Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Алгоритм розпізнавання математичних виразів для iOS-пристроїв

(тема)

Виконав:
студент II курсу, групи СПм-19-1
Лук'яненко Є.С.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: ст. викл. Єрьоміна Н.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп’ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Лук'яненку Євгенію Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Алгоритм розпізнавання математичних виразів для iOS-пристроїв

затверджена наказом по університету від “ 30 ” жовтня 2020 р. № 1486Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи Дослідження та аналіз алгоритмів розпізнавання тексту
Використання iOS фрейморків: Vision, CoreML, SwiftOCR, TesseractOCR
Проектування завдяки патерну Delegation
Використання польського інверсного запису

4. Перелік питань, що потрібно опрацювати в роботі Вступ
1. Аналітична частина
2. Теоретична частина
3. Розроблення тест-проекту на базі запропонованих рішень
4. Проведення експериментів та їх аналіз
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Презентація – 10 слайдів.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання у керівника	02.11.20	
2	Огляд існуючих алгоритмів розпізнавання тексту		
	та вибір кращих для дослідження	03.11.20-10.11.20	
3	Проектування та реалізація тест-додатку для роз-		
	пізнавання та вирішення математичних виразів	11.11.20-24.11.20	
4	Проведення експерементів та їх аналіз	25.11.20-02.12.20	
5	Оформлення матеріалів атестаційної роботи	03.12.20-07.12.20	
6	Подання атестаційної роботи керівникові та		
	її попередній захист	08.12.20-09.12.20	
7	Подання атестаційної роботи на рецензування	10.12.20-11.12.20	

Дата видачі завдання 02 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Єрьоміна Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 69 с., 13 рис., 3 табл., 1 дод., 16 джерел.

ДЕТЕКТУВАННЯ ТЕКСТУ, РОЗПІЗНАВАННЯ ТЕКСТУ, OCR, МАШИННЕ НАВЧАННЯ, ОБРОБКА ЗОБРАЖЕНЬ, МАТЕМАТИЧНІ ВИРАЗИ, SWIFT, IOS.

Дана атестаціна робота присвячена дослідженню алгоритмів та методів детектування і розпізнавання тексту математичних виразів на платформі iOS. Метою роботи є демонстрація головних особливостей зовнішніх та нативних бібліотек на основі розробки тестового програмного забезпечення.

В роботі використано основні технології для розробки мобільного додатку, проведено їх аналіз та визначено найкращий засіб для реалізації поставленої задачі.

Було розроблено тестовий додаток на основі необхідних функцій та інтерфейсу для проведення аналізу якості та швидкодії обраних рішень. Зроблено оцінку використаних та досліджених підходів. Визначено основні переваги та недоліки використаних рішень.

ABSTRACT

Master’s thesis: 69 pages, 13 figures, 3 tables, 1 appendices, 16 sources.

TEXT DETECTION, TEXT RECOGNITION, OCR, MACHINE LEARNING, IMAGE PROCESSING, MATH EXPRESSIONS, SWIFT, IOS.

This thesis is devoted to the investigation of algorithms and methods for detecting and recognizing the text of mathematical expressions on the iOS platform. The aim of the work is to demonstrate the main features of external and native libraries based on the development of test software.

In this work the basic technologies for developing a mobile application, their analysis and the best means for realization of the set task are determined.

A test application was developed based on the necessary functions and interface to analyze the quality and performance of selected solutions. An evaluation of the applied and investigated approaches is made. The main advantages and disadvantages of the used solutions are determined.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

Фреймворк – інфраструктура програмних рішень, що полегшує розробку складних систем

API – набір визначень взаємодії різнотипного ПЗ (англ., Application Programming Interface)

Apple – американська технологічна компанія, яка проєктує та розробляє побутову електроніку, програмне забезпечення й онлайн-сервіси

AR (Augmented Reality) – доповнена реальність, складова частина змішаної реальності, коли реальні об'єкти інтегруються у віртуальне середовище (англ., Augmented Reality)

Developer Tools – інструменти для розробки

Interface Builder – середовище для розробки інтерфейса корситувача

IDE – комп'ютерна програма, що допомагає програмістові розробляти нове програмне забезпечення чи модифікувати (удосконалювати) вже існуюче (англ., Integrated Environment)

ML – машинне навчання, підгалузь штучного інтелекту, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (англ., Machine Learning)

LLVM (Low Level Virtual Machine) – універсальна система аналізу, трансформації і оптимізації програм, щореалізує віртуальну машину з RISC-подібними інструкціями (англ., Low Level Virtual Machine)

OCR – оптичне розпізнавання тексту, механічне або електронне переведення зображень рукописного, машинописного або друкованого тексту в послідовність кодів (англ., Optical Character Recognition)

SDK – набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або

для певної платформи (англ., Software Development Kit)

UI – інтерфейс користувача (англ., User Interface)

Unit-тестування – модульне тестування

VR – віртуальна реальність, ілюзія дійсності, створювана за допомогою комп'ютерних систем, які забезпечують зорові, звукові та інші відчуття (англ., Virtual Reality)

WWDC (Apple Worldwide Developers Conference) – щорічна конференція розробників для платформи Macintosh програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D моделі, текстового документу тощо.

ВСТУП

Теперішнє життя складно представити без сучасних технологій, тому зараз найпотужніші корпорації, такі як Apple і Google прагнуть реалізувати найбожевільніші думки наукових фантастів.

Раніше ніхто і не думав, що звичайний телефон зможе замінити такі важливі речі як наручний годинник, будильник або ж секундомір, проте це сталося і ми навіть не помітили як це зручно в сучасному житті. Але як відомо, прогрес не стоїть на місці, тому моніторинг погоди в телевізорі або словник-перекладач перемістилися в мобільні додатки, які дають дуже швидкий доступ до потрібного ключового слова. В сьогоденні майже кожний використовує їх, але що буде, якщо вам тяжко розібрати слово або ж предмет, який вас цікавить вам незнайомий. Таким питанням турбувались і розробники технології машинного навчання ML. Тому зараз багато нових програмних рішень в мобільній розробці містять в собі хоча б одну реалізацію технології ML. Це може бути наш вище згаданий словник-перекладач з функціями визначення мови та розпізнавання складних текстів або технології зв’язані з розпізнаванням обличчя, що активно використовуються в безпеці та медицині.

Розвиток цієї галузі уже має вплив на кожного, та буде мати його в майбутньому. Адже крім ігрових технологій AR та VR, які тісно зв’язані з ML, існують і по-справжньому корисні їх імплементації для людей. Одна із них це майбутні Apple Glass, окуляри, які зможуть в собі поєднувати покращення зору та технології доповненої реальності, тобто кожна людина зможе чітко розпізнавати будь-який об’єкт, навіть якщо незнайома з ним. Цей напрямок має ще безліч прикладів, які роблять сучасний прогрес таким фантастичним.

Саме тому величезною актуальністю користується така послуга, як розробка мобільних додатків під ОС iOS. Все це стало актуальним як тільки компанія Apple зайняла лідируючі позиції на ринку мобільних телефонів обігнавши BlackBerry Limited та почала інвестувати в різноманітні стартапи.

Тому варто відзначити, що зараз без спеціальних тестових додатків навряд чи б була досягнута необхідна ефективність нових технологій. Тепер без них не обійтися і при вирішенні таких завдань, як архітектурна візуалізація, обробка контурів обличчя та розпізнавання тексту або голосу.

Головною ціллю дипломної роботи є дослідження розроблених алгоритмів розпізнавання, а також використання існуючих рідних технологій Apple для порівняння точності та швидкодії їх роботи на реалізованих тестових додатках. Адже компанія Apple постаралася створити просту і сучасну мову – Swift, яка набільш підходить для даної задачі.

Основною метою дипломної роботи є навчитися проектувати iOS додатки з новітньою технологією машинного навчання та аналізувати наявні рішення з метою визначити кращий.

Відповідно до цього на шляху досягнення мети будуть вирішені такі задачі:

а) огляд існуючих алгоритмів розпізнавання тексту;

б) вибір та обгрунтування кращих алгоритмів для дослідження;

в) створення тестого додатку з впровадженням фреймворків, які створенні на базі технологій машинного навчання;

г) оцінка використаних та досліджених підходів;

д) визначення переваг та недоліків створеного рішення.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Розпізнавання різноманітних об’єктів у сучасному житті

Розпізнавання об’єктів стало активною та складною областью досліджень. Система розпізнавання обличчя, голосу та тексту відіграє дуже важливу роль у сучасному світі. Існує багато областей, де нам потрібно індетефікувати персони, голоса або слова, цифри. Як приклад банківська перевірка, де нам потрібно визнати почерк або ідентифікувати людину за рисами обличчя. Перевірка часто зосереджена на різній техніці, яка використовується при розпізнаванн. Існує, в основному, два різних типи розпізнавання в режимі онлайн та розпізнавання за допомогою фото.

Виходить, що розпізнавання різноманітних об’єктів було вивчено з багатьох десятиліть. Система розпізнавання може бути використана для вирішення багатьох складних проблем і може зробити роботу людини легкою.

Система оптичного розпізнавання символів була вивчена протягом останніх десятиліть. У 1914 році Еммануель Голдберг розробив систему, яка читає рукописні символи та цифри і потім конвертується в телеграфний код. У той же час Едмунд Фурньє д'Альбе розробив сканер Optofon, який сканує друковану сторінку та випускає вихід. Голдберг продовжував розвивати систему розпізнавання рукописного введення даних. Через деякий час він запропонував підібрати зображення з шаблонами, що містять ідентифікацію бажання. Цей метод відомий як спосіб відповідності шаблону. Після цього Пауль У. Хандель також запропонував американський патент на технологію рукописного впорядкування шаблонів у США у 1933 році. У 1994 році інженери RCA запропонували спочатку первісний комп'ютерний тип оптичного розпізнавання символів для допомоги сліпим людям. Він призначений для перетворення рукописного звіту в перфокарти для введення в комп'ютер для допомоги у обробці відвантаження 20-25 мільйонів книг на

рік. У 1965 році Reader's Digest та RCA співпрацювали з метою створення оптичної системи розпізнавання символів. У 1985 році запропоновано структурні підходи зі статистичними методами. У цих системах символи розбиті на безліч моделей, таких як горизонтальні та вертикальні лінії та різні криві. У цьому методі система зосереджена на формі персонажів. Після 1990 року реальний прогрес досягнуто за допомогою нових методів та методологій обробки зображень та розпізнавання образів. У сучасному світі використовуються більш потужні комп'ютери та більш точні пристрої, такі як електронна ручка, сканер та планшети. Багато підходів, таких як HMM, нейронна мережа, алгоритм зворотного поширення, нечітка нейронна мережа, використовуються для розпізнавання символів та цифр.

Розпізнавання рукописних символів є одною з практично важливих проблем у програмах розпізнавання об’єктів. Основним завданням є здатність розробляти ефективний алгоритм, здатний розпізнавати цифри, написані рукописним словом, і який способом подає користувачі сканера, планшета та інших цифрових пристроїв. Незважаючи на велику кількість інструментів для написання технологічних текстів, багато людей все-таки мають унікальні почерки. Тому завжди існують недоліки в написанні тексту. Таким чином, ми вирішили вирішити цю проблему у нашому проекті, оскільки ми вважаємо, що значно більша зручність керування цифровим текстом порівняно з письмовим текстом допоможе людям більш ефективно отримувати доступ, шукати, ділитися ними та аналізувати їхні записи, досі дозволяючи використовувати їх бажаний метод письма.

Виявлення рукописного тексту - це техніка або здатність комп'ютера

отримувати та інтерпретувати зрозумілі рукописні введення з джерела, наприклад паперові документи, сенсорний екран, фотографії тощо. Коли ми

щось пишемо і подаємо як вхід до системи, то як ми розпізнаємо ці слова та рукописні документи Цей процес виконується системою розпізнавання рукописного тексту.

Отже виходить, що розпізнавання рукописного вводу – це здатність

комп'ютера або мобільного пристрою читати рукопис як фактичний текст. Тому вже існують методи та алгоритми для розпізнавання. Одним з популярних алгоритмів є оптичний метод розпізнавання символів (OCR).

