МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра дискретного аналізу та інтелектуальних систем

Магістерська робота

РОЗРОБКА ДОДАТКУ ДЛЯ ВИЯВЛЕННЯ ДЕМЕНЦІЇ З ВИКОРИСТАННЯМ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Виконав: студент групи ПМіМ-23с
спеціальності
122 «Комп'ютерні науки»

Панюш В.В.

Керівник Пелюшкевич О.В.

Рецензент Ярмола Г.П.

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Дискретного аналізу та інтелектуальних систем
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Притула М.М.

"31 "серпня 2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Панюша Володимира Володимировича

1 .Тема роботи «Розробка додатку для виявлення деменції з використанням штучних нейронних мереж»

керівник роботи Пелюшкевич Ольга Володимирівна, канд. фіз.-мат. наук, доцент ,

затверджені Вченою радою факультету від **"13" вересня 2022 року № 15**

2. Строк подання студентом роботи **12.12.2022р.**

3. Вихідні дані до роботи Документації фреймворків Flutter, TensorFlow, Teachable Machine; портал з навчальними вибірками Kaggle; наукові статті за темою деменції, середовище розробки Visual Studio Code.

4. Зміст магістерської роботи (перелік питань, які потрібно розробити) Розглянути необхідні інструменти для реалізації додатку на Flutter; налаштувати середовище для розробки; підготувати навчальні приклади та навчити нейронну мережу класифікувати зображення аналового годинника; реалізувати додаток з тестом на деменцію на Flutter

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схема найпростішої ГНМ; Годинники, намальовані людьми з деменцією; Фрагмент створених класів на порталі Teachable Machine; фрагменти програмної реалізації; результат роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Ознайомлення з документацією Flutter та TensorFlow	01.09.22 – 15.09.22	
2	Ознайомлення з науковими статтями про деменцію та тест на деменцію з годинником	16.09.22 – 30.09.22	
3	Детальне вивченння інструментів Flutter та TensorFlow	01.10.22 – 15.10.22	
4	Оцінка можливості розробки додатку з тестом на деменцію з годинником за допомогою Flutter та TensorFlow	16.10.22 – 23.10.22	
5	Навчання нейронної мережі, її тестування, підготовка додаткових навчальних прикладів	24.10.22 – 31.10.22	
6	Розробка Flutter-додатку з тестом на деменцію з годинником	01.11.22 – 20.11.22	
7	Аналіз отриманих результатів та оформлення роботи	21.11.22 – 27.11.22	

Студент _____ Панюш В.В.

Керівник роботи _____ Пелюшкевич О.В.

РЕФЕРАТ

Магістерська робота складається зі вступу, трьох розділів, висновків і списку використаних джерел. Загальний обсяг складає 37 сторінок, список використаних джерел містить 27 найменувань, робота ілюстрована рисунками.

У даній роботі за допомогою фреймворку Flutter було розроблено мобільний додаток з тестом для виявлення деменції. Двома основними частинами в ході розробки було навчання нейронної мережі та її інтеграція у застосунок.

У розробленому додатку генерується випадкове число, що позначає час, який користувач повинен намалювати на схематичному зображенні аналогового годинника. Після цього у діалоговому вікні відображається результат, який позначає, наскільки правильно людина намалювала годинник. Цей додаток може використовуватись лікарями та іншими спеціалістами, напрямком роботи яких є деменція та когнітивні порушення.

Ключові слова: Flutter, штучна нейронна мережа, TensorFlow, тест на деменцію з годинником.

ЗМІСТ

ВСТУП

Сьогодні смартфони є невід'ємною частиною життя більшості людей на планеті. Компактні, але водночас потужні та багатофункціональні, ці пристрої допомагають їхнім користувачам вирішувати щоденні проблеми. Ці проблеми та завдання можуть стосуватися будь-яких сфер життя – роботи, навчання, відпочинку, саморозвитку, здоров'я тощо. За приблизно п'ятнадцять років свого існування смартфони досягнули неймовірного прогресу і цей розвиток продовжується і сьогодні. Тому важливим питанням у сфері інформаційних технологій є створення нових і підтримка дійсних мобільних додатків.

У мобільній розробці є дві вітки – нативна розробка та кросплатформна. У випадку нативної розробки створюються додатки, які працюють виключно на конкретній цільовій операційній системі та використовують всі можливості цієї системи. Двома основними мобільними операційними системами є Android від компанії Google та iOS від компанії Apple і для розробки додатку під обидві платформи необхідно організовувати дві команди.

Альтернативою нативній розробці є кросплатформна, яка дозволяє створювати додатки, які працюють на кількох платформах. Перевагою кросплатформної розробки є економія ресурсів, оскільки потрібна лише одна команда для створення додатку. Проте такий варіант розробки обмежує використання можливостей платформи, оскільки розробка ведеться під деяку узагальнену модель декількох операційних систем. Тому при виборі способу розробки, звісно, слід враховувати вимоги та можливості замовника.

Сьогодні одним з найпопулярніших засобів для кросплатформної розробки є фреймворк Flutter. Він був створений компанією Google у 2018 році і за відносно короткий період досяг розвитку як в технічному, так і в маркетинговому напрямку.

Іншим розділом інформаційних технологій, який стрімко розвивається є штучний інтелект, зокрема, застосування штучних нейронних мереж (ШНМ) – обчислювальних систем, що моделюють роботу мозку людини для вирішення різних завдань – розпізнавання і генерування зображень та відео, машинний

переклад, прогнозування, класифікація тощо. З розвитком штучних нейронних мереж, людина буде мати змогу виконувати все менше і менше об’єму роботи самотужки, а натомість лише слідкувати за роботою програм на основі ШНМ.

Багато розділів інформаційних технологій тісно пов’язані між собою і нерідко використовується поєднання різних технологій в одному продукті. Не є винятком і комбінація мобільних технологій зі штучними нейронними мережами.

Сьогодні важливою проблемою в медицині є вивчення деменції, оскільки нею хворіє багато людей у всьому світі. Оскільки деменція означає порушення багатьох базових навичок людини (зокрема, когнітивних), то ця хвороба негативно впливає як на життя пацієнтів, та і на їхнє оточення. Одним з важливих факторів у вивченні хвороби є її діагностика та виявлення на ранніх стадіях. У контексті деменції для виявлення хвороби часто використовується тест з годинником, у якому людині пропонують намалювати аналоговий годинник із заданим часом.

Це дослідження є новим, оскільки після проведених пошуків, не було знайдено жодного мобільного додатку в публічному доступі, який би надавав змогу пройти тест на деменцію з годинником.

Очевидно, на даний момент у більшості випадків використовуються звичайні методи малювання та оцінки результатів тесту, яку проводить спеціаліст. Оскільки в наш час відбувається все більше інтеграцій комп’ютерних технологій з різними сферами, зокрема, з медициною, то ця робота буде вагомим кроком вперед у цьому напрямку та буде цікавою для спеціалістів у напрямку вивчення деменції.

Метою цієї роботи є дослідження можливостей застосування штучних нейронних мереж у додатках, написаних за допомогою Flutter і розробка додатку, який використовує ШНМ для визначення ризику розвитку деменції в людини. Ця тема є актуальною, оскільки об’єктом дослідження є новітні прогресивні технології та можливість їхнього використання в медицині.

РОЗДІЛ 1
ОГЛЯД ТЕХНОЛОГІЙ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Flutter

Flutter – це фреймворк з відкритим вихідним кодом, розроблений компанією Google. Використовується для розробки мобільних додатків під операційні системи Android та iOS, десктопних аплікацій під платформи Windows, macOS та Linux, а також веб-додатків.

Перша версія фреймворку під назвою «Sky» була представлена у 2015 році на конференції Dart-розробників [1]. Вона дозволяла створювати додатки під Android без використання звичної на той момент Java. При цьому команда розробників Sky робила акцент на швидкості – за допомогою Sky можна було розробляти додатки з швидкістю рендерингу у 120 кадрів на секунду, у той час як звичним стандартом для більшості гаджетів і розробників була швидкість у 60 кадрів за секунду. Це був великий крок вперед, що робило Sky, а в подальшому і Flutter потенційним майбутнім лідером за швидкістю в розробці мобільних додатків.

