларативний синтаксис, завдяки якому код виглядає набагато простішим і легшим для сприйняття, що точно зекономить час на його написання. В SwiftUI ви створюєте ієрархію відображень і вказуєте залежності даних між ними. Коли змінюється дані, SwiftUI автоматично оновлює потрібні частини інтерфейсу. Як результат, фреймворк автоматично виконує більшу частину роботи, яку традиційно виконують контролери.

2.2. Використання фреймворку Core ML в iOS застосунках

У вересні 2017 інтегрувати машинне навчання у свої додатки стало ще легше – на WWDC'17 Apple представила новий фреймворк для роботи з технологіями машинного навчання Core ML[6].

Core ML – фреймворк, завдяки якому можна підключати попередньо підготовлені моделі машинного навчання до своїх додатків всього в пару рядків коду, він доступний на будь-якому пристрої виробництва Apple. Core ML дозволяє виконувати обрахування в реальному часі на самому пристрої, при цьому забезпечує дуже високу продуктивність. Починаючи з Core ML

3, моделі машинного навчання можуть бути персоналізовані під окремого користувача ним самим, а дані при цьому залишаються приватними.

Рисунок 2.1. Використання моделі за допомогою Core ML

У Core ML всі запити відбуваються локально на девайсі користувача. Звідси немає будь-якої необхідності відсилати дані на сервер для обробки, що підвищує безпеку даних та пришвидшує роботу програми. Останнє особливо важливе, коли треба робити розрахунки машинного навчання в реальному часі, наприклад коли використовується машинне навчання для прямої трансляції відео. Тоді процес відправлення HTTP-запитів буде занадто повільним для такої задачі. Взагалі, оскільки немає потреби мати підключення до інтернету, програма може працювати офлайн – що є вигідним як для споживача, так і розробника, адже не потрібно витрачати кошти на мобільний трафік та платити за сервер.

Зараз існує декілька фреймворків: Natural Language, Speech, Vision і Sound, які базуються на Core ML. Вони забезпечують розширені функції машинного навчання за простими API.

Фреймворк *Natural Language* створений для автоматичного визначення мови; токенізації – розбиття тексту на лінгвістичні одинці; виявлення частин мови; лематизації – виведення основи слова шляхом його морфологічного аналізу; розпізнавання назв сутностей, таких як імена людей, місць, організацій. За допомогою Create ML можна створювати власні моделі для обробки тексту.[7]

Speech фреймворк використовується для розпізнавання слів у записі, або живому аудіо, розпізнавання словесних команд. Розпізнавання доступне багатьма мовами, але Speech Recognizer розпізнає тільки одну

мову за раз. Також, для розпізнавання голосу завжди потрібне підключення до мережні.[8]

Для аналізу живого або записаного аудіо і виявлення в ньому звуків, наприклад сміху чи оплесків, використовуйте фреймворк *SoundAnalysis*. Для його застосування вам знадобиться модель машинного навчання для звукової класифікації, яку ви також можете самостійно створити в Create ML.[9]

Фреймворк *Vision* – про нього детальніше в наступному розділі.

Для інтегрування моделей в свій додаток, ви можете створити свою модель в Create ML, або використати уже готову. На сайті Apple ви можете знайти уже пре-треновані моделі в форматі Core ML, готові для використання[10].

Якщо у вас є модель машинного навчання, яка виконує потрібні вам функції, але не в форматі Core ML, можна конвертувати її за допомогою coremltools.

Coremltools - Python-утиліта для перетворення моделей із сторонніх навчальних бібліотек, таких як Apache MXNet, Caffe, Keras. PyTorch, scikit-learn, TensorFlow, у формат Core ML. Після конвертації ви можете використовувати Core ML для інтеграції моделей у свій додаток.

Важливо зазначити про розмір моделей машинного навчання, які інтегруються в додатки. Чим більша модель, тим більше енергії акумулятора вона споживає, і тим повільніше працює додаток. Розмір файлу Core ML пропорційний розміру дата-сету.

2.3. Використання фреймворку Vision в iOS застосунках

Сьогодні в кожному мобільному телефоні є вбудована камера, люди «наповнили» світ фотографіями і відео. Тому очевидно, що галузь комп'ютерного бачення дуже швидко розвивається. Так, на WWDC`17 поряд з Core ML, Apple представила фреймворк Vision.

Vision – фреймворк Apple для розробки додатків з використанням комп’ютерного бачення. Він використовується для таких задач як виявлення обличчя та його рис, орієнтирів, розпізнавання тексту, штрих-кодів, зіставлення зображень та загальне відстеження особливостей. До того ж можна класифікувати зображення та відстежувати об’єкти на зображеннях за допомогою власних моделей Core ML [11].

Стандартний робочий процес фреймворка Vision працює так: 1) створення моделі Core ML 2) створення одного або декількох запитів (Request), 3) створення і запуск обробника запитів (Request Handler), 4) повернення результатів обробки запиту(Observation)

Request – у ньому вказується, що треба виявити. Бувають різні типи запитів:

- VNDetectFaceRectanglesRequest - для виявлення облич на зображенні.
- VNDetectBarcodesRequest - для виявлення штрих-коду.
- VNDetectTextRectanglesRequest - для виявлення видимої області тексту на зображенні.
- VNCoreMLRequest - для аналізу зображень, які використовують Core ML моделі для обробки зображень.
- VNClassifyImageRequest - запит на класифікацію зображення.
- VNDetectFaceLandmarksRequest - запит на пошук рис зображення на зображенні, таких як очі та рот.
- VNTrackObjectRequest - запит, що відстежує переміщення об'єкта на декількох зображеннях або відео.

Та інші.

RequestHandler – викликається після завершення запиту. Він може виконати більше одного запиту на дане зображення. Є два типи обробників:

- VNImageRequestHandler - для аналізу зображення.
- VNSequenceRequestHandler - для аналізу серії зображень.

Observation – повертаються результати обробки запитів. Результати можуть бути:

- VNDetectedObjectObservation – надає положення та масштаб виявленого елементу зображення
- VNHumanHandPoseObservation – повертає точки знаходження людських рук на зображенні
- VNContoursObservation – повертає всі виявлені контури на зображенні.