OCR – це найбільш основний метод, який використовується для розпізнавання рукописних символів. Це робиться шляхом сканування рукописного запису, а потім перетворення його в основний текстовий документ. Це також працює, якщо взяти зображення рукописного тексту та спробувать просканувати його в реальному часі. Оптичне розпізнавання символів стало цікавою темою протягом багатьох років. Воно визначається як процес оцифрування зображення документа в складові символи. Уявіть собі систему, яка може розшифрувати рецепт лікаря з поганим почерком. Незважаючи на десятиліття інтенсивних досліджень, розробка OCR з можливостями, порівнянними з можливостями людини, все ще залишається відкритим викликом. За останні кілька років кількість академічних лабораторій та компаній, що займаються дослідженнями розпізнавання символів, різко зросла. OCR є складною проблемою через різноманітність мов, шрифтів і стилів, в яких текст може бути написаний, і не кажучи вже про складні правила мов. Отже, методи різних галузей інформатики, тобто обробка зображень, класифікація моделей та обробка природної мови тощо, використовуються для вирішення різних завдань.

OCR дозволяє створювати велику кількість корисних додатків в режимі онлайн і офлайн, таких як:

1) Введення даних для бізнес-документів. Хіміки / аптеки можуть сканувати рукописи лікарів без будь-якої боротьби.

2) Автоматичне розпізнавання номерного знака.

3) Автоматичні страхові документи, які можуть витягувати ключову інформацію без втручання людини.

4) Витяг інформації візитної картки в список контактів.

5) Сканування книг.

6) Зробити електронні зображення друкованих документів для пошуку,

наприклад, Google Books.

7) Перетворення рукописного тексту в режимі реального часу для керування комп'ютером (ручне обчислення).

8) Допоміжні технології для сліпих та людей з вадами зору. Уявіть, що всі дорожні знаки доступні для сліпого на відстані.

Можна застосовувати практично у всіх сферах життя. Ви просто повинні думати, де ви зустрінете письмовий текст [1].

Збіг шаблонів – це один із найпростіших і найстаріших підходів. У цьому багато шаблонів кожного слова зберігаються для вхідного зображення, обчислюється помилка або різниця з кожним шаблоном. Виводиться символ, що відповідає мінімальної похибці. Ця техніка ефективно працює для розпізнавання стандартних шрифтів, але призводить до слабкої роботи з написаними вручну символами.

Матрична відповідність передбачає порівняння зображення зі збереженим гліфом на основі пікселів на пікселі; він також відомий як «відповідність шаблону», «розпізнавання образів» або «співвідношення зображень». Це покладається на те, що вхідний гліф правильно ізольований від решти зображення, а на збереженому гліфі знаходиться такий же шрифт і в тому ж масштабі. Ця техніка найкраще працює з машинописним текстом і не працює належним чином, коли виникають нові шрифти. Ця методика реалізує ранній фізичний орієнтований на основі фотоелементів.

Розпізнавання зображень – це ще один підхід, в якому аналізується статистичний розподіл точок та витягуються ортогональні властивості. Для кожного символу вектор функції розраховується та зберігається в базі даних.

Причому розпізнавання виконується шляхом знаходження відстані вектора вектора вхідного зображення до зображення, збереженого в базі даних, і виведення символу з мінімальним відхиленням.

З геометричного підходу, з іншого боку, функції залежать від фізичних властивостей, таких як кількість стиків, відносних положень, кількості кінцевих точок, співвідношення довжини до ширини тощо. Класи, вимушувані

на основі цих геометричних ознак, досить відрізняються від незначного перекриття. Однак головним недоліком такого підходу є те, що цей підхід значною мірою залежить від набору символів, алфавітів або цифр тощо. Можливості, витягнуті для одного набору, навряд чи працюють для іншого набору символів. Ці фактори впливають на рівень розпізнавання та багато інших показників продуктивності визначення.

З огляду на величезне нагромадження зібраних і доступних даних різних компаній, у промисловості, науці, техніці, аналіз даних стає все більш важливими. Сьогодні, дані, щоб бути проаналізованими, більше не обмежуються примітивним виглядом і класичними базами даних, а все більше і більше можуть включати в себе текстові документи і веб-сторінки (інтелектуальний аналіз тексту, веб-сторінок та інше), просторові дані, мультимедійні дані, реляційні дані (соціальні мережі). Аналітичні інструменти дозволяють кінцевим користувачам збирати значущі зразки, приховані в великих обсягах структурованих і неструктурованих даних. Аналіз великих масивів даних дає користувачам можливість визначити нові джерела надходження, розвивати лояльні і рентабельні зв’язки з клієнтами, і вести загальну організації більш ефективно, а також ефективно відносно вартості.

Дані технології міцно проникли в наше життя і значно його спростили. Зараз ми вже не витрачаємо час на багато завдань, так як за нас їх вирішує машина. Однак поки не всі завдання вирішені, а навіть якщо і вирішені, то практично завжди є можливість якимось чином поліпшити запропоноване рішення. На даний момент одним з перспективних напрямків вважається розробка алгоритмів розпізнавання об'єктів на зображенні або відео.

Завдання детектування і розпізнавання текстової інформації на зображеннях представляє інтерес в галузі комп'ютерного аналізу. На даний момент різні системи використовують абсолютно різні технології для вирішення даного завдання від різних варіацій обробки зображень з метою отримання репрезентативних ознак до нейронних мереж. Різні методи дають різну точність і займають різний час на обробку. Таким чином, розробка

унікальної системи, яка могла б розпізнавати текст на різних типах відеозаписів, навіть на сильно неякісних, була б дуже актуальна. Крім того, дана розробка активно б застосовувалась в таких практичних завданнях, як розпізнавання автомобільних номерів, дорожніх знаків, перекладі тексту та іншої важливої інформації, що записується на фото або відео.

1.2 Технології машинного навчання

Застосування популярних алгоритмів машинного навчання для великих обсягів даних спричинили нові практичні проблеми. Традиційні бібліотеки машинного навчання не підтримують обчислення великі масиви даних, тому

необхідні нові підходи. Як наслідок, парадигма хмарних обчисленнь вплинули на сферу машинного навчання. Один з підходів – це використання статистичних інструментів та бібліотек, застосованих в хмарі. Інший напрям продуктів – додавання існуючих інструментів з плагінами, які дозволяють користувачам створити кластер Хадуп в хмарі. Далі в списку бібліотеки розподіленої реалізації для алгоритмів машинного навчаннчя, а також на передумові розгортання складних систем для аналізу даних і інтелектуальний аналіз даних. Останній підхід цього дослідження машинного навчання, програмне забезпечення як послуга (Software-as-a-Service), декілька стартапів великих даних (а також великі компанії) вже відкривають свої рішення на ринок [2]. Протягом більше двох десятиліть, продукти, які розробляються паралельно з базами даних, таких як Teradata, Oracle або Netezza надали кошти для реалізації алгоритмів машинного навчання в коді SQL, що є складним завданням для підтримки. Крім того, великомасштабні установки цих продуктів є дорогими і не є доступним варіантом у більшості випадків. Інший рушій для зсуву парадигми від реляційної моделі до інших альтернатив - це нова природа даних. Приблизно п'ять років тому, велика частина даних мала транзакційний характер, що складається з числових або рядкових даних, які легко зберігаються в рядках і стовпцях реляційної бази даних. З тих пір, у той

час як структуровані дані мають лінійне зростання, неструктурованих (наприклад, аудіо та відео) і напівструктуровані дані (наприклад, дані веб-трафіку, соціальний медіаконтенту і т.д.) демонструють експоненціальне зростання. Велика частина нових даних або напівструктурованного формату, тобто складається з заголовків, за якими слідують рядки тексту або чисті неструктуровані дані (фото, відео, аудіо).

У той час як останні мають обмежений текстовий зміст і є більш складним для синтаксичного аналізу, напівструктуровані дані викликали безліч нереляційних сховищ даних (NoSQL), адаптованих для обробки величезної кількості даних. Як наслідок, за останні 5 років дослідники бачили перехід до паралелізації машинного навчання з використанням цих нових платформ, таких як NoSQL сховищ даних, розподілених середовищах обробки (MapReduce), або хмарних обчислень. Варто згадати гарну метафору Бена Уертера [3], співзасновник Platfora, для обробки великих даних: «Якщо говорити у термінах промисловій революції, ми в доіндустріальній епосі ремісництва, що передує масовому виробництву. Це еквівалентно необхідності залучати експерта коваля викувати вилки і ложки для нашого обіднього столу».

Машинне навчання за своєю суттю є трудомістким завданням, тобто багато зусиль було проведено з метою зменшення часу виконання. Парадигма хмарних обчисленнь і провайдери хмарних технологій виявилися цінними елементами для прискорення платформ машинного навчання. Таким чином, популярні статистичні інструменти середовища, як R, Octave, Python - перемістились в хмару також. Є два основних напрямки для їх інтеграції з постачальниками хмарних технологій: створення кластера в хмарі з самозавантаженням його статистичних інструментів, або розширення статистичних середовищ за допомогою плагінів, які дозволяють користувачам створювати кластери Hadoop у хмарі і запустити роботу на них.

Середовища R, Octave, Mapple та аналогічні пропозиції інфраструктур низького рівня для аналізу даних, які можуть бути застосовані для великих

наборів даних, вже забезпечують постачальники хмарних технологій. Машинне навчання є те, що приходить на вершині цього і полегшує витяг корисних знань з великих обсягів даних для клієнтів, які мають слабкі знання у статистиці або взагалі не мають таких знань, тепер мають можливітсь автоматично отримувати так звані моделі знань з великих наборів даних. Спостерігається вибух стартапів протягом останніх 5 років, деякі з них ще в прихованому режимі, які пропонують послуги машинного навчання своїм клієнтам і послуги аналізу великих даних. Ці ініціативи можуть бути PaaS / SaaS платформи або продукти, які можуть бути розгорнуті на приватних середовищах. Огляд літератури і ринку, можна зробити висновок, що машинне навчання має різноманітні прояви [4]. Можна класифікувати 5 підходів:

1) Середовище машинного навчання в хмарі (створити комп'ютерний кластер в хмарі і розгорнувши його інструментами статистики).

2) Модулі для інструменів машинного навчання - доповнення статистичних інструментів за допомогою плагінів, які дозволяють користувачам створювати кластер Hadoop в хмарі і виконувати завдання машинного навчання в ньому.

3) Розподілені бібліотеки машинного навчання - колекції параллелізованних реалізацій алгоритмів машинного навчання для розподілених середовищ (Hadoop, Dryad і т.д.).

4) Комплексні системи машинного навчання - продукти, які повинні бути встановлені в приватних центрах обробки даних (або в хмарі) і пропонують інтелектуальний аналіз даних високої продуктивності).

5) Постачальники програмне забезпечення як послуг для машинного навчання - PaaS / SaaS рішень, які дозволяють клієнтам отримати доступ до алгоритмів машинного навчання через веб-сервіси).

1.3 Інструменти Apple для машинного навчання

В останні кілька років тема штучного інтелекту і машинного навчання перестала бути для людей чимось з області фантастики і міцно увійшла в повсякденне життя. Соціальні мережі пропонують відвідати цікаві нам заходи, автомобілі на дорогах навчилися пересуватися без участі водія, а голосовий помічник в телефоні підказує, коли краще виходити з дому, щоб уникнути пробок, і чи потрібно брати з собою парасольку.

Отже, машинне навчання як вважають в Apple – це процес, в ході якого система, використовуючи певні алгоритми аналізу даних і обробляючи величезну кількість прикладів, виявляє закономірності і використовує їх, щоб прогнозувати характеристики нових даних.

Машинне навчання народилося з теорії про те, що комп'ютери можуть вчитися самостійно, будучи ще не запрограмованими на виконання певних дій. Іншими словами, на відміну від звичайних програм із заздалегідь введеними інструкціями для вирішення конкретних завдань, машинне навчання дозволяє системі навчитися самостійно розпізнавати шаблони і робити прогнози [5].

Apple використовує технології машинного навчання на своїх пристроях вже досить давно: Mail визначає листи, що містять спам, Siri допомагає швидко дізнатися відповіді на питання, що цікавлять, Photos розпізнає обличчя на зображеннях.