Перша стабільна версія фреймворку вийшла 4 грудня 2018 року і була представлена на конференції Flutter Live [2]. Команда Google представила цю версію слоганом з чотирьох слів – «Beautiful, Productive, Fast, Open» (англ. – «Прекрасний, Продуктивний, Швидкий, Відкритий»). Від тоді Flutter стрімко набирав популярності в індустрії розробки. На момент написання цієї роботи репозиторій Flutter на Github [3] позначили зіркою вже понад 147 тисяч користувачів, і ця цифра продовжує зростати.

Flutter привернув увагу не лише розробників, але й замовників, які бажають, щоб мобільний додаток, який представляє їхній продукт, був створений за допомогою цього фреймворку. Серед відомих компаній, додатки яких написані на Flutter є BMW, Tencent, Toyota, eBay та інші [4]. Також на Flutter написаний популярний фінансовий застосунок Google Pay. За словами розробників Google

Pay, перехід на Flutter зменшив витрати інженерних ресурсів на 70% та кількість коду на 35% [5].

Поточною стабільною версією Flutter на момент написання цієї роботи є версія 3.3.9, яка вийшла 23 листопада 2022 року [6].

Перевагою Flutter є відкритість вихідного коду фреймворку. Завдяки цьому, розробники можуть переглядати код класів та функцій і робити свій внесок в структуру цього програмного каркасу. Це може бути як виправлення помилок, так і додавання нового функціоналу, або ж пропозиції щодо оптимізації вже існуючих елементів.

Крім цього, завдяки відкритості коду розробники можуть створювати бібліотеки з різним функціоналом для вирішення різних завдань. Це значно спрощує роботу для інших розробників, оскільки їм не потрібно власноруч писати необхідні класи та функції, а натомість вони можуть під'єднати бібліотеку до свого проекту. Усі бібліотеки мають відкритий код та опубліковані на сайті pub.dev [7]. Завдяки цьому спрощується підтримка бібліотек, оскільки їхніми авторами часто є ентузіасти, в яких не завжди є достатньо ресурсів для підтримки і розвитку бібліотеки. Втім, контроль за внесками інших розробників у код бібліотеки (схвалення чи відхилення змін) все ж залишається за її автором. Більшість бібліотек є безкоштовними для використання, проте є і такі, доступ до функціоналу яких є платним. Оскільки код бібліотек є відкритим, то зазвичай обмеження доступу організовується за допомогою API ключів.

Таким чином, розвиток фреймворку відбувається не лише завдяки Flutter-команді в Google, але й завдяки розробникам з різних куточків планети, з різним досвідом та кваліфікацією.

Спільнота навколо Flutter росте разом із розвитком самого фреймворку. У мережі Інтернет з'являється дедалі більше інформації про Flutter. Статті на популярному порталі Medium, відео-презентації та відео-уроки на YouTube, обговорення на сервісах GitHub, Stack Overflow, Gitter та ін. – усе це не лише допомагає розробникам-початківцям полегшити своє знайомство з технологією, але й привертає увагу інших, більш досвідчених розробників.

Основними елементами в архітектурі Flutter [8] є:

а) платформа Dart;

б) Flutter-рушій;

в) базова бібліотека;

г) набори віджетів.

Розглянемо детальніше кожен з них.

На відміну від багатьох відомих платформ (наприклад, React Native, Tabris.js, Native script) Flutter не використовує JavaScript. Натомість в якості мови програмування була вибрана Dart – мова програмування, розроблена компанією Google, вперше представлена 10 жовтня 2011 року [9]. Розробники на той момент вважали, що Dart у довгостроковій перспективі стане заміною JavaScript, і такі плани справді були в команди розробників мови, які хотіли інтегрувати віртуальну машину Dart у Chrome. Проте згодом вони відмовились від цієї ідеї, анонсувавши в березні 2015 року нову стратегію розвитку Dart без прив'язки до браузера [10]. Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерах, визнано недоцільним. У вище згаданій стратегії також було вказано, що у веб-розробці Dart буде розроблятися в напрямку кращої інтеграції з JavaScript, з метою зробити Dart проміжною мовою, яка буде компілюватись в JavaScript. За задумом команди розробників мови Dart, такий підхід повинен дозволяти писати більш якісний код для веб-розробки. При цьому розвиток віртуальної машини Dart VM продовжувався у напрямку застосування в розробці серверних і мобільних аплікацій.

Перша стабільна версія Dart вийшла 14 листопада 2013 року [11], а поточна версія – 2.18.5 – 23 листопада 2022 року [12]. Незважаючи на те, що Dart і Flutter тісно пов'язані між собою і в деяких джерелах можна зустріти словосполучення «Flutter/Dart», ці поняття не варто ототожнювати, оскільки Dart використовується не лише в розробці мобільних додатків у Flutter.

Dart має схожий на Java синтаксис, проте не вимагає явного визначення типів. Це дозволяє створювати як невеликі програми без жорсткої структури, так і високо

масштабовані великі модульні проекти, з потребою більш явної типізації для того, щоб уникнути плутанини і помилок.

Важливою особливістю Dart є технологія null-safety [13], яка була представлена у версії 2.12. Це принцип, згідно з яким у програмному коді не можна звертатись до об’єкту, який потенційно може мати значення null. Таким чином мінімізується кількість null-помилок в програмі, оскільки практично всі потенційні null-помилки виявляються ще на етапі компіляції. Null-safety у Dart досягається за рахунок строгого поділу типів на nullable (ті, які можуть мати значення null) та non-nullable (ті, які точно ніколи не будуть мати значення null). Для звернення до nullable об’єктів потрібно спочатку провести перевірку чи їхнє значення не null. Null-safety у Dart є аналогом таких же інструментів, представлених у інших мовах програмування, наприклад, C#, Kotlin, Swift та інших.

Flutter-рушій забезпечує високу швидкість рендерингу – 120 кадрів на секунду. Такої швидкості могла досягяти ще перша версія фреймворку Sky. Flutter не використовує нативних компонентів, тому немає необхідності писати ніяких прошарків для комунікації з ними. Натомість весь графічний інтерфейс відмальовується всередині Flutter-рушія, подібно до рендерингу в ігрових рушіях. Такий підхід і забезпечує швидкість рендерингу, рівнозначну швидкості в нативній розробці. Flutter-рушій написаний в основному на C++ і використовує графічну бібліотеку Google Skia, яка також використовується в таких продуктах від Google як Google Chrome, Chrome OS, Chromium OS.

Базова бібліотека (Foundation library) [14] – бібліотека, написана мовою Dart, що включає в себе основні класи та функції, які використовуються для розробки додатків на Flutter. Вона включає в себе, наприклад, такі класи як ChangeNotifier (використовується для реагування на зміни об’єкта), Key (ключ для віджета), константу required (для позначення обов’язкових полів) тощо.

Віджети – це компоненти, які є основними елементами у побудові графічного інтерфейсу додатків, написаних на Flutter. Однією з тез фрейморку є «Everything is a widget», тобто «Усе є віджетом». На відміну від інших фреймворків, які

розділяють представлення, контролери та інші властивості, Flutter використовує уніфіковану об’єктну модель, якою і є віджет.

Віджет може визначати будь-який елемент графічного інтерфейсу додатку:

а) структурні елементи (кнопки, меню, текстові поля та ін.);

б) елементи компонування (колонки, рядки, віджети, які визначають відносне положенння елементу та ін.);

в) елементи обробки жестів (натискання, натискання з утриманням та інші способи взаємодії з екраном пристрою);

г) інші елементи інтерфейсу.

Віджети формують ієрархію, яка заснована на композиції, тобто на побудові дерева віджетів. Кожен віджет є вкладеним в інший і успадковує значення властивостей від свого батька. Тому майже всі віджети мають властивість «child» або «children» і можуть мати в залежності від свого призначення одного або багатьох нащадків у дереві. Є і такі віджети, які не мають нащадків, наприклад, Text. Кореневим віджетом зазвичай є один з класів MaterialApp, CupertinoApp, WidgetsApp або обгортка одного з цих віджетів. За рендеринг віджета відповідає метод build, який програмно повертає дерево віджетів (яке може складатись і з одного об’єкту), а графічно – відмальовує це дерево на екрані пристрою.