Та інші.

2.4. Використання інструменту Create ML

Через два роки після виходу Core ML, в 2019 році Apple представив новий окремий інструмент завдяки якому створення власних моделей машинного стало ще легшим - Create ML[12].

Раніше для створення моделей використовувався Playground. Наприклад, щоб створити модель для класифікації зображень, в Playground треба було імпортувати бібліотеку CreateML і запустити MLImageClassifierBuilder. Тепер ж для цього не потрібно писати жодного рядка коду.

Також окрім моделей для роботи з зображеннями, текстом і таблицям, в Create ML є можливість створити і інші: для роботи з відео, рухом та звуками.

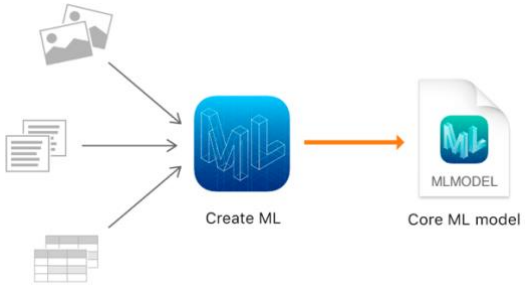


Рисунок 2.3. Навчання моделі за допомогою Create ML

Щоб натренувати модель, вам потрібно зібрати набір даних – дата-сет. Наприклад, щоб створити модель машинного навчання, яка б розпізнавала знаки дорожнього руху, потрібно було б зібрати велику кількість різних фотографій різних знаків, розбити на окремі класи і після цього додати їх на тренування моделі. На кожній фотографії класу має бути присутній лише один знак цього класу, щоб модель не заплутати модель. Класи повинні бути збалансованими, тобто в кожному має бути приблизно однакова кількість елементів.

Важливо серйозно поставитись до цієї задачі, адже саме від даних, якими ви «нагодуєте» модель, залежить наскільки добре вона виконуватиме свою задачу і чи буде вона робити те, що ви очікуєте. У разі неправильно підібраних тренувальних, модель може розпізнавати зовсім інше. Наприклад, якщо фотографії одного знаку будуть зроблені тільки в сонячний день, а другого в похмурий, модель може розпізнавати не знаки, а погоду.

Під час тренування моделі в Create ML ми бачимо графік, де на кожній ітерації показується training accuracy – точність моделі на прикладах, на яких вона вчиться, validation accuracy – це точність моделі на прикладах, яких вона не бачила. Точність – кількість правильних класифікацій.

Після того, яка ви закінчили навчати модель, перевірте її на даних, яких вона ще не бачила. На цьому етапі треба звернути увагу на відсоток влучності(precision) та повноти(recall) для кожного класу.

Влучність показує, який відсоток з зображень, з тих, які модель прогнозує як об’єкт класу "X", насправді є класом "X". Якщо модель робить мало помилок, то відсоток влучності високий, і навпаки, якщо багато помилок – низький. Помилкою вважається результат якщо об’єкт на зображенні насправді не є з класу, яким його класифікувала модель.

Повнота показує, який відсоток зображень класу "X" знайшла модель серед загальної кількості зображень класу "X". Так ми можемо зрозуміти,

скільки у нас хибно негативних класифікацій. Хибно негативнивним вважається результат, якщо об’єкт на зображенні насправді належить до одного класу, але модель відносить його до іншого.

Як тільки робота моделі вас задовільнить, ви можете додати її у свій додаток. Для цього вам треба її зберегти на свій комп’ютер, а потім перетягнути у потрібний проект в Xcode.

2.5. Висновки до розділу 2

В розділі 2 були розглянута теоретична складова використаних у практичній частині технологій та описані їх переваги використання.

Розглянута мова програмування Swift, фреймворк для написання графічного інтерфейсу SwiftUI, фреймворк для інтеграції машинного навчання в додаток Core ML, фреймворк для роботи з комп’ютерним баченням Vision та інструмент для створення моделей машинного навчання Create ML.

Розділ 3. Опис реалізації продукту

3.1. Аналіз технічного завдання

Технічним завданням є розробка iOS додатку з технологією комп'ютерного бачення, щоб полегшити користувачам процес приготування їжі.

Необхідно створити або використати існуючу модель машинного навчання для класифікації зображень. Розробити додаток, завдяки якому користувач міг знайти рецепт, за його складовими. У споживача також має бути можливість сфотографувати, або завантажити фото інгредієнту, щоб програма сама додала б його в список, без необхідності користування клавіатурою.

Також потрібно створити інтуїтивно-зрозумілий користувацький інтерфейс, для полегшення взаємодії користувача з додатком.

3.2. Огляд засобів розробки

Як середу розробки було обрано Xcode – IDE розроблену Apple. Xcode має весь необхідний набір інструментів, для проектування, написання, тестування та створення інтерфейсу додатків для платформ Apple. [13]

Щоб протестувати написаний код під iOS, можна просто підключити девайс через кабель до свого комп'ютера. А якщо ж немає телефону під рукою, можна використати симулятор. Симулятори емулюють роботу пристроїв iPhone, iPad, Apple Watch та Apple TV. Кожен можна по-різному налаштувати та вибрати версію операційної системи.

В меню «Інструменти розробника» в Xcode, можна запустити інші засоби розробки, один з них є Create ML.

Саме Create ML був обраний для створення моделі машинного навчання, він є власною розробкою Apple . З його допомогою можна дуже швидко і просто створювати моделі машинного навчання. В ньому одразу надається великий вибір шаблонів для створення моделей, одним з яких є

класифікація зображень. Create ML має зрозумілий інтерфейс, завдяки якому можна легко відстежувати етап тренування, а також є можливість одразу протестувати новостворену модель. Всі моделі в Create ML створюються в форматі Core ML.

Core ML – фреймворк, з допомогою якого можна інтегрувати моделі створені в Create ML в будь-який додаток.