На WWDC16 компанія представила два API на базі нейронних мереж - Basic Neural Network Subroutines (BNNS) і Convolutional Neural Networks (CNN). BNNS - частина системи Accelerate, яка є основою для виконання швидких обчислень на CPU, а CNN - бібліотеки Metal Performance Shaders, що використовує GPU [6].

Три роки тому Apple анонсувала фреймворк, значно облягаючий роботу з технологіями машинного навчання – Core ML. В його основі ідея про те, щоб взяти заздалегідь навчену модель даних, і буквально в кілька рядків коду

інтегрувати її в свій додаток.

За допомогою Core ML можна реалізувати безліч функцій:

1) детектування об'єктів на фото і відео;

2) інтелектуальне введення тексту;

3) відстеження і розпізнавання осіб;

4) аналіз рухів;

5) визначення штрихкодів;

6) розуміння і розпізнавання тексту;

7) розпізнавання зображень в реальному часі;

8) стилізація зображень;

9) і багато іншого.

Core ML, в свою чергу, використовує низькорівневі Metal, Accelerate і BNNS, і тому результати обчислень відбуваються дуже швидко. Ядро підтримує нейронні мережі, узагальнені лінійні моделі (generalized linear models), генерацію ознак (feature engineering), деревовидні алгоритми прийняття рішень (tree ensembles), метод опорних векторів (support vector machines), пайплайн моделі (pipeline models). Але Apple з самого початку не показала власних технологій для створення і навчання моделей, а лише зробила конвертер для інших популярних фреймворків: Caffe, Keras, scikit-learn, XGBoost, LIBSVM [7].

Використання сторонніх інструментів часто було не найпростішим завданням, навчені моделі мали досить великий обсяг, а саме навчання займало чимало часу.

В кінці 2017 року компанія представила Turi Create - фреймворк для навчання моделей, основною ідеєю якого була простота у використанні і підтримка великого числа сценаріїв - класифікація зображень, детектування об'єктів, рекомендаційні системи, і безліч інших. Але Turi Create, незважаючи на свою відносну простоту у використанні, підтримував тільки Python.

І ось в 2018 році Apple, крім Core ML 2, нарешті показала власний інструмент для навчання моделей - фреймворк Create ML, який використовує

нативні технології Apple - Xcode і Swift. Він працює швидко, а створення типових моделей за допомогою Create ML стає по-справжньому простим. На WWDC були озвучені вражаючі показники Create ML і Core ML 2 на прикладі додатка Memrise. Якщо раніше на навчання однієї моделі з використанням 20 тисяч зображень потрібно 24 години, то Create ML скорочує цей час до 48 хвилин на MacBook Pro і до 18 хвилин - на iMac Pro. Розмір навченої моделі зменшився з 90MB до 3MB. Create ML дозволяє використовувати в якості вихідних даних зображення, тексти і структуровані об'єкти - наприклад, таблиці. Для тестування моделі я зазвичай використовують фреймворк Vision. Він дозволяє працювати з моделями Core ML і вирішувати з їх допомогою такі завдання, як класифікація зображень або виявлення об'єктів.

1.4. Розпізнавання рукописних математичних формул

Розпізнавання математичних виразів є більш складним завданням розпізнавання, ніж розпізнавання символів, тому що крім розпізнавання безпосередньо математичного символу необхідно також розпізнати структуру математичного виразу.

У математичному виразі символи можуть бути просторово розташовані як комплекс двовимірної структури, можливо, різного характеру та розмірів символів. Всі символи, які правильно згруповані, утворюють внутрішню ієрархічну структуру. Однак належне групування символів у математичному виразі не є тривіальним. По-перше, є два типи символів. Один тип включає всі основні символи, а інший – зв’язуючі, огороджені та символи оператора. Кожен тип символів має свої критерії групування. По-друге, існують також два типи операторів, а саме явні та неявні оператори. Явні оператори - це символи операторів, тоді як неявні оператори - просторові оператори. По-третє, деякі символи можуть представляти різні значення у різних контекстах. Ці властивості разом роблять процес розпізнавання дуже складним, навіть коли всі окремі символи можуть бути розпізнаним правильно.

Розпізнавання символів та аналіз структури двовимірних візерунків широко вивчалися протягом десятиліть. Багато методів розпізнавання символів працюють з припущенням, що символи вже ізольовані один від одного. Якщо це припущення виконується, розпізнавання символів може бути просто використовувати якийсь існуючий метод. Однак така ж ситуація не трапляється на етапі структурного аналізу. Двовимірні візерунки в різних доменах зазвичай мають дуже різні простори. Хоча деякі прийоми використовуються для розпізнавання інших двовимірних візерунків можуть, в принципі, також застосовуватись до математичного розпізнавання таких методів зазвичай вимагають значних модифікацій перед тим, як їх можна використовувати. Більше того, деякі ситуації дуже специфічні для математичного розпізнавання виразів і вимагають спеціально розроблених методи.

Коли ми пишемо математичний вираз на планшеті, ми отримуємо послідовність точок. З іншого боку, якщо ми скануємо вираз із друкованого документа, то отримуємо двовимірний масив пікселів. Дані в першому випадку зазвичай розглядаються як онлайн дані, проте в іншому випадку це лінійні дані. В обох випадках, якщо ми зможемо сегментувати дані групи, так що кожна група представляє окремий символ, ми зможемо потім безпосередньо застосувати існуючий метод розпізнавання символу для вирішення його ідентичності. Потім список об'єктів з пов'язаними атрибути (включаючи місцезнаходження, розмір та ідентифікацію) повертається. Тобто, ми застосовуємо деякі методи структурного аналізу для отримання ієрархічної структури виразу.

Після етапу сегментації ми маємо список об’єктів з деякими відомими значеннями атрибутів. Лише відсутні значення - тотожності символів. Теоретично ми можемо застосувати будь-який метод розпізнавання символів, якщо він розроблений для відповідного типу даних (тобто онлайн або оффлайн). Протягом багатьох років досліджень було запропоновано різні підходи для розпізнавання символів, включаючи узгодження шаблонів,

структурні, нейронні мережі та інші статистичні підходи.

Усі вищезазначені підходи вимагають сегментування символів перед кроком розпізнавання. Однак деякі методи, такі як методи, засновані на прихованих марковських моделях (HMM), не мають цього обмеження. Підхід HMM виявився дуже ефективним у галузі розпізнавання мови. Таким чином, деякі дослідники намагалися застосувати цей підхід до онлайн розпізнавання математичних виразів. Як зазначалося вище, онлайн дані математичного виразу - це просто послідовність точок. Це аналогічно випадку з мовою, за винятком того, що мова є послідовністю акустичних сигналів. Отже, методи HMM, розроблені для розпізнавання мови, можуть бути легко модифіковані для онлайн-розпізнавання рукописних математичних виразів для досягнення одночасної сегментації та класифікації.

1.5 Огляд програмних продуктів, які здійснюють розпізнавання тексту для платформи iOS

Перед тим як розробляти програмний продукт необхідно розглянути існуючі розробки. Для платформи iOS в основному всі програми, які реалізують алгоритм розпізнавання символів є додатки перекладачі. У практично всіх додатках для перекладу є як функція перекладу в реальному часі, так і з завантажується фотографії.

1.5.1 Перекладач Google

Один з найпопулярніших додатків в App Store. Перекладач Google вміє не тільки розпізнавати текст на фото (що робить досить непогано), але і дозволяє користувачеві працювати зі звичайним перекладачем. Цю програму можна використовувати без підключення до інтернету, при скачуванні додаткових мовних пакетів. Крім того, є функція перекладу SMS, рукописного тексту (можна малювати ієрогліфи) і розпізнавання мови. В фото-перекладача

закладена можливість сприймати не тільки базові мови, а й такі, як грецька, угорська і індонезійська. Грамотний переклад останніх мов займає значно більше часу, тому краще використовувати його в разі, якщо в посланні міститься часто використовувана інформація, яка трапляється в місцях паломництва туристів і в побутовому оточенні. Крім самого перекладу, програма також виводить користувачеві сприйнятий текст і його транскрипцію. З поки недоопрацьованих моментів можна відзначити невелике зміщення деяких слів тексту при скануванні приблизно на рядок. Також якщо заблокувати екран після виведеного перекладу, а потім знову повернути телефон в робочий стан, то результат перекладу буде втрачено і доведеться проходити процес заново.

1.5.2 ABBYY TextGrabber + Translator

Перекладач компанії ABBYY, що спеціалізується на розпізнаванні тексту з фотографій. Відмінною особливістю є те, що переклад можна здійснювати з 60 вбудованих мов, без скачування додаткових пакетів. Розпізнавання тексту відбувається без використання інтернету, але для його перекладу з'єднання буде потрібно. Підтримується можливість правити перероблений текст і зберігати його на телефоні, додатково вони зберігаються у внутрішній історії додатки, де з ними також можна проводити базові операції. Через велику кількість вбудованих мов додатком складно розпізнати мову самому в процесі сканування, це займає досить багато часу. Рекомендується заздалегідь вибирати відповідний оригінальний текст мови розпізнавання для швидшої роботи програми. До несуттєвим недоліків програми відноситься той факт, що вбудована функція завантаження зображень з галереї підтримує тільки стандартні формати зображень. ABBYY TextGrabber + Translator теж не форматує текст при виведенні сканованих зображень. Тобто текст йде суцільним потоком, ігноруючи абзаци і великі проміжки, роблячи поділ тільки між словами.

1.5.3 Photomath

Проект хорватської однойменної компанії, який викликав фуров у світі дизайну та можливостей математичних калькуляторів. Цей додаток для iOS та Android дозволяє автоматично вирішувати математичні приклади, просто знявши їх на камеру. Система автоматично розпізнає формулу на фотографії, а також проводить її обчислення. Вміє вирішувати не лише прості приклади, в один рядок, а й квадратні рівняння, інтеграли тощо.

Велика частина аналогів – це онлайн розпізнавання. Проте мобільний додаток Photomath являється яскравим представником оффлайн розпізнавання, яке володіє широким функціоналом і дозволяє користувачеві не залежати від мережі.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Огляд середовища розробки Xcode

Xcode – це пакет інструментів для розробки додатків під Mac OS X і iPhone OS, розроблений Apple. Остання версія Xcode 12.2, доступна для безкоштовного звантаження через App Store або Apple Developer Connection. Оновлення можна безкоштовно скачати на офіційному сайті підтримки. На сьогодні є єдиним засобом написання «універсальних» (Universal Binary) прикладних програм для Mac OS X.

Xcode тісно інтегрований з фреймворком Cocoa. Його використовують і при розробці самої Apple Mac OS X. Цей набір інструментів включає:

1) Xcode IDE (для кодування, створення і налагодження додатків);

2) Interface Builder (для розробки користувальницького інтерфейсу);

3) SwiftUI – альтернатива розробки UI компонентів від Apple;

4) Інструменти для аналізу поведінки і продуктивності.

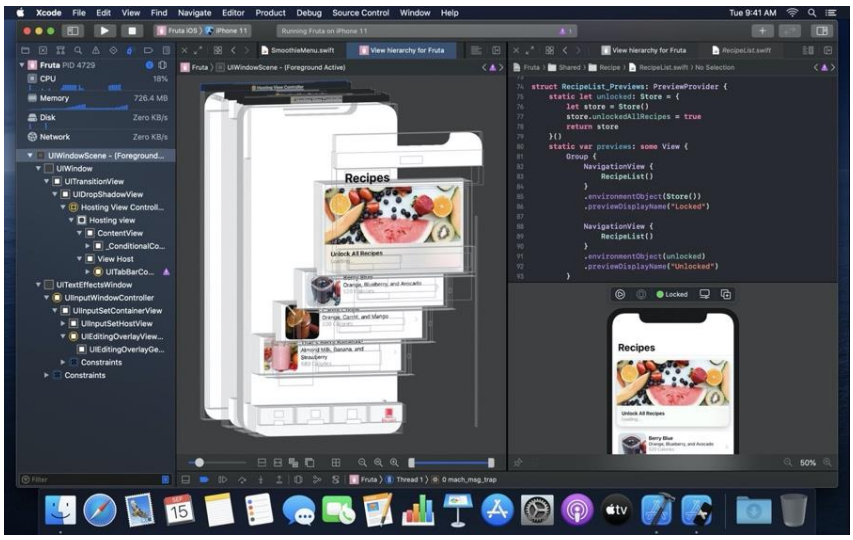

Рисунок 2.1 – Xcode IDE

Xcode IDE надає все, що потрібно для розробки: від професійних редакторів з функцією автозаповнення коду і Cocoa рефакторінга, до налаштування open-source компіляторів. Xcode IDE розроблений з нуля, щоб можна було скористатися всіма можливостями Cocoa і новітніми технологіями Apple (рисунок 2.1).