Віджети нерідко складаються з багатьох інших, невеликих віджетів, які відповідають переважно за якусь одну властивість компонента. Наприклад, Container, часто використовуваний віджет, складається з кількох віджетів, які відповідають за компонування, кольори, позиціонування та розміри. Таким чином, розробники теж можуть створювати нові віджети, які будуть виконувати різноманітні ролі та мати різноманітні властивості.

Існує два основних підкласи віджетів у Flutter:

а) віджети, які мають сталий стан (StatelessWidget);

б) віджети, стан яких може змінюватись у ході роботи додатку (StatefulWidget).

Stateless-віджети не змінюють свій стан під час роботи додатку й описують ті частини графічного інтерфейсу, які є статичними. Вони є корисними для опису

об’єктів, які не залежать від ніяких конфігурацій, окрім тієї, яка була передана в цей віджет під час його першого відмальовування.

Найпоширенішими прикладами Stateless-віджетів є Container, AlertDialog, MaterialButton, Theme, GestureDetector.

У свою чергу Stateful-віджети зберігають стан, який може змінюватись під час роботи аплікації. Зазвичай зміна стану відбувається в залежності від певної взаємодії користувача з додатком або ж при роботі з даними, які завантажуються зі сторонніх джерел.

Часто використовуваними прикладами Stateful-віджетів є Navigator, TextField, Scaffold, MaterialApp, BottomAppBar.

Іншими підкласами віджетів, крім StatelessWidget та StatefulWidget є:

а) PreferredSizeWidget;

б) ProxyWidget;

в) RenderObjectWidget.

1.2 Штучні нейронні мережі

Штучні нейронні мережі – це обчислювальні системи, які моделюють роботу мозку тварин. Нейрони у штучних нейронних мережах можуть містити один або більше входів. Після подання сигналів на ці входи обчислюють лінійну комбінацію сигналів з вагами входів, цю комбінацію передають аргументом у функцію активації. Отриманий результат передається на вихід нейрона. Штучні нейронні мережі будують з’єднавши виходи одних нейронів із входами інших.

Одним з поширених підвидів ШНМ є глибинні нейронні мережі – штучні нейронні мережі, які містять один або більше прихованих прошарків нейронів між вхідним та вихідним шаром. Схематично найпростіша ГНМ з одним прихованим шаром може бути зображена за допомогою схеми на рисунку 1.1 [15].

Важливим етапом в роботі зі штучними нейронними мережами є навчання мережі. Під час цього процесу формують навчальну вибірку з деякими даними, наприклад, із зображеннями та по черзі подають приклади з вибірки на входи мережі. Якщо результат на виході мережі збігається з очікуваним, то переходять до

наступного прикладу. У протилежному випадку корегують ваги входів нейронів. Якщо процес навчання побудований грамотно, то після його завершення нейронна мережа має відкореговані ваги та здатна вирішувати поставлене перед нею завдання.

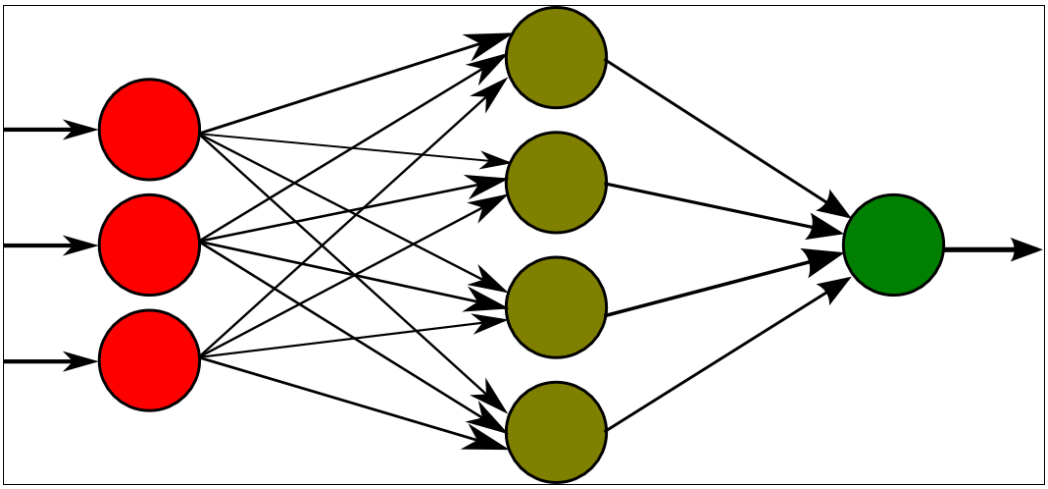

Рисунок 1.1 – Схема найпростішої ГНМ

Штучні нейронні мережі мають досить широке застосування – їх використовують у задачах розпізнавання, прогнозування, кластеризації, машинному перекладі та інших.

1.3 Тест на деменцію з годинником

Деменція є однією з найпоширеніших неврологічних хвороб у світі. Згідно з дослідженням Всесвітньої організації охорони здоров'я, на деменцію страждає більше, ніж 55 мільйонів людей по всій планеті, а число нових діагнозів цієї хвороби на рік сягає 10 мільйонів [16].

На сьогодні, на жаль, немає методів лікування деменції, проте раннє виявлення ознак хвороби та вжиті заходи можуть значно спростити життя хворих.

Одним з методів діагностики деменції є тест на деменцію з годинником – це простий інструмент для перевірки наявності ознак розвитку деменції в людини, включаючи хворобу Альцгеймера [17]. Ця перевірка часто використовується в поєднанні з іншими тестами, проте може використовуватись і без додаткових досліджень.

Метою тесту є визначення того, чи є в людини втрата когнітивних навичок. Під когнітивними навичками зазвичай розуміють здатність розуміти, пізнавати, вивчати, усвідомлювати, сприймати і переробляти зовнішню інформацію. Цей метод здатен визначати ментальні відхилення, оскільки люди з деменцією часто мають труднощі з розпізнаванням аналогових годинників. Це пов’язано з тим, що «читання» годинників вимагає розуміння розміщення стрілок годинника і часу, який це розміщення означає. Ця здатність часто втрачається навіть на ранніх стадіях деменції.

Суть тесту полягає в тому, що лікар (або інша компетентна особа) дає пацієнту завдання намалювати аналоговий годинник з конкретно заданою годиною. Для проходження тесту необхідні лише олівець та аркуш паперу (або інші засоби малювання). Часто лікарями використовується година «десять хвилин по одинадцятій», проте в загальному можна задавати будь-який час. Також існують різні варіації об’єму роботи, який потрібно виконати пацієнту:

а) намалювати лише стрілки годинника, які показують заданий час, на малюнку годинника без стрілок;

б) намалювати цифри годинника та стрілки, які показуються заданий час, на намальованому колі;

в) намалювати годинник із заданим часом на повністю чистому аркуші паперу.

Також лікар може вводити різні особливості для ускладнення проходження тесту, наприклад, обмеження в часі або уникнення словосполучення «стрілки годинника» (для того, щоб не давати людині, яка проходить тест підказки, що саме вона повинна зробити в деталях).

Під час проходження тесту в людини задіюються наступні навички:

а) розумові навички, такі як робота пам’яті та гнучкість мислення;

б) здатність розрізняти відносне розташування об’єктів у просторі;

в) здатність відтворювати послідовності та рух в абстрактній формі;

г) увага та концентрація.

Перевагами тесту з годинником є:

а) він є швидким та простим – тест може бути виконаний за кілька хвилин та потребує лише будь-яких засобів малювання;

б) він є простим для спеціаліста;

в) окрім деменції, тест допомагає виявляти ознаки інших хвороб, таких як делірій, печінкова енцефалопатія, а також допомагає оцінити рівень відновлення від мозкової травми.

Недоліком тесту є те, що він не дозволяє визначити тип деменції.

Приклади годинників, намальовані людьми з деменцією, зображені на рисунку 1.2 [18].