Vision – фреймворк на основі Core ML для роботи з комп'ютерним баченням. Він безперечно є дуже швидким і надійним, адже, як раніше було сказано, базується на Core ML. Останній ж оброблює всі дані на самому пристрої. Тому не витрачається додатковий час на надсилання запитів на сервер, а дані ніколи не покидають гаджету.

Spoonacular API [14] надає доступ до понад 365 000 рецептів та 86 000 харчових продуктів. Це один з найбільших на потужніших API пов'язаних з їжею на сьогоднішній день.

API (application programming interface – з англ. «прикладний програмний інтерфейс») – це набір готових класів, функцій, процедур, структур і констант. Вся ця інформація надається самим додатком (або операційною системою).

Мета наданої інформації - використання цих даних при взаємодії з зовнішніми програмами. API різних програмних продуктів використовуються програмістами для створення додатків, що будуть взаємодіяти один з одним. У загальному випадку API застосовується з метою об'єднання роботи різних додатків в єдину систему.

Alamofire[15] - це бібліотека з відкритим кодом на базі Swift для зручної роботи з HTTP запитами. Завдяки цій бібліотеці робота з мережею відбувається набагато легше, а код виглядає елегантніше. Вона не входить до стандартного пакету бібліотек Apple, тому її необхідно додатково підключити. Для підключення Alamofire використовувася Swift Package Manager.

Swift Package Manager[16] – це нативний менеджер залежностей Apple, він вбудований в Xcode.

3.3. Опис та обґрунтування архітектури проекту

Для створення додатку був обраний шаблон проектування MVVM – Model - View – View Model. Шаблони використовуються для розподілення обов'язків між сутностями, для кращої організації коду і читабельності.
В MVVM код розподілений на ролі:

Model – в ній представлені моделі даних, вона повідомляє про реальний стан даних.

View Model – зв'язує дані Model та їх відображення у View та відслідковує поточний стан даних Model. View Model не має доступу до View.

View – відповідає за макет, структуру, зовнішні вигляд інтерфейсу користувача та обробку взаємодії користувача з додатком, повідомляє View Model про зміни.

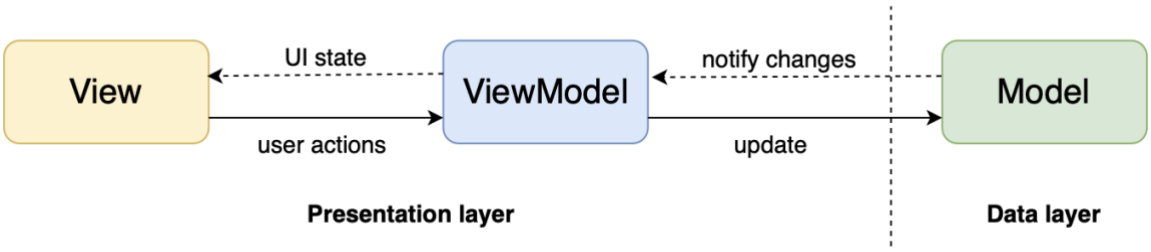


Рисунок 3.1. Схема шаблону проектування MVVM

Мета MVVM полягає в відділені моделі та відображення один від одного. При цьому зв'язок між рівнями будується «зв'язуванням даних» (binding).

Binding – забезпечує зв'язок між властивістю, що зберігає дані, та відображенням (View), яке їх відображає. [17]

Взагалі, розробка iOS додатків була завжди зосереджена на Model-View-Controller (MVC), але на практиці в models та view controllers було забагато коду. Тому в MVVM ж уже з'являється новий клас – ViewModel і тепер «ViewController» відповідає лише за відображення.

З виходом SwiftUI, стало дуже зручним використовувати саме MVVM архітектуру, адже концепція «зв'язуванням даних» лежить в її основі.

SwiftUI використовує обгортки(wrapper) властивостей, такі як @State, @Binding та @ObservedObject, @Published. View залежить від стану властивостей і щоразу, як оновлюються дані, змінні у View також змінюються, а інтерфейс перебудовуються автоматично.

@State – властивість, яка використовується в межах структури відображення. Вона завжди має бути private.

@Binding – пов'язує дану властивість у структурі із джерелом істини, яке знаходиться в іншому місці. Зміна значення локально, змінює її значення в джерелі істини.

@ObservedObject – відноситься до екземпляру зовнішнього класу, який відповідає протоколу ObservableObject.

@Published – приєднується до властивостей усередині ObservableObject і повідомляє SwiftUI, що він повинен оновити змінені властивості.

3.3. Опис розробки застосунку

Для даного застосунку, було обрано створити свою невеличку модель машинного навчання. Для цього підготовлено дата-сет, в якому налічується 9 класів, в кожному з яких по 70-100 фотографій, загалом в яких містяться 744 фотографії. За замовчуванням, пропонується зробити 30 ітерацій для тренування моделі. На кожній ітерації модель повністю проходить весь дата-сет.

Оскільки набір даних досить маленький, тренування закінчилось на 10 ітерації. 30 ітерацій було б забагато, і в такому випадку, могла б статися така річ, як перенавчання моделі. Це коли модель замість того, щоб «вчитися» починає «заучувати» тренувальні данні. Тоді навіть невеликі відмінності будуть сприйматися нею, як похибка.

Рисунок 3.2. процес тренування моделі машинного навчання в Create ML

На тестових даних, загалом, показники досить високі. Найнижчим показником є 75% - повнота класу «хліб». Отже, модель класифікувала один або декілька об’єктів інших класів хлібом. Це говорить про те, що або даних для тренування було недостатньо, або були неправильно підібрані тестові дані.

Class	Item Count	Precision	Recall
bread	6	75%	100%
carrot	5	83%	100%
cucumber	7	100%	100%
eggs	6	100%	100%
garlic	8	100%	100%
onion	7	100%	86%
pepper	8	100%	88%
potato	6	100%	83%
tomato	6	83%	83%

Рисунок 3.3. влучність та повнота створеної моделі на тестових даних

Проаналізувавши тестові данні, було виявлено неоднозначні елементи, на яких були елементи більше одного класів, це і було причиною нижчих показників. Однак, для реалізації застосунку показники задовільні, тому створення та тестування моделі припинено на цьому етапі.