Xcode включає велику кількість можливостей, щоб полегшити розробки iOS-додатків, включаючи наступні:

1) систему управління проектом для визначення програмних продуктів;

2) середовище для редагування коду, що включає можливості підсвічування синтаксису, автозаповнення коду та індексацію символів;

3) просунуту програму перегляду і пошуку документації Apple;

4) контекстно-залежний інспектор для перегляду інформації про виділений в коді символі;

5) просунуту систему збирання проекту з перевіркою залежностей і перевіркою правил складання;

6) компілятори GCC, що підтримують мови C, C ++, Objective-C, Objective-C ++ та інші;

7) підтримка LLVM і Clang для мов C, C ++ і Objective-C;

8) вбудоване налагодження вихідного коду з використанням GDB;

9) розподілені обчислення, що дозволяють поширювати великі проекти через кілька мережевих пристроїв;

10) інтелектуальна компіляція, яка прискорює час компіляції одного файлу;

11) просунуті можливості по налагодженню коду;

12) просунуті засоби налагодження, що дозволяють робити глобальні зміни в коді без зміни його поведінки;

13) підтримка знімків проекту, які надають легше управління вихідним кодом;

14) підтримка запуску засобів продуктивності для аналізу програми;

15) підтримка вбудованої системи управління вихідним кодом;

16) підтримка AppleScript для автоматизації процесу складання;

17) підтримка налагоджувальної інформації в форматах DWARF і Stabs (налагоджувальна інформація для всіх проектів за замовчуванням генерується в форматі DWARF) [8].

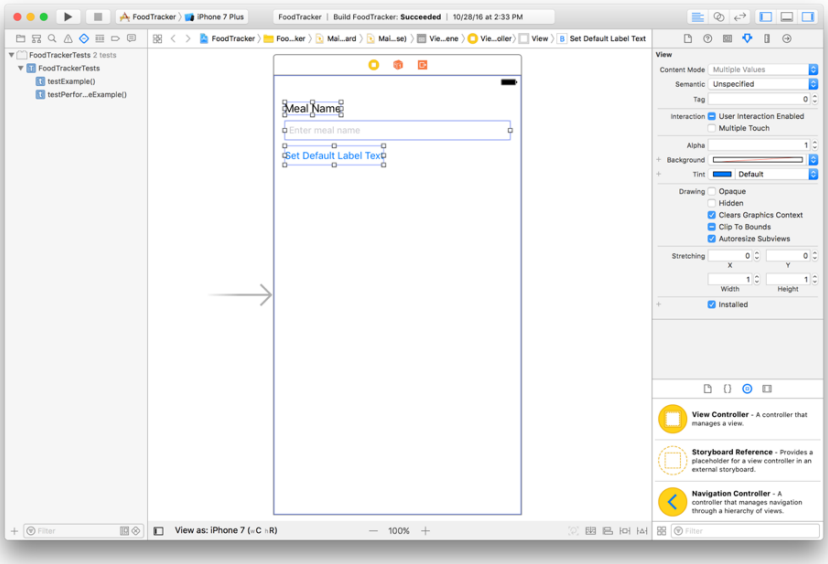

Рисунок 2.2 – Interface Builder

Interface Builder спрощує створення призначеного для користувача інтерфейсу (UI) (рисунок 2.2).. З його допомогою можна легко, без написання коду, створити шари з вікон, різні кнопки, повзунки та інші елементи управління. Потім можна перетворити цей прототип UI в реальний додаток, додавши нові можливості. Xcode працює з Interface Builder в режимі реального часу, так що можна спостерігати в графічному інтерфейсі (Interface Builder) те, що розробляється в Xcode.

Використовуючи Interface Builder, можна створювати вікна програми шляхом перетягування в нього попередньо налаштованих компонентів. Компоненти включають стандартні системні елементи управління, такі як перемикачі, текстові поля, кнопки тощо. Компоненти поміщаються вікна, після цього їх можна позиціонувати, перетягуючи по всій величині вікна,

налаштовувати атрибути, використовуючи інспектор і встановлювати зв'язки між цими об'єктами і кодом. Вміст вікна зберігається в nib-файл, який є ресурсним файлом особливого формату. Він містить всю інформацію, яка потрібна бібліотеці «UIKit» для створення тих же самих об'єктів у додатку під час виконання. Завантаження nib-файлу створює версії часу виконання всіх об'єктів, збережених у файлі, налаштовує їх в точності, якими вони були в Interface Builder. Також використовується інформація про взаємозв'язок, зазначена розробником для установки зв'язку між заново створеними об'єктами вже існуючими в додатку. Ці зв'язки забезпечують код вказівниками на об'єкти nib-файлу, а також надають інформацію самим об'єктам, необхідну для передачі дій користувача вихідному коду.

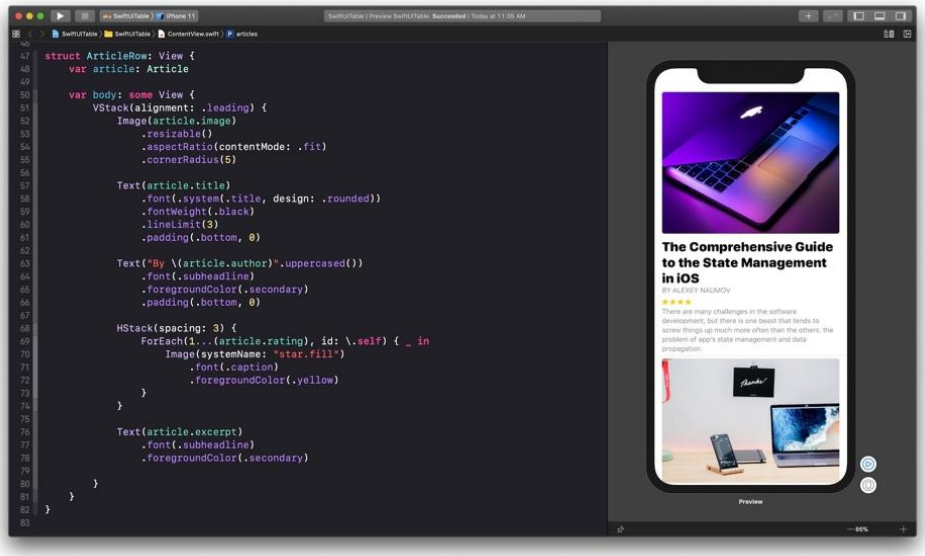

Рисунок 2.3 – SwiftUI

В цілому, використання Interface Builder зберігає величезну кількість часу, що припадає на створення UI. Interface Builder усуває написання коду, необхідного для створення, налаштування і позиціонування об'єктів, які використовуються для розробки інтерфейсу. Тому тут можна побачити, як інтерфейс буде виглядати під час виконання. UI фактично є архівами об'єктів

Cocoa, які не вимагають генерації коду. Зміни в інтерфейсі користувача (UI) не вимагають перекомпіляції (перевірки) коду, а зміни в коді не вимагають перекомпіляції UI [8].

SwiftUI – це інноваційний, надзвичайно простий спосіб створення користувацьких інтерфейсів на всіх платформах Apple за допомогою потужності Swift (рисунок 2.3). SwiftUI дозволяє повністю забути про Interface Builder (IB) та Storyboard. Тобто тепер існує можливість створювати динамічні інтерфейси швидше, ніж будь-коли раніше, використовуючи декларативне програмування на основі композиції. Рамка містить структури представлень, елементів керування та компонування для оголошення користувальницького інтерфейсу програми. Він також надає обробники подій для доставки натискань, жестів та інших типів введення у вашу програму та інструменти для управління потоком даних із моделей вашого додатка аж до представлень даних та елементів управління, з якими користувачі побачать та взаємодіють [9].

Рисунок 2.4 – Developer Tools

Величезний світ додатків для Mac і iPhone надає користувачам відмінний досвід, на який вони покладаються при створенні програм. Додаток має включати елегантний користувальницький інтерфейс і оптимальну

продуктивність. Інструменти розробника включають в себе потужні інструменти – Instruments і Shark, які дозволяють аналізувати продуктивність додатків iOS при запуску на симуляторі або пристрої. Інструменти розробника збирають дані з запущеного додатку і відображають їх графічно (шкала часу). Він дозволяє збирати дані про використання пам'яті, активності диска, мережевої активності і графічної продуктивності мобільних додатків. Тимчасова шкала може відображати всі типи інформації одночасно, що дозволяє вам співвідносити загальну поведінку додатка, а не тільки поведінку певних параметрів. Все це дозволяє вам легко визначати проблемні області вашого застосування, а потім переходити до проблемних рядків коду.

Developer Tools надають засоби для аналізу поведінки програми з плином часу (рисунок 2.4).. Наприклад, вікно середовища Developer Tools дозволяє зберігати дані декількох запусків, дозволяючи тим самим бачити, чи поліпшилося поведінка застосування або воно все ще потребує доопрацювання. Також можна зберігати дані цих запусків в документах і відкривати їх в будь-який час.

Developer Tools надають інструменти для аналізу поведінки програм з плином часу. Наприклад, вікно Developer Tools дозволяє зберігати дані з декількох запусків, щоб ви могли бачити, чи поліпшилося поведінку додатки або все ще необхідно виконати деяку роботу. Ви також можете зберегти ці дані прогону в документах і відкрити їх в будь-який час [8].

2.2 Огляд мов програмування для розробки

2.2.1 Swift

Програмування – основа основ комп'ютерної техніки. Корпорація Apple давно відома своїм умінням задавати тон розвитку індустрії на роки вперед, але з огляду на поступове сходження купертіновцев з ринку професійних рішень, надкушене яблуко асоціюється в першу чергу з споживчими товарами.

Проте, саме розробка мобільних додатків відіграє все більш важливу роль для організацій, яким необхідно спілкуватися зі співробітниками або клієнтами за допомогою вбудованих додатків. На сьогоднішній день існує великий вибір мов програмування для розробки мобільних додатків. Це пов'язано з тим, що для різних мобільних пристроїв доводиться використовувати різні мови програмування, що обумовлене тим, що мобільні пристрої мають різні операційні системи (ОС). Однак чергове дослідження громадської думки показує, що розробники ПЗ більш ніж задоволені свіжою пропозицією компанії – мовою програмування Swift [16]. Згідно з інформацією ресурсу Stack Overflow, майже 80 відсотків професіоналів із задоволенням працювали або планують працювати з випущеним не так давно інструментом розробки від Apple (рис. 2.5) [10]. Дослідникам вдалося опитати 26 тисяч відвідувачів з більш ніж 150 країн світу. Близько третини постійно зайнятих в сегменті написання мобільного ПЗ респондентів працюють в основному на платформі iOS, а менше половини також займаються створенням додатків для конкуруючої ОС Android. 20 % опитаних не змогли визначитися, швидше за все, сюди входять фахівці, які регулярно розробляють програмне забезпечення для різних платформ.