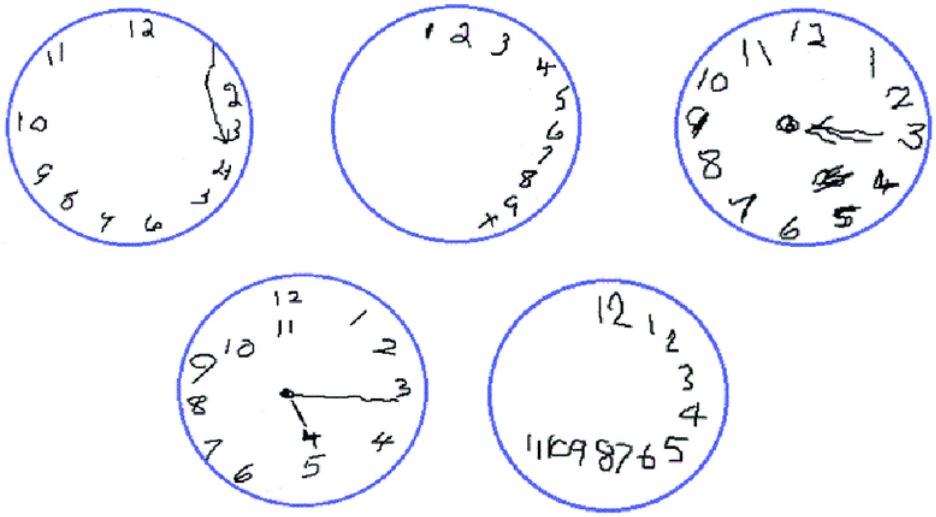

Рисунок 1.2 – Годинники, намальовані людьми з деменцією

1.4 Постановка задачі

Задача магістерської роботи полягає в наступному:

а) розглянути необхідні інструменти для реалізації додатку на Flutter;

б) налаштувати середовище для розробки;

в) підготувати навчальні приклади та навчити нейронну мережу класифікувати зображення аналового годинника;

г) реалізувати додаток з тестом на деменцію на Flutter.

При розгляді необхідних інструментів для реалі зації додатку в першу чергу потрібно ознайомитись зі способами налаштування Flutter SDK. Також необхідно

розглянути найпоширеніші елементи, які застосовуються при розробці додатків на Flutter та які, зокрема, можуть бути використані в програмній реалізації додатку з тестом на деменцію. Важливим є також вивченння інструментів обробки стану, оскільки це є одним з ключових факторів у розробці сучасних додатків. Крім того, варто розглянути інструменти для навчання нейронної мережі та її інтеграції у Flutter-додаток.

Наступним етапом дослідження є налаштування середовища розробки, яке буде використаним для програмної реалізації додатку. При виборі середовища важливо враховувати такі фактори, як наявність плагінів для роботи з Flutter та зручність використання.

Перед безпосередньою програмною реалізацією потрібно підготувати навчальні приклади зображень з аналоговими годинниками та на їхній основі за допомогою інструментів, що були розглянуті на попередніх етапах, навчити нейронну мережу класифікувати зображення годинника.

На етапі програмної реалізації необхідно розробити застосунок, основним функціоналом якого буде малювання користувачем стрілок годинника на екрані, передача зображення в нейронну мережу для класифікації та відображення отриманих результатів.

РОЗДІЛ 2
НЕОБХІДНІ ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ

2.1 Flutter SDK та середовище програмування

Для розробки на Flutter насамперед необхідно завантажити Flutter SDK. Це можна зробити або шляхом завантаження zip-архіву з офіційного сайту [19] або шляхом клонування Github-репозиторію [3].

Крім цього, потрібно обрати середовище програмування. Це можуть бути такі середовища як Android Studio, IntelliJ, VS Code, або Emacs [20]. У вибраному середовищі потрібно налаштувати Flutter плагін.

2.2 Найпоширеніші віджети у Flutter

Розглянемо деякі найпоширеніші віджети, які використовуються у Flutter на прикладі методу build екрану з текстовим полем та кількома декоративними та семантичними елементами (рисунок 2.1).

```
Widget build(BuildContext context) => Scaffold(
    appBar: AppBar(title: Text(widget.title)),
    body: Padding(
      padding: EdgeInsets.all(20),
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: <Widget>[
            Container(
              child: Text('Hello'),
              alignment: Alignment.center,
              height: 200,
              decoration: BoxDecoration(
                shape: BoxShape.circle,
                color: Colors.yellow,
              ), // BoxDecoration
            ), // Container
            TextField(decoration: InputDecoration(hintText: 'Type something'),),
          ], // <Widget>[]
        ), // Column
      ), // Center
    ), // Padding
  ); // Scaffold
```

Рисунок 2.1 – Метод build простого екрану Flutter-додатку

Перший віджет, який повертає метод це Scaffold. Scaffold – це віджет, який описує базову структуру аплікації, що використовує матеріальний дизайн. У ньому можна задати такі характеристики, як випадаючі меню, нижнє меню, панель додатку та ін. Scaffold займає весь доступний йому простір. Зазвичай це означає, що він повністю займає екран пристрою. Коли на екрані пристрою з'являється клавіатура, властивість viewInsets предка віджета Scaffold MediaQuery змінюється і Scaffold перебудовується. Scaffold спроектований як єдиний контейнер верхнього рівня в MaterialApp, тому немає необхідності робити вкладені Scaffold. В іншому випадку, це може порушити стабільну роботу додатку, особливо при роботі з текстовими полями, коли на екрані регулярно з'являється клавіатура пристрою.

У даному прикладі Scaffold має дві властивості – appBar та body. Властивість appBar та однойменний віджет позначають панель аплікації, на якій можуть розташовуватись заголовок екрану, панель з кнопками та елемент, який відкриває випадаюче меню. В body міститься тіло віджета Scaffold, тобто основна його частина.

Значенням параметру body в даному прикладі передається Padding. Це віджет компонування, в якого є всього дві властивості – padding (внутрішні відступи) та child (віджет-нащадок). Його нащадком тут є Center – ще один віджет компонування, функція якого – розташування нащадків по центру відносно віджета-предка. Альтернативою до нього є віджет Align, в якому через властивість alignment можна передавати значення, які будуть розташовувати нащадків не лише по центру, але і з іншим компонуванням, наприклад, в правому верхньому куті, по центру внизу тощо.

Наступним віджетом у дереві є Column – віджет, який розташовує список нащадків у вигляді колони. Список нащадків, звичайно, може складатись і з одного елементу, проте в такому разі використання Column буде недоцільним. Крім властивості children, куди, власне, передається список нащадків, однією з основних характеристик віджета Column є mainAxisAlignment – властивість, яка описує розташування елементів у колоні відносно головної осі (яка проходить вертикально). Елементи можуть розташовуватись послідовно з початку колони, з

простором між елементами, простором навколо елементів, по центру відносно головної осі тощо. Подібним до Column є Row – віджет, що розташовує список нащадків у вигляді ряду. Він має той самий набір властивостей, що і Column з єдиною відмінністю – у Row головна вісь проходить горизонтально.

Першим елементом у колоні в даному прикладі є Container – один з найбільш часто використовуваних віджетів у Flutter. Він поєднує в собі такі аспекти дизайну як розташування, стилістика кольорів, розмір та форма. У прикладі на рисунку 2.1 задано висоту контейнера (властивість height), тип розташування нащадка (alignment) та стилістику самого контейнера (decoration). Значенням параметру decoration передається об'єкт BoxDecoration, в якому можна задати різні стилістичні ознаки контейнера, наприклад, колір, форма, рамки тощо. Крім цих властивостей контейнеру також можна задавати ширину, внутрішні за зовнішні відступи, обмеження розмірів тощо.