Розробка додатку було розпочата з користувацького інтерфейсу. Графічний інтерфейс є дуже важливою невід'ємною частиною кожного додатку. Це саме та складова, через яку користувач взаємодіє з системою. Тому, від гарно продуманих UI (user interface - з англ. «користувацький інтерфейс») та UX(user experience - з англ. «досвід користувача») програми залежить, чи будуть користувачі далі користуватися додатком.

UX – відповідає за структуру, зрозумілість навігації додатком.

UI – за візуальну складову: кольорову гаму, підбір шрифтів, вигляд елементів на екрані.

Коли додаток завантажується, користувач бачить вікно запуску, після нього відкривається головний екран.

Головний екран містить список інгредієнтів та можливості роботи з ним: рядок введення тексту, кнопка камери, кнопка, щоб очистити список та кнопка пошуку. За відображення його відповідає SeachView, окремими структурами виділені опис-привітання - InfoView та список інгредієнтів - IngredientsListView.

При натисканні кнопки «камера» з'являється actionSheet, щоб користувач міг зробити вибір, як саме завантажити фотографію: сфотографувати, чи вибрати фото з галереї. Коли він зробить свій вибір, на екрані з'являється ImagePicker.

Хоча SwiftUI справді є дуже потужним фреймворком, але оскільки він досить новий, не всі функції можливо реалізувати, користуючись лише ним. Для кращої функціональності, використовується UIKit[18]. Щоб

користувач міг вибрати фотографію або сфотографувати новий інгредієнт, використовуємо UIImagePickerController та протоколи UINavigationControllerDelegate і UIImagePickerControllerDelegate. Протоколи визначають список необхідних методів, властивостей, які підходять для конкретного завдання. UIKit використовує шаблон делегування, який застосовується щоб розуміти, де саме відбувається зараз робота. UIKit використаний для ImagePicker.

Структура ImagePicker є одною з найважливіших, адже саме в ній викликається та оброблюється запит для розпізнавання зображення. Як було сказано раніше, для цієї задачі використовується фреймворк Vision. Щоб розпізнати зображення, нам треба створити запит та запит на обробку, після цього обробити результат.

Для створення змінної classificationRequest типу VNCoreMLRequest, спочатку створюємо екземпляр класу IngredientClassifier. Це автоматично згенерований Xcode клас при додаванні моделі в проект. Після цього створюємо об'єкт VNCoreMLModel, він з'єднує екземпляр моделі Core ML з Vision. Об'єкт VNCoreMLRequest перетворює зображення до CVPixelbuffer, масштабує його до потрібного розміну (зазвичай це 224 × 224, 227 × 227 або 229 × 229 – Vision автоматично зменшує зображення до правильного розміру, але якщо ви самостійно реалізуєте масштабування, треба подивитися необхідний формат в налаштуваннях моделі), запускає модель Core ML, інтерпретує результати.

```
lazy var classificationRequest: VNCoreMLRequest = {
    do {
        let config = MLModelConfiguration()
        let ingredients = try IngredientsClassifier(configuration: config)
        let visionModel = try VNCoreMLModel(for: ingredients.model)
        let request = VNCoreMLRequest(model: visionModel, completionHandler: { [weak self] request, error in
            self?.processObservations(for: request, error: error)
        })
        request.imageCropAndScaleOption = .centerCrop
        return request
    } catch {
        fatalError("Failed to create VNCoreMLModel: \(error)")
    }
}()
```

Лістинг 3.1. Створення об’єкту запиту VNCoreMLRequest

В методі classify(image:) спочатку конвертуємо зображення, яке ми отримали з UIImagePickerController з UIImage в CGImage або CIImage, бо саме з цими об’єктами працює Vision. Для процесу обробки та класифікації зображення потрібно більше часу, тому об’єкт VNImageRequestHandler створюємо та оброблюємо в окремому потоці.

```
func classify(image: UIImage) {
    guard let ciImage = CIImage(image: image) else {
        print("Unable to create CIImage")
        return
    }
    DispatchQueue.global(qos: .userInitiated).async {
        let handler = VNImageRequestHandler(ciImage: ciImage)
        do {
            try handler.perform([self.classificationRequest])
        } catch {
            print("Failed to perform classification: \(error)")
        }
    }
}
```

Лістинг 3.2. Метод classify(image:)

В класі DataModel відбувається вся робота з мережею, в ньому відбуваються запити до сервера через HTTP запити. Відповідь нам повертається в форматі JSON, отримані дані декодуються в моделі, щоб продовжити роботу з ними.

```
class DataModel {

    //MARK:  Properties
    private var task: Cancellable? = nil
    private let baseUrl = "https://api.spoonacular.com/recipes/"
    private let API_KEY = "d99016218c4242bba3927c475191b2f9"


    //MARK: - fetching
    func loadRecipeInfo(id: Int, completion: @escaping((Details)-> Void)){

        self.task = AF.request(baseUrl + "\(id)/information?",
                               parameters: ["apiKey": API_KEY])
            .publishDecodable(type: Details.self)
            .sink(receiveCompletion: {completion in
                switch completion{
                    case .finished:
                    ()
                case .failure(let error):
                    print(error.localizedDescription)
                }
            }, receiveValue: { (response) in
                switch response.result {
                case .success(let model) :
                    completion(model)
                case .failure(let error):
                    print(error.localizedDescription)
                }
            })
     }
```

Лістинг 3.3. Метод loadRecipeInfo(id:,completion:)

3.4. Принцип роботи готового застосунку

Коли додаток завантажується, користувач бачить вікно запуску, де зображений логотип на назва програми – SeenGredient. Після цього відкривається головний екран.

Натискаючи на кнопку «камера», користувачу надається вибір, сфотографувати, або завантажити фотографію з галереї для розпізнавання інгредієнту. При першому запуску додатку, користувач має надати додатку доступ до галереї свого телефону, якщо хоче користуватися технологією розпізнавання.