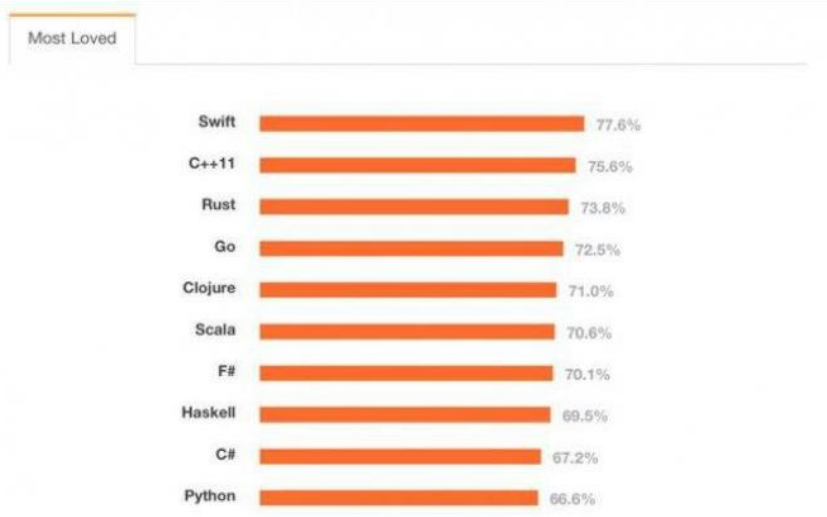

Рисунок 2.5 – Оцінки мов програмування

Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta [14]. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування.

Основним застосуванням Swift є розробка користувацьких застосунків для MacOS X і Apple iOS з використанням фреймворка Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C. Вихідний код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті.

Swift щільно інтегрований у власницьке середовище розробки Xcode і не може бути використаний відособлено на платформах, відмінних від OS X. Окремо варто відзначити, що Swift від Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache [11].

2.2.2 Objective-C

Objective-C – рефлективна, високорівнева об'єктно-орієнтована мова програмування загального призначення, розроблена у вигляді набору розширень стандартної C.

Розроблена компанією Apple, використовується в основному у Mac OS X та GNUStep – середовищах, розроблених на основі стандарту OpenStep, та Cocoa – бібліотеки компонентів для розробки програм. Програму на Objective-C що не використовує цих бібліотек можна скомпілювати для будь-якої платформи, яку підтримує gcc компілятор з підтримкою Objective-C.

Objective-C є розширенням C і тому будь-яку програму на C можна скомпілювати компілятором Objective-C.

ООП в Objective-C включає інтерфейси, класи, категорії. Реалізовано одиничне, невіртуальне спадкування. Немає єдиного базового класу для всіх об'єктів. Всі методи в класі – віртуальні. Категорія – це парадигма, яка дозволяє описувати інтерфейс з методами, які «необов'язково» імплементувати.

Синтакс Objective-C породжений одночасно від C та Smalltalk. Від останньої взято основний семантичний конструкт мови – замість виклику методу об'єктові надсилається повідомлення. Наприклад, якщо клас об'єкта obj імплементує метод doJob то говориться що об'єкт відкликається на повідомлення doJob. Щоб надіслати повідомлення doJob цьому об'єктові потрібно написати: [obj doJob];

Такий механізм дозволяє надсилати повідомлення навіть до тих об'єктів які не підтримують їх обробки. Такий підхід відрізняється від тих, що використовуються в статично типізованих мовах C++ чи Java [12].

2.3 Основні фреймворки для розробки тестового додатку.

2.3.1 Фреймворк UIKit

UIKit – це платформа, яка визначає компоненти ядра iOS-додатка від ярликів і кнопок до табличних видів і контролерів навігації. У той час, як платформа Foundation визначає класи, протоколи і функції в розробці як під iOS, так і під OS X, платформа UIKit спрямована виключно на iOS-розробку. Це еквівалент середовища розробки Application Kit або AppKit для розробки OS X.

Як і Foundation, UIKit визначає класи, протоколи і функції, типи даних і константи. Він також додає додатковий функціонал в різні класи Foundation, начебто NSObject, NSString і NSValue за допомогою використання категорій Objective-C.

UIKit забезпечує ключову інфраструктуру для реалізації графічних, заснованих на подіях iOS-додатків. Кожне iOS-додаток використовує цю бібліотеку для реалізації наступних центральних можливостей:

- управління додатком;
- управління інтерфейсом користувача;
- підтримка графіки і вікон;
- підтримка багатозадачності;
- підтримка друку;
- підтримка обробки подій дотиків і жестів;
- об'єкти для відображення стандартних системних вікон і елементів управління;
- підтримка текстового і web-вмісту;
- підтримка «Вирізання», «Копіювання» та «Вставки»;
- підтримка анімації вмісту інтерфейсу користувача;
- інтеграція з іншими додатками системи за допомогою URL-схем;
- підтримка служби push-повідомлень Apple;

- підтримка додаткових можливостей для користувачів з обмеженими можливостями;

- планування і доставка локальних повідомлень;

- створення PDF-документів;

- підтримка особливих вікон введення інформації, які працюють як системна клавіатура;

- підтримка створення особливих текстових полів, які взаємодіють з системної клавіатурою.

2.3.2 Фреймворк Vision

Apple випустила Core ML і Vision в iOS 11. Core ML дає розробникам можливість перенести моделі машинного навчання у свої програми. Це дозволяє створювати на пристрої інтелектуальні функції, такі як виявлення об’єктів. iOS 13 додав навчання на пристрої в Core ML 3 і відкрив нові способи персоналізації користувацького досвіду.

Фреймворк Vision виконує виявлення обличчя та обличчя, орієнтир, розпізнавання тексту, розпізнавання штрих-коду, реєстрацію зображень та загальне відстеження функцій. Vision також дозволяє використовувати власні моделі Core ML для таких завдань, як класифікація або виявлення об'єктів [7].

2.3.3 Фреймворк Core ML

Фреймворк Core ML розробники використовують для інтеграції моделей машинного навчання у свій додаток. Core ML забезпечує уніфіковане представлення для всіх моделей. Тому додаток використовує Core ML API та користувацькі дані для прогнозування, а також для підготовки та точного налаштування моделей на пристрої користувача [7].

Модель в Core ML є результатом застосування алгоритму машинного навчання до набору навчальних даних. Її використовують для прогнозування

на основі нових вхідних даних. Моделі можуть виконувати найрізноманітніші завдання, які було б важко або непрактично написати в коді. Наприклад, ви можете навчити модель класифікувати фотографії або виявляти конкретні об’єкти на фотографії безпосередньо з її пікселів. Також ви можете побудувати та навчити модель за допомогою програми Create ML у комплекті з Xcode. Моделі, навчені за допомогою Create ML, мають формат Core ML і готові до використання у вашому додатку. Крім того, ви можете використовувати широкий спектр інших бібліотек машинного навчання, а потім використовувати Core ML Tools для перетворення моделі у формат Core ML. Після того, як модель знаходиться на пристрої користувача, ви можете використовувати Core ML для перекваліфікації або тонкого налаштування на пристрої з даними цього користувача (рисунок 2.6).

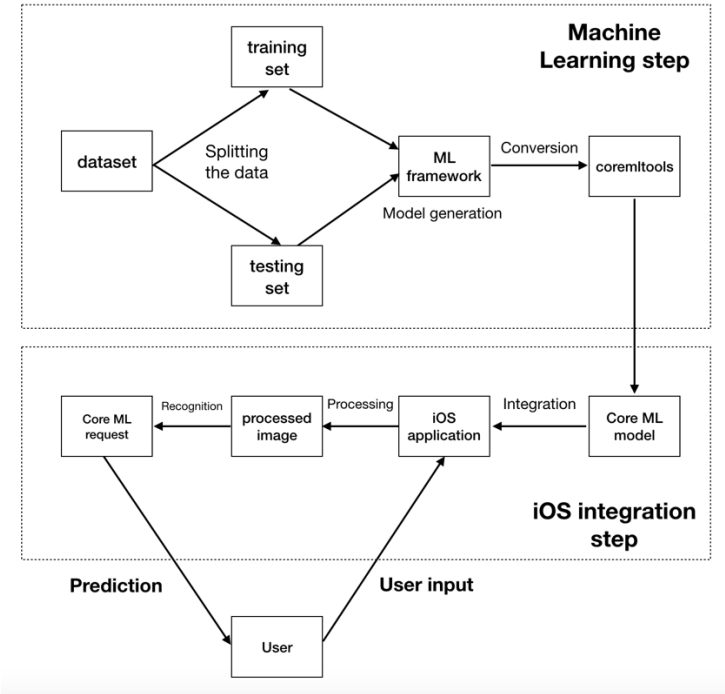

Рисунок 2.6 – Схема комунікації моделей ML з додатком

Core ML оптимізує продуктивність на пристрої, використовуючи центральний процесор, графічний процесор та Neural Engine, мінімізуючи при цьому обсяг пам'яті та енергоспоживання. Запуск моделі строго на пристрої

користувача усуває будь-яку потребу в мережному підключенні, що допомагає зберегти конфіденційність даних користувача та реагувати на вашу програму.

Core ML - це основа для доменних фреймворків та функціональних можливостей. Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення аудіо в текст та Sound Analysis для ідентифікації звуків в аудіо. Сама Core ML будується на основі низькорівневих примітивів, таких як Accelerate та BNNS, а також металевих шейдерів.

2.3.4 Фреймворк SwiftOCR

SwiftOCR - це швидка і проста бібліотека OCR, написана на Swift. Вона використовує нейронну мережу для розпізнавання зображень. На сьогоднішній день SwiftOCR оптимізовано для розпізнавання коротких, довгих в один рядок буквено-цифрових кодів.

Для розпізнавання звичайного текст, як вірш чи статтю новин, зазвичай використовують TesseractOCR, але якщо задача розпізнати короткі буквено-цифрові коди (наприклад, подарункові картки, математичні вирази), бажано зупинити свій погляд на SwiftOCR, оскільки саме він виграє в якості та швидкодії виконання данної задачі. Tesseract написаний на C ++ і йому більше 30 років. Для його використання спочатку потрібно написати для нього обгортку Objective-C ++. Основна проблема, яка сповільнює Tesseract - це спосіб управління пам’яттю. Тоді як SwiftOCR написаний приблизно 5 років тому та використовує синтаксис та управління пам’яттю Swift SDK.

Було проведено на понад 50 тестів важких зображеннях, що містять буквено-цифрові коди. Результати дивовижні. SwiftOCR обіграв Tesseract у кожній категорії.

2.3.5 Фреймворк TesseractOCR

Спочатку Tesseract був розроблений в лабораторіях Hewlett-Packard Bristol та Hewlett-Packard Co, Грілі, Колорадо, між 1985 і 1994 роками, ще деякі зміни були внесені в 1996 році для портування на Windows, а деякі зміни C ++ у 1998 році. У 2005 році Tesseract було відкрито для HP. З 2006 року розробляється Google. Остання стабільна версія (на основі LSTM) - 4.1.1, випущена 26 грудня 2019 року.

Механізм Tesseract OCR для iOS використовує конкретний тип моделі машинного навчання, який називається нейронною мережею. Нейронні мережі вільно моделюються за тими, що знаходяться в мозку людини. Наш мозок містить близько 86 мільярдів з'єднаних нейронів, згрупованих у різні мережі, які здатні вивчати певні функції за допомогою повторення. Подібним чином, у набагато простіших масштабах штучна нейронна мережа приймає різноманітний набір вибіркових входів і виробляє все більш точні результати, навчаючись як своїм успіхам, так і невдачам з часом. Ці зразки вхідних даних називаються «навчальними даними» (рисунок 2.6).

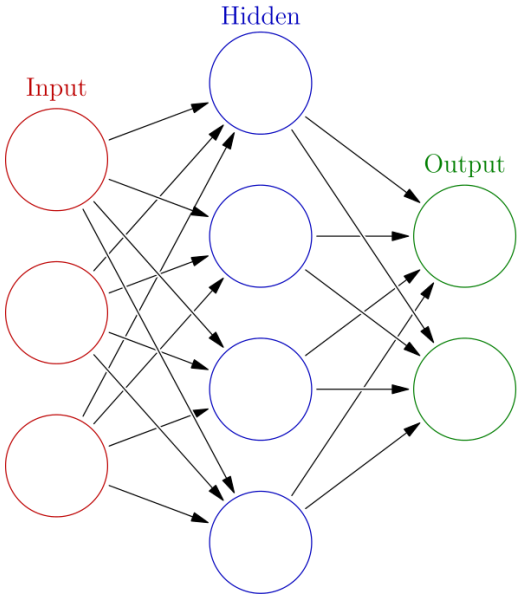

Рисунок 2.6 – Схематична модель TesseractOCR

Під час навчання системи, ці навчальні дані:

1) Входить через вхідні вузли нейронної мережі.