Нащадком вище згаданого контейнера в даному прикладі є Text – простий віджет для відображення текстових даних. Text – це показовий приклад того, що все у Flutter є віджетом – ми не можемо передати нащадком контейнера звичайний рядок, необхідно помістити цей рядок як значення відповідного параметру у віджеті Text. У Text можна задавати різні параметри відображення тексту: вирівнювання тексту, напрямок, максимальна кількість рядків тощо. Важливим атрибутом є параметр style – властивість, що приймає клас TextStyle, у якому задають стилістичні ознаки тексту: колір, розмір та стиль шрифту, відстань між символами та словами тощо. Ці параметри можна задати і для всього додатку, визначивши відповідні атрибути у властивості theme в класі MaterialApp. Проте задання стилів у конкретному віджеті Text теж буває корисним – наприклад, коли потрібно відобразити частину текстових даних курсивом, а в решті застосунку залишити звичайний шрифт. Таким чином, властивість style віджета Text перевизначає параметри тексту, задані для всієї аплікації.

TextField – це віджет, що позначає текстове поле вводу. При натисканні на цей віджет відкривається клавіатура пристрою, через яку користувач і вводить текст. Текстовими даними, що відображаються в полі, оперує спеціальний об'єкт

TextEditingController, який передається у властивіть controller. Через цей об’єкт можна отримувати та змінювати дані в текстовому полі.

У TextField, як і в інших елементів вводу (наприклад, кнопки) є властивості, в яких описують реакцію програми на певну взаємодію з полем, так звані колбеки. Наприклад, параметр onChanged визначає поведінку додатка при будь-яких змінах у цьому текстовому полі. У випадку, коли користувач завершив ввід інформації і підтверджує це натиском відповідної клавіші на клавіатурі, викликається колбек onSubmitted. Параметр onTap визначає колбек, що буде викликатись при натисканні на поле.

Віджетам TextField можна також задавати стилі тексту, курсору і декорацію самого поля. У декорації можна визначати текст і стиль підказки, заголовка та повідомлення про помилку вводу. Як і у випадку з Text, стилі та декорації, задані в конкретному віджеті будуть перевизначати ті, що були вказані на загальному рівні застосунку. Також часто використовуваними властивостями TextField є maxLines (максимальна кількість рядків), autocorrect (визначає використання автовиправлення тексту), maxLength (максимальна кількість символів у полі).

2.3 Обробка стану. StatefulWidget та setState

Не всі додатки вимагають використання архітектурних патернів. Якщо структура проекту досить проста, то використання навіть таких легких шаблонів як BLoC чи Cubit може бути нераціональним та збитковим, осільки вибудовування архітектури вимагає чимало часу. Звісно, на великих проектах є сенс витрачати більше часу для підтримки архітектури за обраним патерном, тоді як в невеликих проектах в цьому просто немає сенсу.

Проте практично всі додатки надають користувачеві опції взаємодії. Під час будь-якої взаємодії користувача з програмою змінюється стан програми, який потрібно обробляти. Найпростіший спосіб обробки стану у Flutter – це використання Stateful-віджетів, які містять набір методів для роботи зі станом віджета.

Для того, щоб віджет був stateful, необхідно при оголошенні зробити його похідним від абстрактного класу StatefulWidget, який є базовим для всіх stateful віджетів.

При створенні Stateful-віджетів утворюється також об’єкт стану віджета, який власне і зберігає стан, набір методів, які ним маніпулюють та набір властивостей, які його характеризують. Одним з методів маніпуляції зі станом є метод setState, який використовується для зміни стану. Метод setState в свою чергу викликає метод build, який перебудовує віджет. Таким чином, метод build Stateful-віджетів може викликатись безліч разів у ході роботи аплікації, кожного разу повертаючи оновлене дерево віджетів.

Двома іншими важливими методами StatefulWidget є initState та dispose. initState викликається один раз при створенні віджета. Тому в цьому методі зазвичай проводиться ініціалізація потрібних об’єктів всередині класу. Метод dispose викликається, коли віджет викидається з дерева віджетів. У цьому методі зазвичай проводиться звільнення ресурсів, які більше не використовуються, для уникнення засмічення пам’яті.

2.4 TensorFlow

TensorFlow – це платформа для машинного навчання, розроблена компанією Google [21]. Початково платформу використовували лише всередині Google, але у листопаді 2015 року компанія відкрила вихідний код системи [22]. Моделі TensorFlow можуть працювати на операційних системах Windows, macOS, Linux, iOS, Android та веб-додатках. Для використання на мобільних платформах iOS та Android використовується TensorFlow Lite API [23]. Для роботи з TensorFlow Lite у Flutter є бібліотека tflite [24].

2.5 Інструменти розробника

Практично всі SDK надають інструменти, які полегшують розробку, процес знаходження помилок тощо.

Однією з особливостей Flutter є hot reload – механізм, що дозволяє оновити віджети за долі секунди, при цьому не перезапускаючи застосунок. Також при виклику hot reload не відбувається зміни стану, тобто стек екранів та дані залишаються незмінними. Це не лише значно пришвидшує розробку, але й надає можливість швидко та ефективно експериментувати з графічним інтерфейсом аплікації.

Є кілька винятків, при яких hot reload використовувати не можна і необхідно перезапускати весь додаток:

а) при помилках компіляції;

б) при зміні шрифтів;

в) при зміні нативного коду в конфігураційних файлах Android або iOS;

г) інші, менш поширені випадки.

Іншим корисним інструментом є Flutter doctor – інструмент, який відображає повідомлення з інформацією, попередженнями та помилками, що стосуються Flutter SDK. Наприклад, це інформація про версію SDK чи список доступних пристроїв. Викликається за допомогою команди flutter doctor у терміналі або через відповідний елемент на панелі інструментів у середовищі розробки.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ ВИЯВЛЕННЯ ДЕМЕНЦІЇ З ВИКОРИСТАННЯМ FLUTTER ТА ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Розглянемо програмну реалізацію додатку, основною функцією якого є надання користувачу можливості пройти тест на деменцію. Застосунок написаний на Flutter з використанням моделі TensorFlow Lite.

Для виконання завдання було використано ноутбук Apple MacBook Air M1. Перевагою цього девайсу є можливість тестування на платформі iOS, оскільки це можна здійснити лише на пристроях з операційною системою macOS.

Насамперед необхідно згенерувати модель tflite, яка буде розпізнавати зображення та визначати наскільки коректно намальовані стрілки годинника. Для цього було використано портал Teachable Machine [25]. Для навчання моделі було завантажено вибірки з порталу Kaggle, частину навчальних прикладів було створено самостійно. Після створення нового Image Project необхідно помістити вибірки зображень у кожен лейбл окремо (рисунок 3.1). У цьому випадку було створено 13 класів. 12 з них – це кожна ціла година. Інший (лейбл «0») – клас із очевидно неправильними малюнками. У ході дослідження було виявлено, що додавання нульового класу покращує точність результатів нейронної мережі в цій задачі. Після натиску на кнопку Train Model та тренування моделі її можна завантажити по натиску на кнопку Export Model.

Корисною опцією порталу Teachable Machine є можливість збереження проекту на Google Drive. Завдяки цьому, при необхідності внесення корективів у навчання моделі, не потрібно заново створювати проект. Особливо зручною така опція є при великій кількості класів. У випадку цього дослідження, процес навчання моделі редагувався двічі: при додаванні нових навчальних прикладів після першого тестування моделі, яке видало не найкращі результати та після додавання класу з очевидно неправильними малюнками, що теж значно покращило роботу моделі.