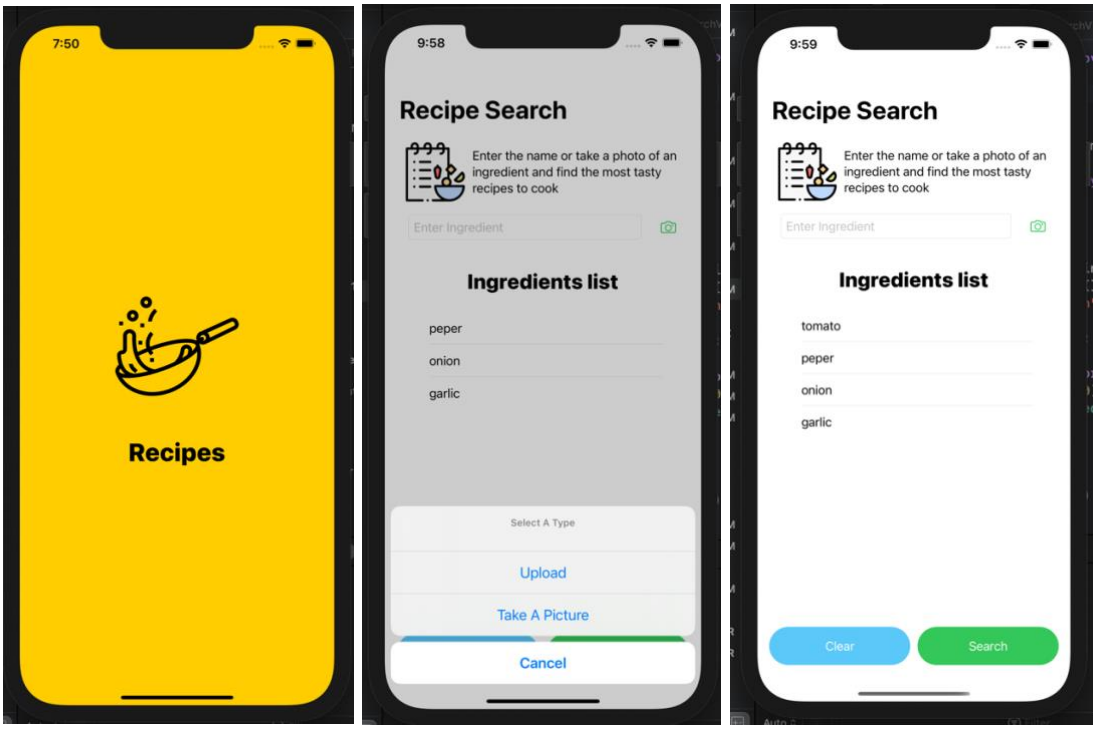


Рисунок 3.4. вікно запуску, головне вікно додатку

Коли список інгредієнтів готовий, користувач переходить до наступного вікна – списку, знайдених за інгредієнтами, рецептів та додаткових складових. Кожен рецепт можна детальніше подивитися в окремому вікні.

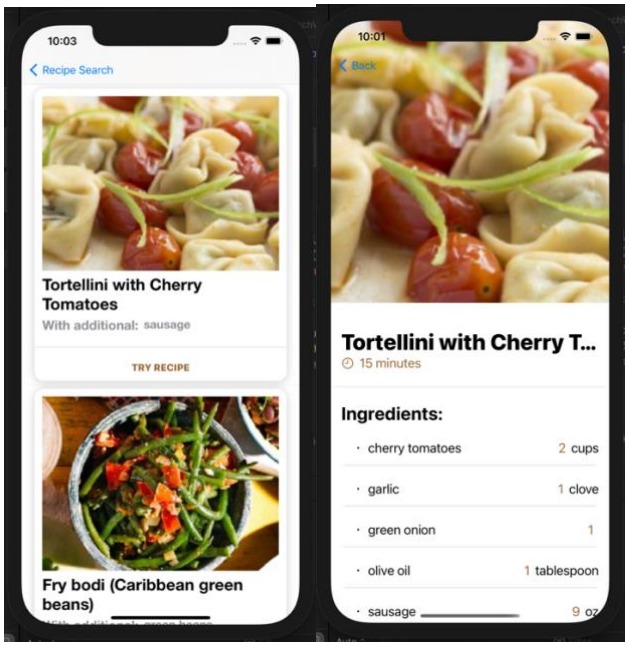


Рисунок 3.5. список рецептів та вікно деталей рецепту

3.5. Висновки до розділу 3

В розділі 3 було описане технічне завдання, використані засоби розробки застосунку та обґрунтована вибрана архітектура застосунку.

Були описані етапи розробки застосунку, такі як: створення навчання та тестування моделі машинного навчання. Під’єднання моделі до проекту та обробки зображень з використанням фреймворку Vision.

В кінці розділу наведені скріншоти з роботою готового продукту.

Висновки по роботі та аналіз можливостей для подальшого розвитку застосунку

Результатом виконаної роботи стала реалізація застосунку під операційту систему iOS з використанням технологій комп’ютерного бачення. При створенні цього програмного продукту, були успішно виконані всі функціональні вимоги, які ставились за мету.

Досліджені технології для створення моделей машинного навчання – Create ML. Також було досліджено фреймворки для роботи зі створеними моделями – Core ML, та для роботи з комп’ютерним баченням – Vision та новий фреймворк SwiftUI для створення графічного інтерфейсу. Провівши дослідження, були сформовані та проаналізовані недоліки та переваги кожного вибраного інструментарію.

Основною перевагою роботи з технологіями машинного навчання та комп’ютерного бачення на iOS, що обидві технології Core ML та Vision надають високоякісний та потужний функціонал за простими API. Великою перевагою фреймворків є їх безпечність та швидкість роботи. З недоліків можна виділити, що для роботи з моделями машинного навчання на самому девайсі, програма збільшується в розмірі, а батарея пристрою розряджається швидше через більше обчислювальне навантаження.