2) Подорожі через міжвузлові зв’язки, які називаються «ребрами», кожен з яких зважується з передбачуваною ймовірністю того, що вхідні дані повинні рухатися цим шляхом.

3) Проходить через один або кілька шарів «прихованих» (тобто внутрішніх) вузлів, які обробляють дані за допомогою заздалегідь визначеної евристики.

4) Повертається через вихідний вузол із передбачуваним результатом.

Потім цей вихід порівнюється з бажаним результатом, а крайові ваги коригуються відповідно, так що подальші навчальні дані, передані в нейронну мережу, повертають все більш точні результати. Tesseract шукає візерунки в пікселях, літерах, словах та реченнях. Tesseract використовує двопрохідний підхід, який називається адаптивним розпізнаванням. Для розпізнавання символів потрібен один прохід через дані, а потім другий прохід, щоб заповнити будь-які літери, в яких він не був впевнений, буквами, які, швидше за все, відповідають контексту даного слова чи речення.

2.4 Обґрунтування вибору підходящих рішень

Були розглянуті сучасні фреймворки розпізнавання та обробки тексту зображень, які складаються з відповідних алгоритмів. Також було обрано найбільш підходяще середовище для розробки тестових проектів – Xcode IDE та нативна мова програмування – Swift. Після огляду існуючих фреймворків та опису роботи їх алгоритмів, а також складності їх впровадження в тестовий додаток було вирішено розглянути три доступних рішення:

1) нативний Vision фреймворк в зв'язці з Core ML для інтеграції моделей ML.

2) нову, швидку і просту бібліотеку OCR, написану на Swift – SwiftOCR.

3) механізм Tesseract OCR для iOS – TesseractOCRiOS.

За допомогою аналітичних інструментів можна буде зрозуміти актуальність використання наявних рішень в кінцевому розробленому продукті. Аналізуючи велику кількость відповідних тестів з’явиться змога отримати та зрозуміти картину швидкодії роботи та точності розпізнавання доступних алгоритмів. Так як дослідження в області виявлення знань і машинного навчання поєднує в собі класичні питання інформатики (алгоритми оптимізації, системи програмного забезпечення, баз даних) з елементами штучного інтелекту і статистики, аж до питань, орієнтованих на користувачів (візуалізація, інтерактивний інтелектуальний аналіз).

З’явиться змога використати отриманий тестовий продукт при розробці універсального продукту, який зможе поєднувати в собі як внутрішні, так і зовнішні бібліотеки ML та нейромереж. А також мати унікальний та простий API для вибору кращого алгоритму для заданої задачі.

3 РОЗРОБЛЕННЯ ТЕСТ-ПРОЕКТУ НА БАЗІ ЗАПРОПОНОВАНИХ РІШЕНЬ

3.1 Постановка завдання

Необхідно розробити алгоритм та протестувати його за допомогою мобільного додатоку для операційної системи «iOS» компанії Apple, версії вище 13.0.

Додаток має являти собою екран камери, в якій робоча область може розпізнавати математичні вирази та виводити їх кінцеве значення на даний екран. (рисунок 3.1).

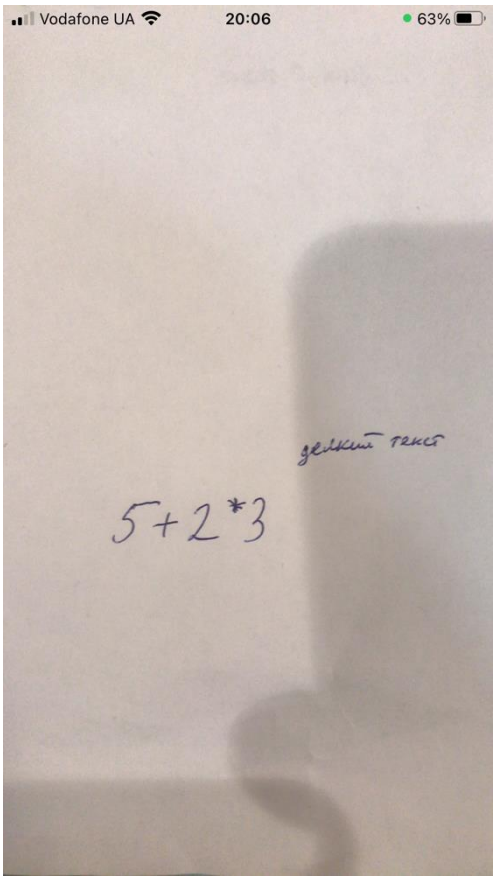

Рисунок 3.1 – Головний екран додатку

Визначення математичних виразів здійснюється в реальному часі, способом захоплення всіх можливих варіантів робочої області екрану. Захоплення проходить за допомогою фреймворка Vision та внутрішніх менеджерів по збереженню та обробці зображень. Та являє собою синій прямокутник в середені якого міститься написаний чи надрукований математичний вираз (рисунок 3.2).

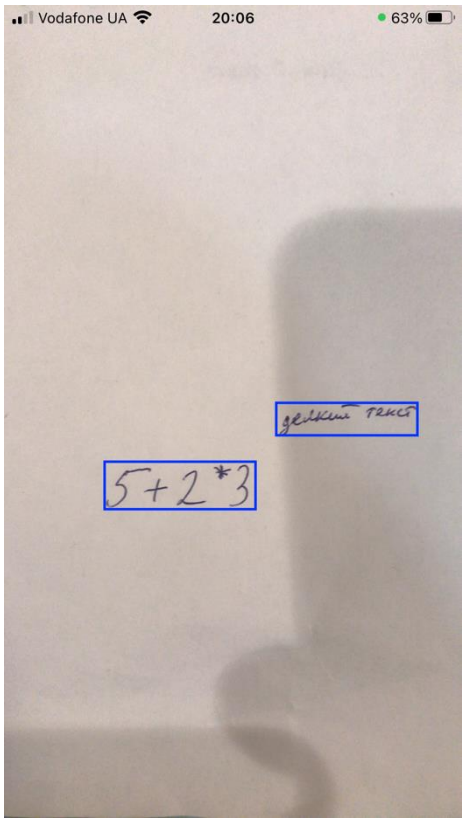

Рисунок 3.2 – Детектування тексту в реальному часі

Після захоплення області виразу потрібно натиснути на неї та почнеться обробка попередньо збереженого зображення за допомогою одного із трьох рішень по розпізнаванню тексту. Отриманий результат передається в менеджер, який приймає розпізнані символи та вирішує рівняння. Після його вирішення кінцевий результат відображається на екрані у користувача червоним кольором (рисунок 3.3).

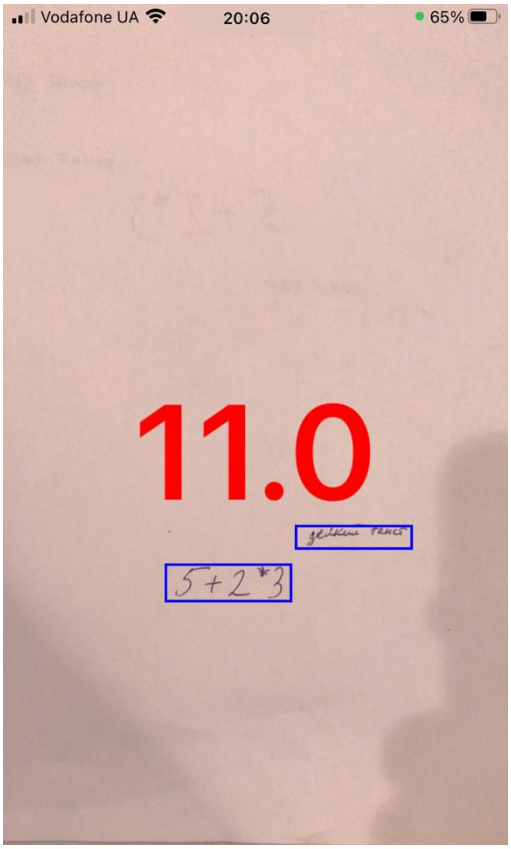

Рисунок 3.3 – Отриманий результат

Як результат, дипломний проект являє собою аналіз наявних бібліотек та розробку тестового додатку для розпізнавання математичних виразів за допомогою зовнішніх алгоритмів, рідних технологій Apple та подальшого вирішення його.

3.2 Проектування

3.2.1 Шаблон проектування Delegation

У розробці ПО, патерн (шаблон) делегування (Delegation pattern) – це спосіб, яким об'єкт зовні виражає деяку поведінку, але в реальності передає відповідальність за виконання цієї поведінки пов'язаному об'єкту. Шаблон делегування є фундаментальною абстракцією та наслідком іншого популярного патерну Observer, який дозволяє мати суб'єкт (Subjects) і об'єкти,

які на нього підписані (Observers). Але тільки Observer реалізує залежність «один до багатьох» в той час як Delegation – «один до одного».

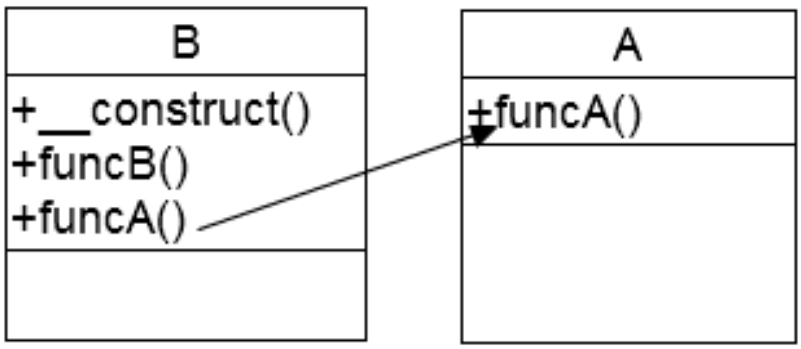

Рисунок 3.4 – Схема шаблону проектування «Delegation»

Даний шаблон є одним з найпоширеніших шаблонів проектування в iOS. Практично неможливо створити додаток для iOS без використання делегування. Більшість основних класів в iOS SDK використовують делегування. Шаблон делегування досягається шляхом використання абстрактного прошарку під назвою протокол. Протокол визначає загальний набір методів, властивостей та інших вимог, які підходять конкретному завданні або відповідають окремої функціональності. Протоколи це «абстракція», тому що вони не несуть в собі деталі реалізації ... Тільки імена функцій і властивостей. Це лише креслення, як її і позначає Apple.

Отже, делегування є гнучким, оскільки воно не вимагає, щоб делегуючий клас що-небудь знав про делегата. Головне тільки те, що він відповідає певному протоколу. Єдина життєздатна альтернатива для делегування - це використання замикань. Замість виклику делегує функції, делегує клас, викликає замикання, яке заздалегідь визначено як властивість делегує класу. Використання замикання в якості делегата має один головний недолік: ними важко керувати і організовувати, якщо ви використовуєте їх занадто багато [15].

Тому найкращим рішенням буде використання цього патерну. Адже,

дана концепція дозволяє мати в проекті декілька складних менеджерів та реалізувати швидке спілкування між ними і передачу вихідного результату.

3.2.2 Архітектура проекту

Основний алгоритм розпізнавання і вирішення математичних виразів складається з декількох менеджерів, які ділять основне завдання на кілька підзадач.

Менеджер VisionService вміє визначати текст в робочій області камери і ділити його на підобласті, які заносить в сині рамки. Він реалізований за допомогою нативної технології Apple - фреймворка Vision. При натисканні на обрану область VisionService передає певні результати в BoxService, який формує зображення і зберігає його відокремлюючи від інших знайдених результатів на екрані. Отримане зображення передається менеджеру RecognationService, який використовує один з трьох доступних фреймворків конвертації картинки в масив символів.