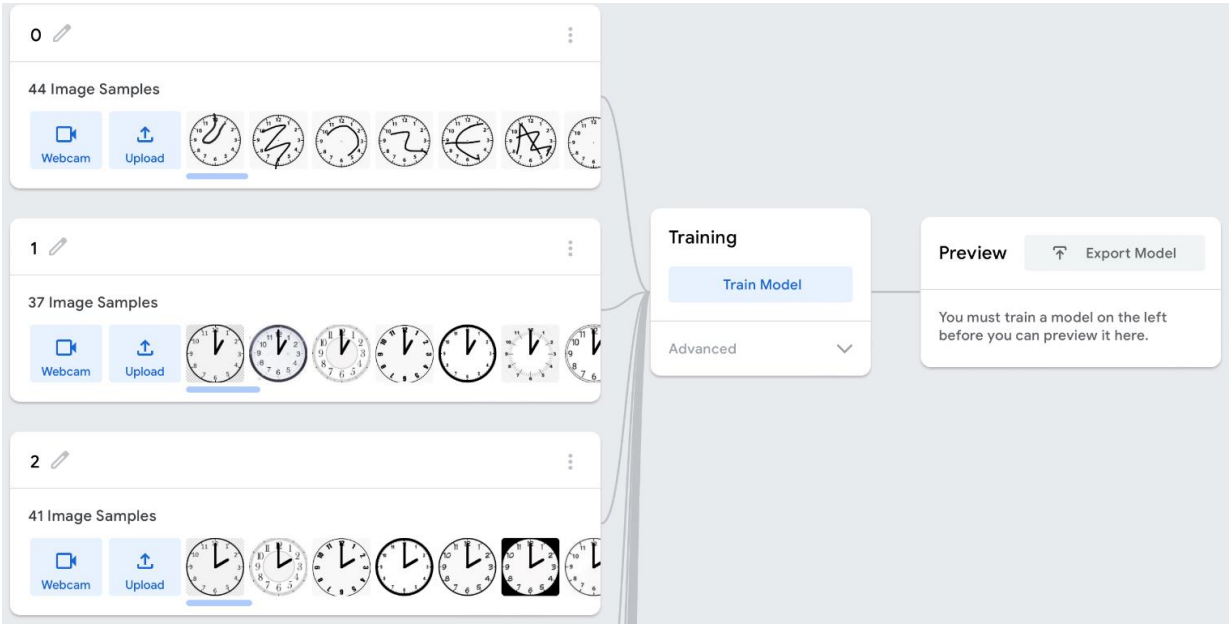

Рисунок 3.1 – Фрагмент створених класів на порталі Teachable Machine

Для роботи з Flutter з офіційного сайту було завантажено zip-архів з Flutter SDK версії 3.3.7. У якості середовища розробки було обрано IDE Visual Studio Code від Microsoft. Після розархівування SDK та додавання середовища Flutter до змінної PATH було викликано команду «flutter doctor», результат якої зображено на рисунку 3.2.

```
vpanyush@Perpetios-MacBook-Air-3            % flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.3.7, on macOS 13.0.1 22A400 darwin-arm, locale en-UA)
[!] Android toolchain - develop for Android devices (Android SDK version 32.1.0-rc1)
    ✗ cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    ✗ Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/docs/get-started/install/macos#android-setup for more details.
[✓] Xcode - develop for iOS and macOS (Xcode 14.0.1)
[✗] Chrome - develop for the web (Cannot find Chrome executable at /Applications/Google Chrome.app/Contents/MacOS/Google
    Chrome)
    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[✓] Android Studio (version 2021.2)
[✓] VS Code (version 1.65.2)
[✓] Connected device (1 available)
[✓] HTTP Host Availability
```

Рисунок 3.2 – Виконана команда «flutter doctor»

Переконавшись за допомогою команди «flutter doctor», що з основними елементами розробки (Flutter, Xcode, VS Code) проблем немає, було створено новий Flutter-проект. Після створення Flutter-проекту у файлі pubspec.yaml необхідно вказати залежності, які будуть використовуватись у додатку. Також в розділі assets необхідно вказати місце розташування tflite-моделі та текстового файлу з лейблами, в даному випадку це assets/network/. Розділи файлу pubspec.yaml із залежностями та assets зображені на рисунку 3.3.

```
  cupertino_icons: ^1.0.2
  flutter_state_notifier: ^0.7.1
  tflite: ^1.1.2
  painter: ^2.0.0
  screenshot: ^1.2.3
  path_provider: ^2.0.11

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^2.0.0

flutter:
  uses-material-design: true

  assets:
    - assets/
    - assets/network/
```

Рисунок 3.3 – Фрагмент файлу pubspec.yaml

Для збереження чистоти структури проекту, весь код було розподілено між різними файлами, а файли – між папками (рисунок 3.4). Таким чином, код різного контексту та призначення знаходиться в різному місці. Це спрощує сприйняття проекту як самим розробником, так і іншими людьми, в яких з'явилась потреба вивчати цей проект. Також такий розподіл дозволяє легко додавати нові складові проекту.

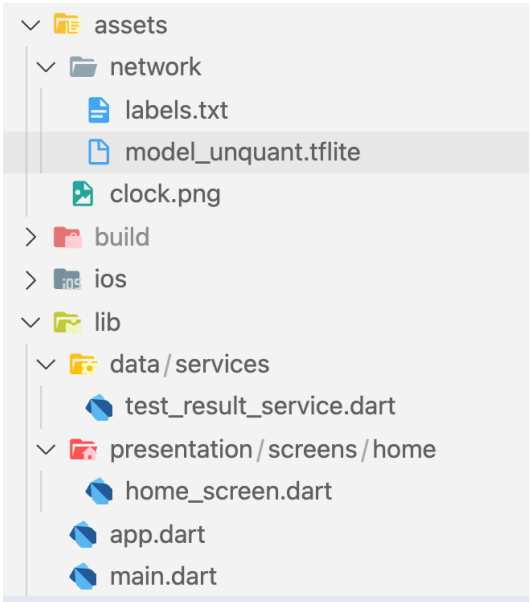

Рисунок 3.4 – Фрагмент структури проекту

Точкою входу в програму, як і в більшості сучасних мов програмування, є функція main, що знаходиться у файлі main.dart (рисунок 3.5). У ній викликається функція runApp, в яку параметром передається клас-аплікація DDD.

```dart
import 'package:flutter/material.dart';

import 'app.dart';

Run | Debug | Profile
void main() {
  runApp(const DDD());
}
```

Рисунок 3.5 – Файл main.dart

Клас DDD знаходиться у файлі app.dart (рисунок 3.6). Цей клас є Stateless-віджетом, який у методі build повертає віджет MaterialApp, головною сторінкою якого є клас HomeScreen.

```dart
import 'package:flutter/material.dart';

import 'presentation/screens/home/home_screen.dart';

class DDD extends StatelessWidget {
  const DDD({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(primarySwatch: Colors.teal),
      debugShowCheckedModeBanner: false,
      home: const HomeScreen(),
    ); // MaterialApp
  }
}
```

Рисунок 3.6 – Файл app.dart

Двома основними класами додатку є TestResultService та HomeScreen. Розглянемо детальніше кожен із них.

TestResultService – це клас, основною функцією якого є обчислення правильності намальованого годинника. Для завантаження моделі нейронної мережі викликається метод loadModel класу Tflite з бібліотеки tflite, у який параметрами передається шлях до самої моделі та до текстового файлу з лейблами labels.txt (рисунок 3.7), який теж генерується при експорті моделі з порталу Teachable Machine. Інкапсульований метод loadModel зображений на рисунку 3.8.

assets > network > labels.txt

```
0
1
2
3
4
5
6
7
8
9
10
11
12
```

Рисунок 3.7 – Файл labels.txt

```
Future<void> loadModel() async {
  await Tflite.loadModel(
    model: 'assets/network/model_unquant.tflite',
    labels: 'assets/network/labels.txt',
  );
}
```

Рисунок 3.8 – Метод loadModel класу TestResultService

Основним методом цього класу є getTestResult (рисунок 3.9). У ньому викликається метод runModelOnImage класу Tflite, куди параметрами передається шлях до зображення годинника, яке потрібно класифікувати. runModelOnImage повертає список об’єктів, кожен з яких складається з трьох полів – index, label та confidence, які позначають індекс, лейбл класу та ймовірність приналежності зображення до цього класу відповідно. Далі розглядається об’єкт з найбільшою впевненістю. Якщо лейбл не збігається з очікуваним часом або лейбл є «0», то правильність намальованого годинника дорівнює нулю. В іншому випадку повертається впевненість помножена на 100 (для отримання результату у відсотках).