Фреймворк SwiftUI має зрозумілий декларативний синтаксис та надає можливість дуже зручно побудувати ієрархію відображень. Завдяки зв’язуванню даних оновлення інтерфейсу відбувається автоматично. Також великим плюсом є можливість використовувати попередній перегляд під час створення графічного інтерфейсу. SwiftUI є новим досить молодим фреймворком і це є його недоліком на сьогоднішній день. Не всі функції можливо реалізувати лише користуючись ним. А ще є ймовірність зіштовхнутися якоюсь проблемою першим, коли ще немає відповідей на це питання в мережі.

У додатку є багато можливостей подальшого розвитку, зокрема додання можливості класифікації готових страв, та можливості розпізнавання багатьох інгредієнтів одночасно, створення особистого кабінету користувача та можливості додати власний рецепт на платформу.

Список джерел

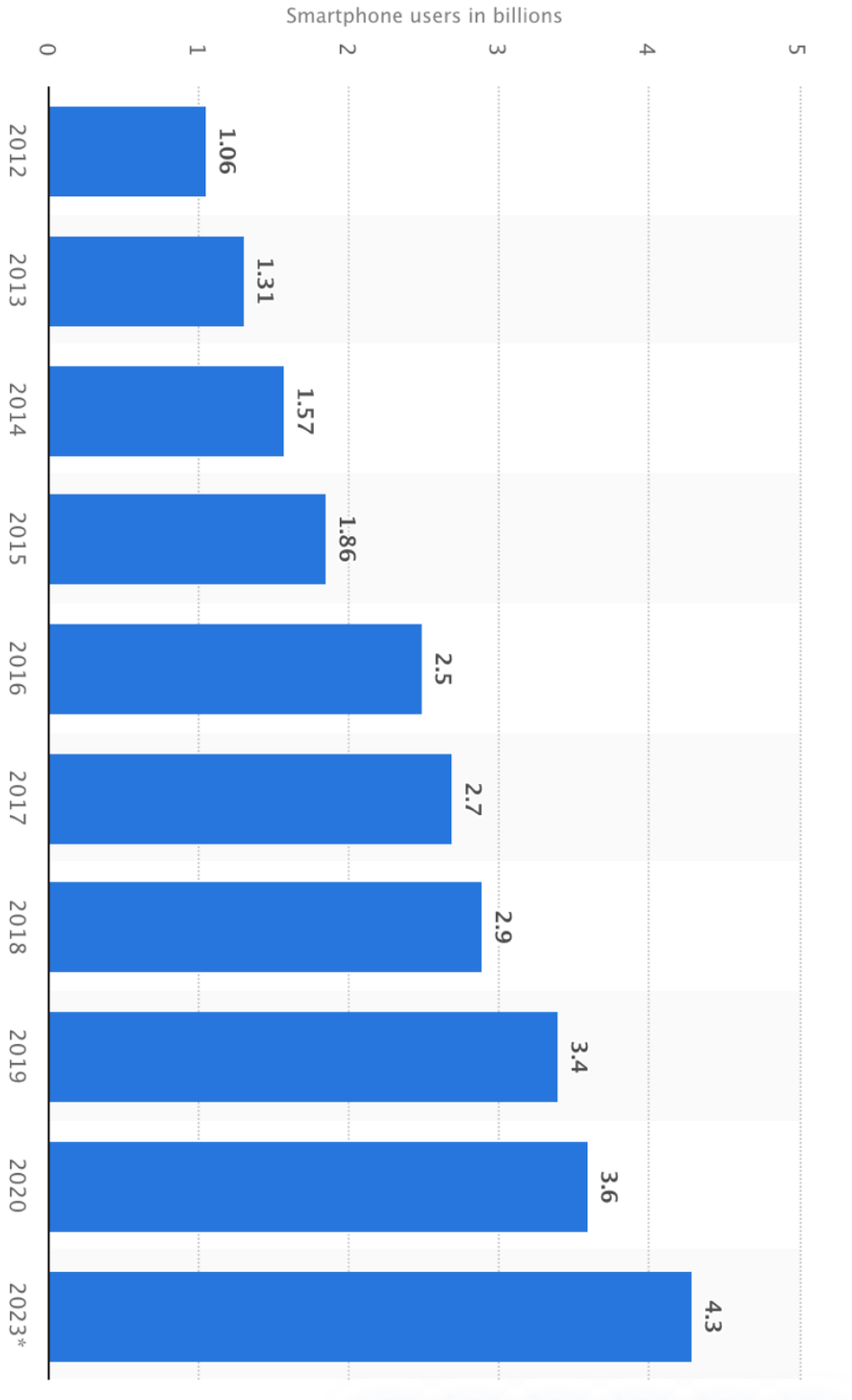
1. Smartphone users worldwide 2016-2023 [Електронний ресурс] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
2. Найпопулярніші теми у 2020 році [Електронний ресурс] https://trends.google.com/trends/yis/2020/GLOBAL/
3. Swift [Електронний ресурс] https://swift.org
4. Swift. A powerful open language that lets everyone build amazing apps. [Електронний ресурс] https://www.apple.com/swift/
5. iOS 13 Programming Fundamentals with Swift, 2019, p.279
6. Core ML [Електронний ресурс] https://developer.apple.com/documentation/coreml
7. Natural Language [Електронний ресурс] https://developer.apple.com/documentation/naturallanguage
8. Speech [Електронний ресурс] https://developer.apple.com/documentation/speech/
9. Sound Analysis [Електронний ресурс] https://developer.apple.com/documentation/soundanalysis
10. Core ML Models [Електронний ресурс] https://developer.apple.com/machine-learning/models/
11. Vision [Електронний ресурс] https://developer.apple.com/documentation/vision
12. Create ML https://developer.apple.com/documentation/createml
13. Xcode [Електронний ресурс] https://developer.apple.com/xcode/

14. Spoonacular API [Електронний ресурс]
https://spoonacular.com/food-api
15. Alamofire [Електронний ресурс]
https://github.com/Alamofire/Alamofire
16. Swift Package Manager [Електронний ресурс]
https://swift.org/package-manager/
17. Binding [Електронний ресурс]
https://developer.apple.com/documentation/swiftui/binding
18. UIKit [Електронний ресурс]
https://developer.apple.com/documentation/uikit/