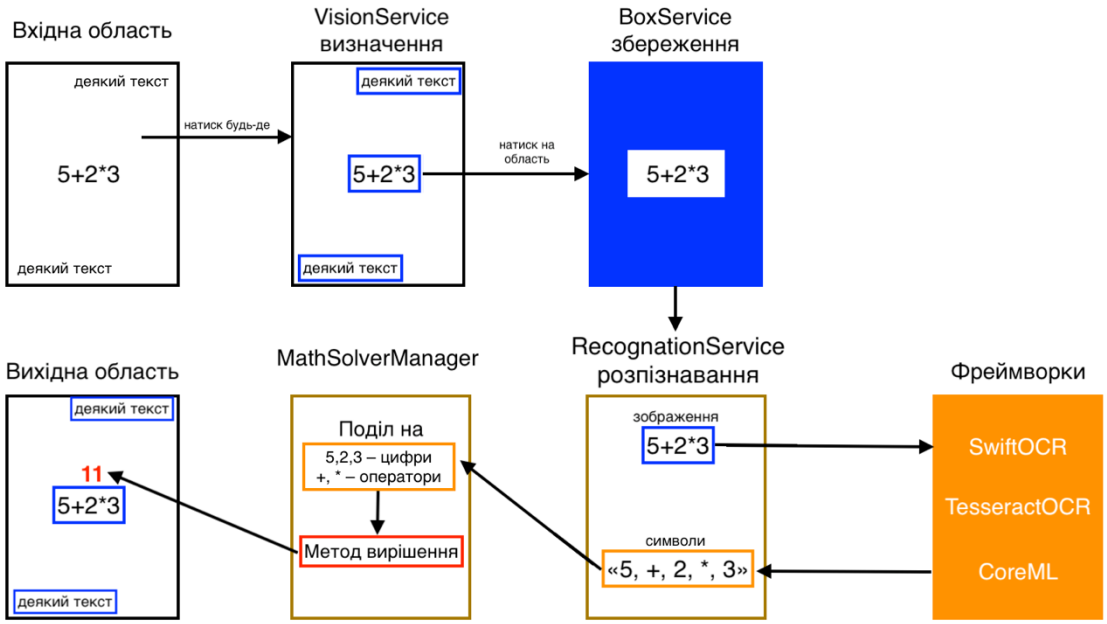

Рисунок 3.4 – Основна архітектура проекту

І далі RecognationService передає отриманий масив в MathSolverManager, який вміє відділяти цифри, арифметичні оператори від символів і розв'язувати рівняння. Далі отриманий результат повертається на вихідну область і відображається поверх рамки захопленого раніше математичного виразу.

За допомогою даного підходу ми можемо повністю якісно і правильно реалізувати потрібний алгоритм, адже коли кожна підзадача робиться окремим модулем, то це спрощує читание і розуміння коду, а також завжди є можливість замінити певний модуль на більш кращий або новий, який робить цю ж підзадачу в рази швидше.

3.3 Програмна реалізація

3.3.1 Склад і розробка тестового проекту

Тестовий проект складається з файлів вихідного коду на мові Swift (файли з розширенням swift) та декільнох реалізацій на мові Oblective C та C++, файлів з тестами функціоналу та тестовими зображеннями зображеннями, які можна розпізнати та вирішити.

Розробка проекту велася однією людиною. Таким чином мною були розроблені: основний контроллер додатку, контроллер камери, менеджер детектування тексту, менеджер зберігання зображення, менеджер конвертації зображення в масив символів, менеджер вирішення математичних виразів.

3.3.2 Менеджер детектування тексту

Менеджер детектування тексту являє собою клас, який складається з декількох методів, які мають залежність від фреймворку Vision та активуються шляхом передачі зображення робочої області від контроллера камери (приклад 3.1).

```
        private func makeRequest(image: UIImage) {
              guard let cgImage = image.cgImage else {
                  assertionFailure()
                  return
              }

              let handler = VNImageRequestHandler(
                  cgImage: cgImage,
                  orientation: CGImagePropertyOrientation.up,
                  options: [VNImageOption: Any]()
              )

              let request = VNDetectTextRectanglesRequest(completionHandler: {
[weak self] request, error in
                  DispatchQueue.main.async {
                      self?.handle(image:  image,  request:  request,  error:
error)
                  }
              })
              request.reportCharacterBoxes = true

              do {
                  try handler.perform([request])
              } catch {
                  print(error as Any)
              }
          }
```

Приклад 3.1 – Лістинг методу запиту на детектування тексту через Vision API

Детектований результат шляхом передачі через делагат отримує підписаний на нього інший менеджер (приклад 3.2).

```
      private func handle(image: UIImage, request: VNRequest, error: Error?) {
              guard
                  let results = request.results as? [VNTextObservation]
              else {
                  return
              }

              delegate?.visionService(self, didDetect: image, results: results)
       }
```

Приклад 3.2 – Лістинг передачі результату в BoxService

Дана функція передає результат в BoxService, який формує рамку для детектованого тексту.

3.3.3 Менеджер формування рамки та зберігання

Як тільки елементи на основному зображені будуть визначені потрібно сформувати рамку для кожної групи тексту, а також зберегти рамку, яку обрав користувач або тест (приклад 3.3).

```
func handle(overlayLayer: CALayer,
                image: UIImage,
                results: [VNTextObservation],
                on view: UIView) {
        overlayLayer.sublayers?.forEach({ (layer) in
            layer.removeFromSuperlayer()
        })

        // Drawing a frame for each found group of text
        let results = results.filter({ $0.confidence > 0.5 })
        results.forEach({ result in
            let normalisedRect = normalise(box: result)
            drawBox(overlayLayer: overlayLayer,
                    normalisedRect: normalisedRect)
        })

        // Processing the selected image
        guard let biggestResult = results
             .sorted(by: { $0.boundingBox.width > $1.boundingBox.width })
             .first else {
            return
        }

        let normalisedRect = normalise(box: biggestResult)
        if let croppedImage = cropImage(image: image,
                                        normalisedRect: normalisedRect) {
            store.save(croppedImage)
            delegate?.boxService(self, didDetect: croppedImage)
        }}
```

Приклад 3.3 – Лістинг формування рамки та збереження зображення

Даний етап в розпізнаванні – це попередня обробка. Цей важливий етап дозволяє виробляти згладжування, нормалізацію і апроксимацію відрізків ліній. Згладжування в даному випадку – це велика група процедур обробки зображень. Часто використовуються морфологічні оператори заповнення і стоншення. Заповнення виконується для усунення невеликих розривів і прогалин. Потоншення – це процес зменшення товщини лінії, в якій на кожному кроці області розміром в декілька пікселів ставиться у відповідність тільки один піксель «витонченої лінії». В той час як геометрична нормалізація

зображень має на увазі під собою використання алгоритмів, які усувають нахили і перекоси окремих символів, слів чи рядків, а також включає в себе процедури, які здійснюють нормалізацію символів по ширині і висоті (приклад 3.4).

```
private func normalise(box: VNTextObservation) -> CGRect {
    return CGRect(
      x: box.boundingBox.origin.x,
      y: 1 - box.boundingBox.origin.y - box.boundingBox.height,
      width: box.boundingBox.size.width,
      height: box.boundingBox.size.height
    )
}
```

Приклад 3.4 – Лістинг реалізації геометричної нормалізації

Далі результат за допомогою делегату буде переданий до наступного менеджеру, а саме RecognationService для подальшого розпізнання.

3.3.4 Менеджер розпізнавання тексту.

Данний функціонал містить в собі основні методи розпізнавання, які звертаються до API обраних фреймворків (приклади 3.5, 3.6).

```
private func handleWithSwiftOCR(image: UIImage) {
    instance.recognize(image, { string in
      DispatchQueue.main.async {
        self.delegate?.recognationService(self, didDetect: string)
      }
    })
  }
private func handleWithTesseract(image: UIImage) {
    tesseract.image = image.g8_blackAndWhite()
    tesseract.recognize()
    let text = tesseract.recognizedText ?? ""
    DispatchQueue.main.async {
        self.delegate?.recognationService(self, didDetect: text)
    }
}
```

Приклад 3.5 – Лістинг методів розпізнання на основі OCR підходу

У всіх системах оптичного розпізнавання символів спочатку виконують те чи інше поліпшення якості та аналіз зображення, поданого на обробку. У тому числі застосовуються спеціальні фільтри відновлення пошкоджених зображень, наприклад, за допомогою гіпоелліптіческой дифузії, фільтри, що усувають змазування. В ході цього етапу визначаються області, рекомендовані для розпізнавання, робляться оцінки орієнтації тексту, виділяються окремі символи і рядки. На другому етапі виконується робота щодо безпосереднього розпізнавання тексту.

В ході першого етапу обробки зображення здійснюється вибір колірної моделі, перевага віддається тим, в яких під яскравість виділяється окремий колірний канал. За рахунок ігнорування яскравості виходить стійкість до різних умов освітлення і знижується обчислювальна складність. Отримавши необхідні дані, слід здійснити сегментацію зображення. Виділяють методи на основі використання контурної інформації: градієнтні методи, методи з обчисленням других похідних та текстурні методи: статистичні та структурні. Далі йде процес безпосереднього розпізнавання тексту в вигляді окремих символів.

В той час як розпізнавання на основі машинного навчання використовує уже навчену модель, яка була сформована на основі аналізу великої кількості даних.

```
      private func handleWithNativeRecognation(image: UIImage) {
         guard let imageData = UIImagePNGRepresentation(image) else { return }
         let requestHandler = VNImageRequestHandler(data: imageData,
            options: [:])
              let request = VNRecognizeTextRequest { (request, error) in
                  guard    let    observations    =    request.results    as?
[VNRecognizedTextObservation] else { return }
                  for observation in observations {
                      let candtidates = observation.topCandidates(1)
                      for candidate in candtidates {
                          DispatchQueue.main.async {
                              self.delegate?.ocrService(self,
      didDetect: candidate.string)
                          }

                      }
                  }
              }
```

```
        DispatchQueue.global(qos: .userInteractive).async {
            try? requestHandler.perform([request])
        }
    }
```

Приклад 3.6 – Лістинг методу розпізнання на основі CoreML API

Отриманий результат являє собою масив символів або строкове представлення зображення групи символів, яке за допомогою делегату отримує менеджер вирішення математичних виразів.

3.3.5 Менеджер вирішення математичних виразів

Цей сервіс уміє розділяти символи на цифри, арифметичні оператори та робити їх валідацію (приклад 3.7).

```
private func infix2postfix(expression: String) -> String {
    return (expression as NSString)
.infixToPostfix(withOutputDecimalSeparator: "")
  }

  private func validate(expression: String) -> String {
    let set = Set("0123456789()+-*/")
    return expression
      .replacingOccurrences(of: "/n", with: "")
      .filter({ set.contains($0) })
}
```

Приклад 3.7 – Лістинг методів сортування та валідації символів

Проте метод вирішення також має залежність від алгоритму RPN (польський інверсний запис (ПОЛІЗ)), який реалізований в даному менеджері (приклад 3.8).

```
  func solve(expression: String) -> Double {
    let validatedExpression = validate(expression: expression)
    return (validatedExpression as NSString)
.evaluateInfixNotationString()
  }
```

Приклад 3.8 – Лістинг методу вирішення за допомогою RPN

Отримане значення використовуютья основним контролером для відображення результату виразу над рамкою захопленого рівняння.

3.4 Аналіз обраного рішення

На основі розробленої архітектури було реалізовано тестовий додаток для розпізнавання рукописних і друкованих математичних виразів.

Процес складається з трьох етапів - спочатку додаток визначає, які з об'єктів на зображенні є потрібними (етап виявлення). Для цього ми використали фреймворк Apple Vision.

Потім з’ясовує, що це за літери (етап розпізнавання). Розпізнавання було реалізовано за допомогою трьох доступних способів, які хоч раз могли розпізнати вираз довжиною понад 15-ти операндів і операторів. Етапи обробки алгоритмів OCR включають в себе бінаризацію, пошук текстової області, подальшу обробку, такі як коригування фону, сегментацію ліній, реконструкцію символів та обробку шумів, а також відрисовки контуру знайденої тестової області. В той час як для алгоритма на основі машинного навчання було використано навчену модель Core ML на основі моделей згорткової нейронної мережі. І нарешті виконує вирішення на основі символів записаних за допомогою форми запису математичних виразів ПОЛІЗ, в якій знаки операцій розташовано після операндів.