Трьома основними параметрами методу runModelOnImage є path, numResults та threshold. Аргумент path відповідає за шлях до зображення, яке потрібно класифікувати. Оскільки цей шлях щоразу буде різним, то значення передається з параметрів методу getTestResult. Параметр numResults позначає кількість можливих класів, до одного з яких буде віднесено зображення. Параметр threshold означає поріг впевненості для результатів моделі.

```
Future<String> getTestResult({
  required String imagePath,
  required int time,
}) async {
  final output = await Tflite.runModelOnImage(
    path: imagePath,
    numResults: 13,
    threshold: 0.1,
    imageMean: 127.5,
    imageStd: 127.5,
  );
  if (output != null) {
    final first = output.first;
    if (first['label'] != '$time' || first['label'] == '0') return '0';
    return (first['confidence'] * 100).toStringAsFixed(2);
  }
  return '0';
}
```

Рисунок 3.9 – Метод getTestResult класу TestResultService

Ще одним методом класу TestResultService є dispose – він звільняє ресурси, зайняті моделлю за допомогою виклику асинхронного методу close класу Tflite (рисунок 3.10).

```
Future<void> dispose() async {
  await Tflite.close();
}
```

Рисунок 3.10 – Метод dispose класу TestResultService

Варто зазначити, що всі методи класу TestResultService є асинхронними, тобто такими, які без спеціальних інструкцій виконуються паралельно з основним потоком. Це пов’язано з тим, що робота цих методів є доволі громіздкою і часовитратною, тому з метою уникнення блокування додатку під час роботи цих методів вони були створені як асинхронні.

Другим основним класом цього додатку є HomeScreen. Це stateful-віджет, який відповідає за роботу головного екрану застосунку.

```
import 'dart:io';
import 'dart:math';
import 'dart:typed_data';

import 'package:dementia_test/data/services/test_result_service.dart';
import 'package:flutter/material.dart';
import 'package:painter/painter.dart';
import 'package:path_provider/path_provider.dart';
import 'package:screenshot/screenshot.dart';
```

Рисунок 3.11 – Імпорт бібліотек для класу HomeScreen

На рисунку 3.11 зображено імпорт бібліотек, які будуть потрібними для реалізації функціоналу класу HomeScreen. Детальніше використання окремих бібліотек буде розглянуто нижче.

У методі initState (рисунок 3.12) проводиться ініціалізація об’єкту TestResultService та двох контролерів – _screenshotController класу ScreenshotController (з бібліотеки screenshot [26]) та _painterController класу PainterController (з бібліотеки painter [27]). _screenshotController відповідає за знімки віджета, _painterController – за можливості малювання на віджеті Painter. Також в initState відбувається завантаження моделі tflite.

```
@override
void initState() {
  _resultService = TestResultService();
  _screenshotController = ScreenshotController();
  _painterController = PainterController();
  _painterController.thickness = 10;
  _painterController.backgroundColor = Colors.transparent;
  WidgetsBinding.instance.addPostFrameCallback((_) async {
    await _resultService.loadModel();
  });
  super.initState();
}
```

Рисунок 3.12 – Метод initState класу _HomeScreenState

Трьома основними елементами екрану є завдання, годинник і кнопки для отримання нового завдання та підтвердження результату.

Метод _task (рисунок 3.13) повертає віджет з текстом завдання. Текст буде різним в залежності від змінної _time, яка позначає очікуваний намальований час. Новий час генерується в методі _newTime (рисунок 3.14) за допомогою класу Random. Для генерації чисел від 1 до 12 було використано метод random.nextInt, в якому максимальним значенням (яке є невключним) є число 12, з додаванням одиниці. Для того, щоб зміна стану відобразилась на екрані, виконується виклик методу setState.

```
Widget _task() => SizedBox(
      height: 20,
      child: _time != null
          ? Text(
              "Please, draw $_time o'clock on the clock",
              style: _textStyle().copyWith(fontWeight: FontWeight.w700),
            ) // Text
          : null,
    ); // SizedBox
```

Рисунок 3.13 – Метод _task класу _HomeScreenState

```
void _newTime() {
  _painterController.clear();
  final Random random = Random();
  final newTime = random.nextInt(12) + 1;
  setState(() {
    _time = newTime;
  });
}
```

Рисунок 3.14 – Метод _newTime класу _HomeScreenState

Годинник будується на екрані за допомогою методу _clock (рисунок 3.15). Віджет Painter дозволяє користувачу малювати в цій зоні з параметрами малювання, які були задані в _painterController. Оскільки клас Painter не передбачає зміни зображення на фоні, то зображення циферблату годинника було поміщено в нижній шар віджету Stack. Stack в свою чергу обгорнутий у віджет Screenshot, який дозволяє робити знімок всієї зони. Саме цей знімок потім буде використаний у класифікації зображення.

```
Widget _clock() => SizedBox(
      height: 400,
      child: Screenshot(
        controller: _screenshotController,
        child: Stack(
          children: [
            Center(child: Image.asset('assets/clock.png')),
            Painter(_painterController),
          ],
        ), // Stack
      ), // Screenshot
    ); // SizedBox
```

Рисунок 3.15 – Метод _clock класу _HomeScreenState

При натискання на кнопку «Submit» викликається метод _submit (рисунок 3.16). У ньому робиться знімок зони годинника за допомогою методу _screenshotController.capture. Це зображення зберігається в тимчасовій пам’яті та передається в метод _resultService.getTestResult разом зі змінною _time для визначення результату тесту. Отриманий результат відображається у діалоговому вікні, яке викликається за допомогою методу _showResultDialog.

Метод _showResultDialog зображений на рисунку 3.17. У ньому виконується виклик методу showDialog з бібліотеки material. Параметром _showResultDialog є correctPercentage, який визначає впевненість моделі при класифікації. Значеннння цього параметру зображається у розділі content.

```
void _submit() async {
  Uint8List? imageUint = await _screenshotController.capture();
  if (imageUint != null) {
    final tempDir = await getTemporaryDirectory();
    File file = await File('${tempDir.path}/image.png').create();
    await file.writeAsBytes(imageUint);
    final correctPercentage = await _resultService.getTestResult(
      imagePath: file.path,
      time: _time!,
    );
    await _showResultDialog(correctPercentage);
  }
}
```

Рисунок 3.16 – Метод _submit класу _HomeScreenState

```
Future<void> _showResultDialog(String correctPercentage) async =>
    showDialog<void>(
      context: context,
      builder: (BuildContext context) => AlertDialog(
        title: const Text('Test result'),
        content: Text('Your answer is correct for $correctPercentage %'),
        actions: <Widget>[
          TextButton(
            onPressed: Navigator.of(context).pop,
            child: const Text('Ok'),
          ), // TextButton
        ], // <Widget>[]
      ), // AlertDialog
    );
```

Рисунок 3.17 – Метод _showResultDialog класу _HomeScreenState

Результати роботи програми зображені на рисунках 3.18 – 3.21.