Додатки

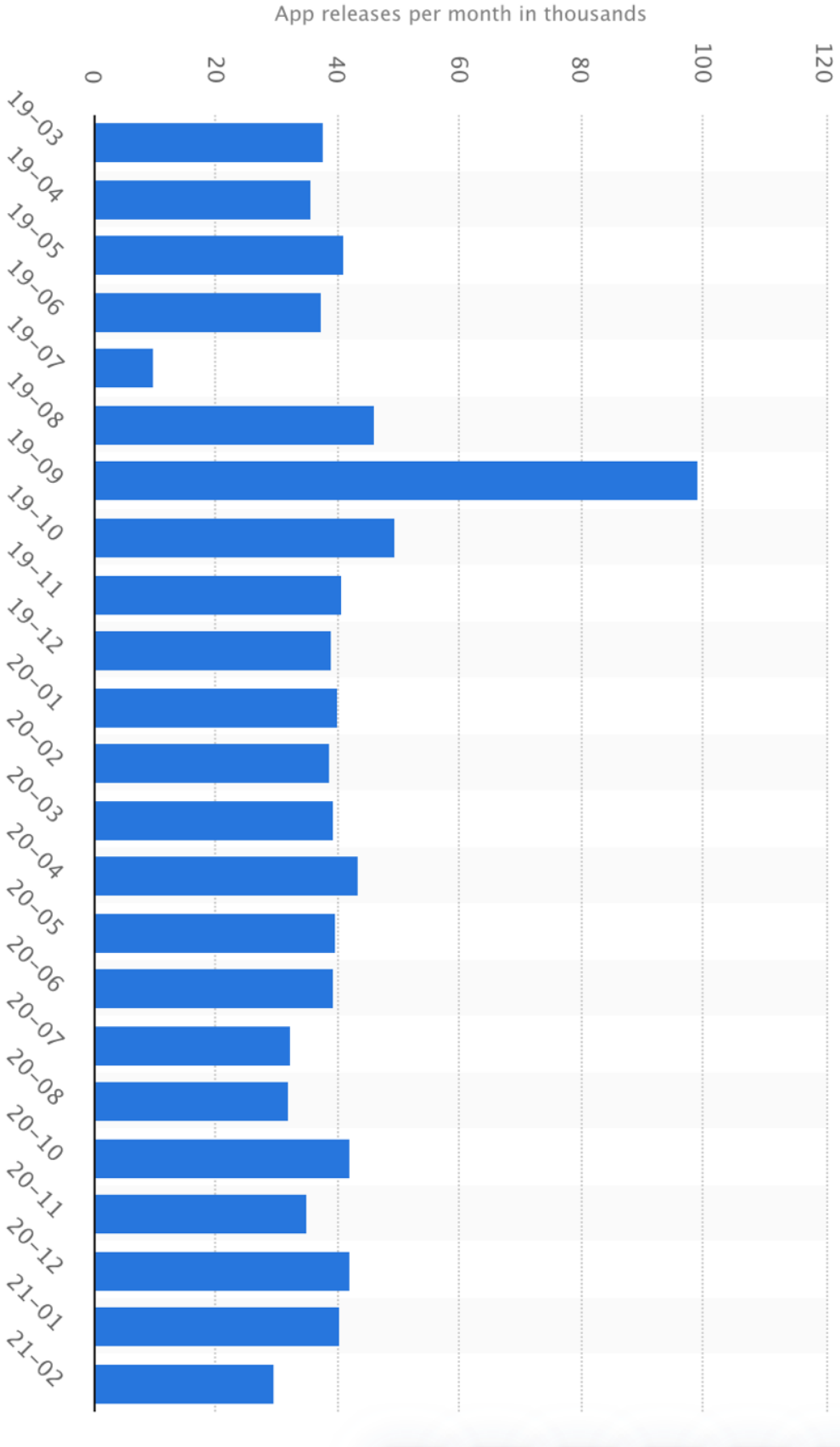
Додаток А

Кількість користувачів смартфонів в мільярдах



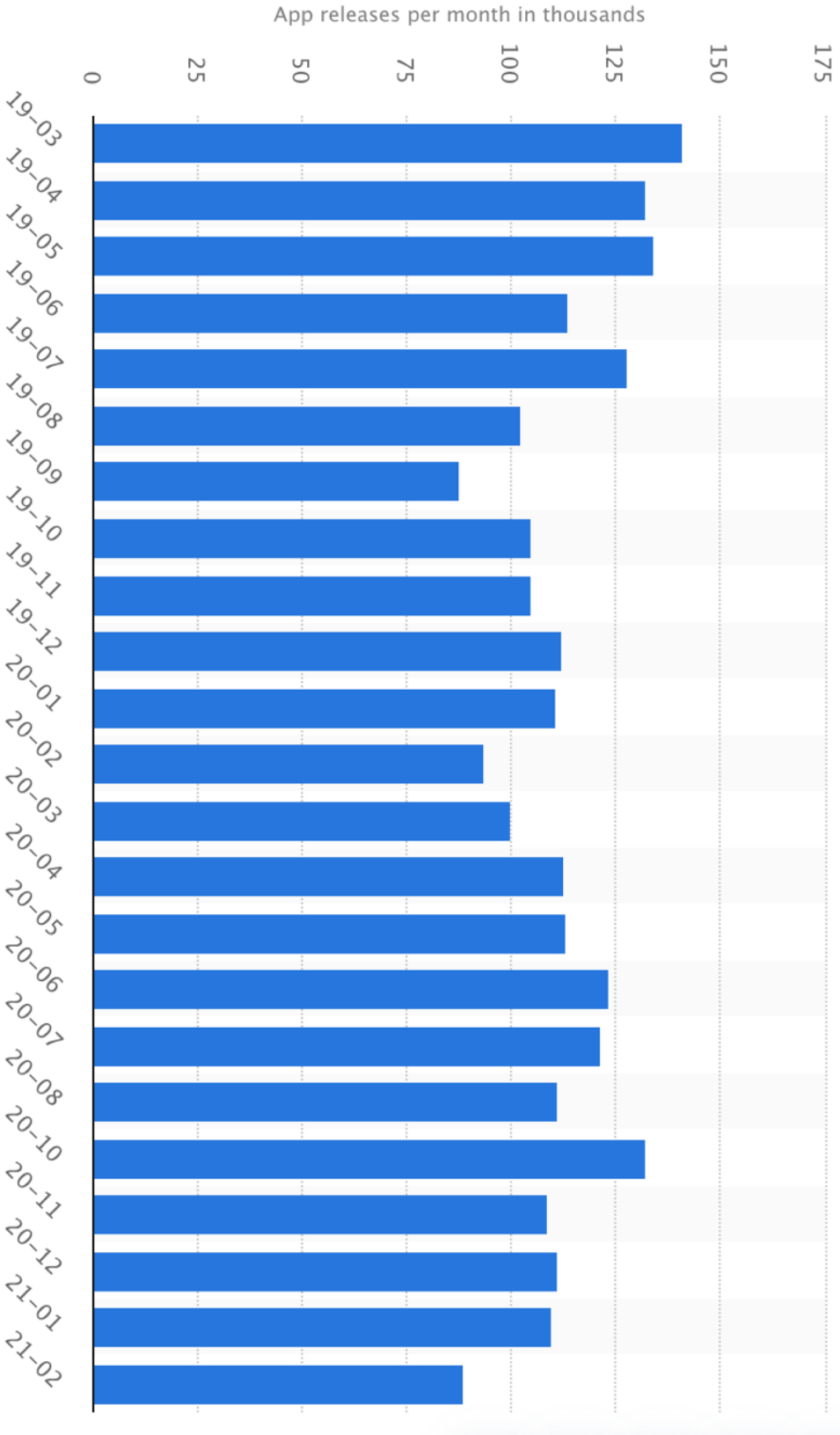
Додаток Б

Кількість релізів iOS додатків за місяць в тисячах