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА ЇХ АНАЛІЗ

4.1 Тестування алгоритмів

Програмна реалізація складається не тільки з проектного коду, а й містить деяку кількість тестів для симуляції розпізнавання тексту. За допомогою цих тестів були отримані такі дані:

1) Core ML розпізнав і вирішив 19 з 20 поданих на вхід математичних виразів;

2) SwiftOCR також розпізнав і вирішив 19 з 20 поданих на вхід математичних виразів;

3) TesseractOCR розпізнав і вирішив 18 з 20 поданих на вхід математичних виразів.

При тестуванні програми в реальних умовах при різній освітленості були отримані зовсім інші результати (таблиця 4.1). Було проведено тестування над двадцятьма різними математичними виразами, які були надруковані посередині аркуша формату А4, 32-м шрифтом.

Таблиця 4.1 – Результати тестів отриманих рішень, розпізнаних з друкованого тексту при різній освітленості.

Основні алгоритми	Висока освітленість	Середня освітленість	Низька освітленість
Core ML	20 успішних	19 успішних	17 успішних
SwiftOCR	19 успішних	17 успішних	16 успішних
TesseractOCR	18 успішних	17 успішних	17 успішних

Також проведено тестування в реальних умовах при різній освітленості над тими ж двадцятьма різними математичними виразами, проте цього рази вони були написані від руки посередині аркуша формату А4, висота символів

відповідає 32-му шрифтові (таблиця 4.2).

Таблиця 4.2 – Результати тестів отриманих рішень, розпізнаних з рукописного тексту при різній освітленості.

Основні алгоритми	Висока освітленість	Середня освітленість	Низька освітленість
Core ML	18 успішних	16 успішних	15 успішних
SwiftOCR	18 успішних	16 успішних	15 успішних
TesseractOCR	17 успішних	16 успішних	16 успішних

Після більшості успішних тестів, було проведено тестування швидкості роботи алгоритмів. Для тестування було обрано вислів сдредней довжини – (36+10)-2*8. Даний вираз складається з цифр, арифметичних операторів, а також символів відкриття і закриття дужки кількістю в 12 символів (таблиця 4.3).

Таблиця 4.3 – Результати тестів швидкості вирішеня отриманих при різних реалізаціях

Основні алгоритми	Швидкість
Core ML	1.02 с
SwiftOCR	0,95 с
TesseractOCR	0,99 с

4.2 Аналіз вихідних даних

У цій роботі було представлено дві системи OCR. У порівнянні з Core ML, набута сумарна точність розпізнавання (92,5%) достатньо добра при ідеальних умовах. А також їх швидкість розпізнавання трохи перевершує розпізнавання на основі моделі згорткової нейронної мережі, яка

використовувалася при роботі фреймворка Core ML. Експерименти показують, що система розпізнавання представлена в цій роботі є обчислювально ефективною для таких низькообчислювальних архітектур як мобільні телефони, особисті цифрові помічники (КПК) тощо.

На основі використання контурної інформації (коли кожен символ володіє чітко вираженою контурною структурою); для локалізації тексту тут використовуються такі методи, як скелетизація, виділення країв та виділення кутів, методи на основі інваріантних моментів. У разі зображення зі складним фоном швидка обробка даних, отриманих на етапі, передобработки, може представляти нетривіальну задачу.

Проте при аналізі текстової інформації (текстові зони можуть кардинально відрізнятися від фону, що дозволяє використовувати різні частотні фільтри для «пірамід» зображень). Для виявлення подрібних зон можемо спостерігати як класичні методи розподілу образів — метод опорних векторів, експертні системи роблять це. Підходи дозволяють працювати із зображеннями, із складним фоном, забезпечуючи високу обчислювальна складність через необхідність масштабування зображень.

Результати показують, що основні проблеми пов’язані з розпізнаванням OCR вирішує, але частково. Подальший аналіз визначає саме такі розбіжності:

а) різноманітність форм накреслення символів;

б) спотворення зображень символів;

в) варіації розмірів і масштабу символів.

Тобто кожен символ може бути написаний різними стандартними шрифтами, наприклад (Arial, Times, Courier), крім того існує величезна безліч нестандартних шрифтів, використовуваних в різних предметних областях. А також рукописний текст з безліччю своїх стилів.

Також можуть бути спотворення в цифрових зображеннях, які викликані:

а) шумами в зображеннях;

б) зміною нахилу символів;

в) зміщенням символів або частин символів щодо їх очікуваного положення в рядку;

г) ефектами освітлення (тіні, відблиски і т.д.);

д) спотворенням форми символу за рахунок оцифровки зображення.

Справа не лише в тому, наскільки якісно працює бібліотека - часом існують параметри, які можуть вплинути на рішення вибору однієї бібліотеки над іншою. І в плані точності Core ML обганяє OCR з достатнім відривом. Ця точка зору була підтверджена тестами зробленими в даному розділі, а також знайденими результатами при досліджені теми атестаційної роботи (рисунок 4.1).

Рисунок 4.1 – Порівняння тестів виконаних на базі фреймворків з OCR та Core ML

Хоча обробка займає більше часу на центральному процесорі (це може бути проблемою для розпізнавання в режимі реального часу), значний запас коригування затьмарює цей аспект.

Навіть у тих випадках, коли ML міг неправильно розпізнати, відсотки правильності були набагато кращими порівняно з помилками OCR.

Отже, в даному розділі проведено повний аналіз тестового ПЗ, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на три частини.

В першій з них проведено дослідження ПЗ на якість розпізнавання в різних умовах освітленності вхідної області – ідеальної (програмне тестування), високої, середньої, низької.

Другу частину присвячено аналізу швидкодії вбудованих фреймворків при розпізнаванні математичного виразу середньої довжини.

Третя частина займає більшу частину розділу і націлена на аналіз результатів на основі отриманих даних, а також результатів мого аналітичного і теоритического дослідження.

Дослідження в цій галузі ведуться вже давно. Даний тип завдань вже успішно вирішується, однак поки сучасні алгоритми не володіють таким же інтелектом, як і людина. Тому деякі об'єкти як і раніше досить складно знайти і будь-яким чином ідентифікувати за допомогою існуючих рішень. Особливо це стосується таких об'єктів як текст.

Також завдання може ще сильніше ускладнити якість вихідного зображення. Тому результат не завжди є точним і коректним. У свою чергу дану проблему можливо вирішити, використовуючи алгоритми попередньої обробки, проте дані методи вимагають значних обчислювальних ресурсів і збільшують час роботи алгоритму.

4.3 Основні враження від використання машинного навчання

Найбільша проблема Core ML полягає в тому, що моделі не можна додатково навчити після того, як користувач почне користуватися програмою. Наприклад, за умови, що символ класифікується як неправильна буква, якщо користувач міг би вказати це як вхідні дані, модель Core ML не зможе цього

дізнатися та виправити для майбутніх подібних запитів. Core ML робить прогнози лише на попередньо навчених моделях, а самого механізму машинного навчання поки немає. Будемо сподіватися, що це буде включено в наступних версіях [13].

Одним із обхідних шляхів була б функція аналізу розпізнавання користувачем та відправки звістки на сервер. Після отримання достатньої кількості таких виправлень, з'явиться можливість повторно навчити свою модель і надіслати оновлення програми. Не такий елегантний, але його можна випробувати, поки не буде кращого способу.

Іншим питанням Core ML є розмір, чим більше тестових даних ви додаєте, тим більший розмір моделі. Ніхто не встановить програму, більшу за 100-200 Мб.

У будь-якому випадку Core ML надає інженерам програмного забезпечення iOS чудовий інструмент для початку роботи з машинним навчанням. Основна роль Core ML в даний час полягає у подоланні розриву між науковими колами (що займаються процесом дослідження, розробкою алгоритмів та навчальними наборами даних) та розробниками (які не мають великого досвіду машинного навчання, але знають, як забезпечити виробництво). готові програми для реального світу). Фреймворк все ще перебуває на початковій фазі, і він значно покращиться, разом із нашими ноу-хау щодо нього та машинного навчання загалом.

ВИСНОВКИ

В результаті виконання даної атестаційної роботи було досліджено основні алгоритми розпізнавання математичних виразів та розроблено тестовий додаток для їх аналізу на платформі iOS.

Також було розглянуто основні особливості шаблонів проектування та технології розробки. Наведено основні фреймворки для вирішення поставленої задачі, обгрунтовано вибір даних технологій, адже ці інструменти найкраще підходять для розробки мобільних додатків. Розглянуті фреймворки і технології відносяться до вищого рівня архітектури, тому вони забезпечують об'єктно-орієнтовані абстракції для низькорівневих структур. Ці абстракції істотно полегшують написання коду, тому що вони зменшують обсяг коду, який необхідно написати і приховують досить складні функції авторського права, такі як реалізація OCR підходу або ML компанії Apple.

Результатом даної роботи є отримані вихідні дані, які допомогли зрозуміти концепцію алгоритмів розпізнавання тексту. А також зробити висновок, що підхід використання машинного навчання на основі моделі згорткової нейронної мережі хоч і молодий порівняно з OCR, проте має багато перспектив в майбутньому, адже, він за якістю розпізнавання не тільки не поступається, а й обходить оптичне розпізнавання символів.

В цілому ідея розробки мобільного ПЗ на основі данного аналізу є досить перспективною, адже протягом останніх років показник, що характеризує рівень попиту на мобільні додатки, котрі володіють технологією розпізнавання, зростає. Статистика дозволяє зробити висновок про те, що подальша розробка мобільного додатку актуальна і доцільна.

Головне грамотно оцінити та проаналізувати кращі технології для створення актульного рішення для споживача. Тільки корисна та сучасна розробка отримає гідне визнання з боку користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Qing Chen Evaluation of OCR Algorithms for Images with Different Spatial Resolutions and Noises – Ottawa, Canada, 2003. – 122 p.

2. Thillou C., Color binarization for complex camera-based images / Thillou C., Gosselin B. // Electronic Imaging Conference of the International Society for Optical Imaging. - 2005. – pp. 1 – 8.

3. Shafer G. Probabilistic Expert Systems. [Text] / Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996.

4. Otsu, N. A threshold selection method from grey-level histograms. / Otsu, N. // IEEE Transactions on System, Man, and Cybernetics. – 1979. – pp. 62-66.

5. Офіційний сайт блогу Habr. [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/redmadrobot/blog/418307/. – Дата доступу 24.11.2020.

6. Офіційний сайт презентацій WWDC16. [Електронний ресурс] – Режим доступу: https://developer.apple.com/videos/wwdc2016/. – Дата доступу 24.11.2020.

7. Офіційний сайт Apple Core ML. [Електронний ресурс] – Режим доступу: https://developer.apple.com/machine-learning/core-ml/. [Електронний ресурс] – Дата доступу 24.11.2020.

8. Офіційний сайт Apple Xcode. [Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode/. [Електронний ресурс] – Дата доступу 25.11.2020.

9. Офіційний сайт Apple SwiftUI. [Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode/swiftui/. [Електронний ресурс] – Дата доступу 25.11.2020.

10. Офіційний сайт Stack Overflow. [Електронний ресурс] –Режим доступу: http://stackoverflow.com/research/developer-survey-2015#tech-super. – Дата доступу 28.11.2020.

11. ANON The Swift Programming Language (Swift 2.1) / ANON – Cupertino: Apple Inc., 2014. – 528 p.

12. Stephen G. Kochan Programming in Objective-C – Boston, 2013. – 552 p.

13. Офіційний сайт блогу Martinmitrevski. [Електронний ресурс] – Режим доступу: https://martinmitrevski.com/2017/10/19/text-recognition-using-vision-and-coreml/. – Дата доступу 03.12.2020.

14. Офіційна документація мови програмування Swift. [Електронний ресурс] – Режим доступу: https://developer.apple.com/library/ios/documentation /Swift/Conceptual/Swift_Programming_Language/. – Дата доступу 26.11.2020.

15. Офіційний сайт SwiftBlog. [Електронний ресурс] – Режим доступу: https://swiftblog.org/delegirovanie-v-swift/. – Дата доступу 26.11.2020.

16. Лук’яненко Є.С., Єрьоміна Н.С. Алгоритм розпізнавання математичних виразів для iOS-пристроїв // Восьма міжнародна науково-технічна конференція. – 2020. – ст. 1–2.