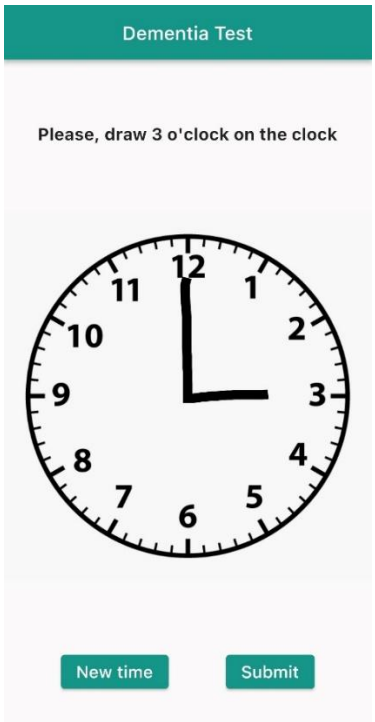

Рисунок 3.18 – Грамотно намальований очікуваний час

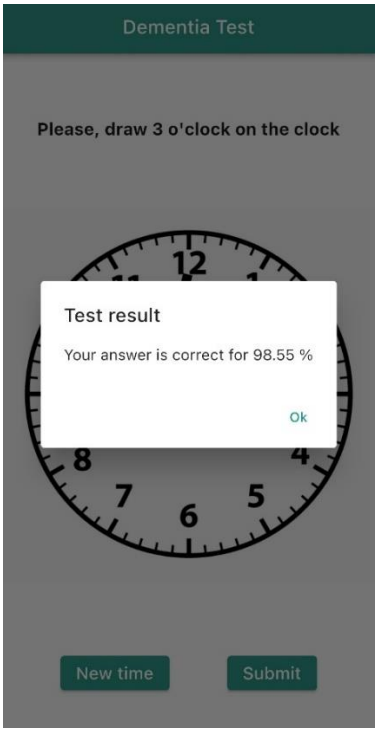

Рисунок 3.19 – Хороший результат тесту

Рисунок 3.20 – Очевидно некоректно намальований час

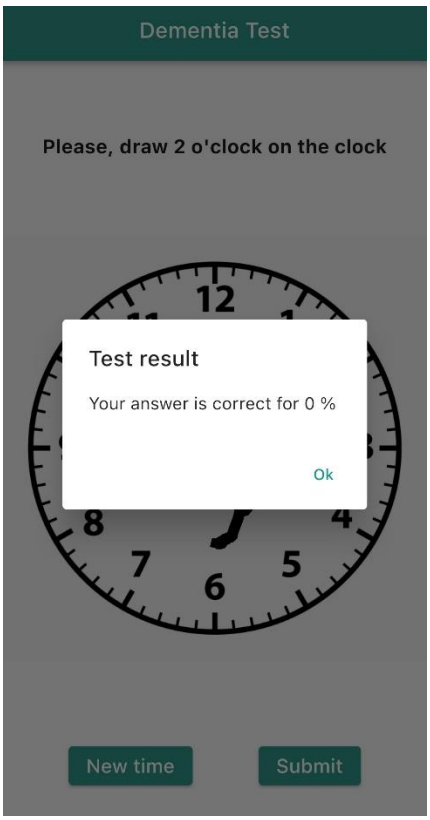

Рисунок 3.21 – Поганий результат тесту

Як бачимо з наведених рисунків, коректність роботи додатку є задовільною. При малюванні заданої години грамотно і акуратно система видає високе значення впевненості у тому, що результат є позитивним. Натомість при малюванні очевидно некоретного зображення стрілок годинника модель видає результат впевненості 0%.

Отже, дану імплементацію тесту з годинником з допомогою фреймворків Flutter та TensorFlow Lite можна вважати успішно виконаною.

ВИСНОВКИ

У даній роботі було розглянуто особливості та основні етапи в роботі з Flutter – фреймворку для кросплатформної мобільної розробки від компанії Google. Були розглянуті основні елементи в архітектурі самого фреймворку, описані найпоширеніші графічні компоненти – віджети. Також було розглянуто принцип роботи штучної нейронної мережі. Крім цього, детально був описаний спосіб обробки стану застосунку за допомогою можливостей віджета StatefulWidget. Також було розглянуто сервіс для машинного навчання TensorFlow.

На основі проведених досліджень було розроблено додаток, основним функціоналом якого є тест на виявлення деменції. Застосунок надає можливість користувачу малювати стрілки на схематичному зображенні аналового годинника. Отримане зображення передається в нейронну мережу, яка виконує класифікацію того, наскільки правильним є зображення. Отриманий результат виводиться у діалоговому вікні.

Перед безпосередньою програмною реалізацією було завантажено та налаштовано середовище розробки Visual Studio Code. У ньому було встановлено Flutter-плагін.

Нейронна мережа, яка використовується в додатку для класифікації зображення, була навчена на порталі Teachable Machine та інтегрована в аплікацію у вигляді моделі TensorFlow Lite за допомогою бібліотеки tflite. Більшість навчальних прикладів у процесі створення нейронної мережі було завантажено з порталу Kaggle, частину було створено самостійно.

Додаток виправдав очікування щодо наявності потрібного функціоналу та коректності роботи, тому можна вважати, що поставленої мети перед виконанням роботи було досягнуто.

Цей застосунок може бути використаний у медицині, зокрема, у напрямку когнітивних порушень, таких як деменція. Додаток може використовуватись як самостійний окремий інструмент для виявлення деменції, так і як інтегрований модуль у будь-який іншим додаток, написаний на Flutter. Особливо актуальною

буде така інтеграція в додатках, цільовою аудиторією яких є люди похилого віку та люди з потенційними ураженнями когнітивних функцій.

Суть та результати цієї роботи можуть бути цікавими не лише для спеціалістів у сфері комп’ютерних технологій та штучних нейронних мереж, але і для спеціалістів у галузі медицини, особливо в напрямку дослідження деменції.

Отже, у ході написання даної роботи було виконано усі поставлені завдання, а саме: розгляд необхідних інструментів для реалізації додатку на Flutter, налаштування середовища для розробки, підготовка навчальних прикладів та навчання нейроннлї мережі класифікувати зображення аналового годинника, реалізація додатку з тестом на деменцію на Flutter.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google’s Dart language on Android aims for Java-free, 120 FPS apps | Ars Technica : [Electronic resource] – Available from : https://arstechnica.com/gadgets/2015/05/googles-dart-language-on-android-aims-for-java-free-120-fps-apps/
2. Flutter Live : [Electronic resource] – Available from : https://developers.google.com/events/flutter-live
3. flutter/flutter: Flutter makes it easy and fast to build beautiful apps for mobile and beyond. : [Electronic resource] – Available from : https://github.com/flutter/flutter
4. Showcase – Flutter : [Electronic resource] – Available from : https://flutter.dev/showcase
5. Flutter Showcase | Google Pay : [Electronic resource] – Available from : https://flutter.dev/showcase/google-pay
6. Flutter SDK releases | Flutter : [Electronic resource] – Available from : https://docs.flutter.dev/development/tools/sdk/releases
7. Dart packages : [Electronic resource] – Available from : https://pub.dev/
8. Flutter architectural overview | Flutter : [Electronic resource] – Available from : https://docs.flutter.dev/resources/architectural-overview
9. Dart: a language for structured web programming : [Electronic resource] – Available from : http://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html
10. Dart for the Entire Web : [Electronic resource] – Available from : https://news.dartlang.org/2015/03/dart-for-entire-web.html
11. sdk/CHANGELOG.md at stable · dart-lang/sdk : [Electronic resource] – Available from : https://github.com/dart-lang/sdk/blob/stable/CHANGELOG.md
12. Dart overview | Dart : [Electronic resource] – Available from : https://dart.dev/overview
13. Sound null safety | Dart : [Electronic resource] – Available from : https://dart.dev/null-safety

14. foundation library - Dart API : [Electronic resource] – Available from : https://api.flutter.dev/flutter/foundation/foundation-library.html
15. File:MultiLayerPerceptron.png - Wikimedia Commons : [Electronic resource] – Available from : https://commons.wikimedia.org/w/index.php?curid=19106639
16. Dementia : [Electronic resource] – Available from : https://www.who.int/news-room/fact-sheets/detail/dementia
17. How the Clock-Drawing Test Screens for Dementia : [Electronic resource] – Available from : https://www.verywellhealth.com/the-clock-drawing-test-98619
18. Examples of clock drawings produced by people diagnosed with dementia. | Download Scientific Diagram : [Electronic resource] – Available from : https://www.researchgate.net/figure/Examples-of-clock-drawings-produced-by-people-diagnosed-with-dementia_fig4_282815058
19. Install - Flutter : [Electronic resource] – Available from : https://flutter.dev/docs/get-started/install
20. Set up an editor – Flutter : [Electronic resource] – Available from : https://flutter.dev/docs/get-started/editor?tab=androidstudio
21. TensorFlow: [Electronic resource] – Available from : https://www.tensorflow.org/
22. Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine | WIRED : [Electronic resource] – Available from : https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/
23. TensorFlow Lite API Reference : [Electronic resource] – Available from : https://www.tensorflow.org/lite/api_docs
24. tflite | Flutter Package : [Electronic resource] – Available from : https://pub.dev/packages/tflite
25. Teachable Machine : [Electronic resource] – Available from : https://teachablemachine.withgoogle.com/
26. screenshot | Flutter Package : [Electronic resource] – Available from : https://pub.dev/packages/screenshot
27. painter | Flutter Package : [Electronic resource] – Available from : https://pub.dev/packages/painter