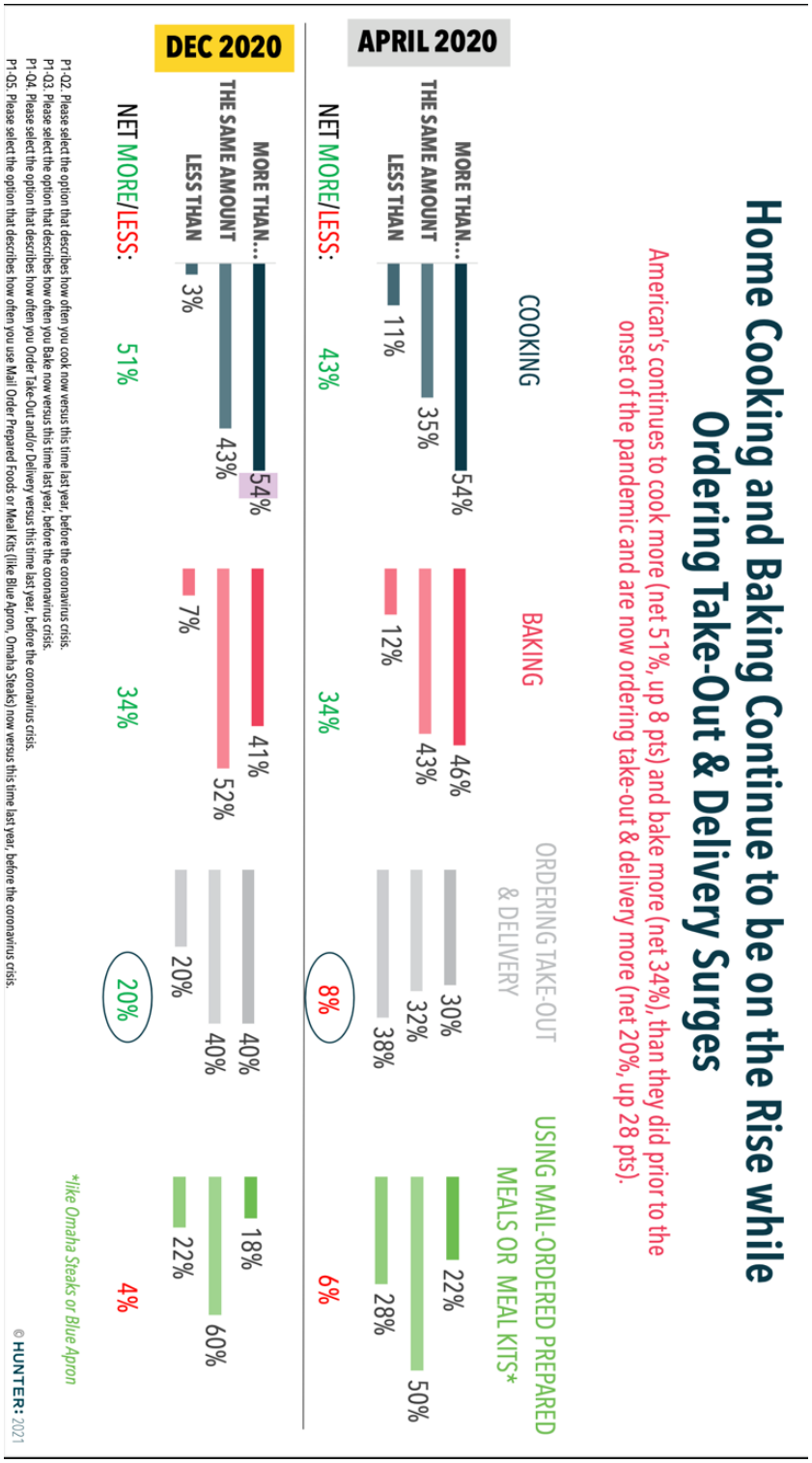
Додаток В

Кількість релізів Android додатків за місяць в тисячах



Додаток Г

Кількість людей, що готують вдома



Додаток Ґ

Скільки в 2020 щохвилини генерувалося даних