ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Android-застосунку для автоматизації перевірки тестувальних бланків у навчальних процесах»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій КОЛОСОВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Андрій КОЛОСОВСЬКИЙ
група КНД-41

Керівник: д. т. н., професор Ільїн Олег Олександрович
науковий ступінь, вчене звання

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Колосовський Андрій Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка Android-застосунку для автоматизації перевірки тестувальних бланків у навчальних процесах

керівник кваліфікаційної роботи Олег ІЛЬЇН д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. №36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи комп'ютерного зору, вимоги до мобільних застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області
Розглянути базові алгоритми обробки та аналізу зображень
Вибір технологій та інструментів для розробки програмного рішення

5. Перелік графічного матеріалу: *презентація*
    1. Опис проблеми та її вирішення
    2. Вступ до комп'ютерного зору
    3. Опис програмного рішення

6. Дата видачі завдання«23» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз поставленої задачі	23.02-01.03.24	
2	Підбір науково-технічної літератури	02.03-15.03.24	
3	Аналіз алгоритмів комп'ютерного зору	16.03-31.03.24	
4	Проектування програмного застосунку	01.04-07.04.24	
5	Розробка програмного застосунку	08.04-30.04.24	
6	Написання вступу, висновків, реферату	01.05-05.05.24	
7	Написання основних розділів пояснюювальної записки	06.05-30.05.24	
8	Подання роботи в деканат	31.05.24	

Здобувач вищої освіти _____ Андрій КОЛОСОВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина бакалаврської роботи: 48 с., 2 рис., 9 джерел.

Наукове завдання – дослідження і розробка Android-застосунку для автоматизації перевірки тестувальних бланків у навчальних процесах. Мета роботи – вирішення проблем перевірки тестувальних бланків механичнім шляхом.

Об’єкт дослідження – аналіз алгоритмів обробки зображень.

Предмет дослідження – технології комп’ютерного зору для вирішення задач перевірки тестових бланків.

Короткий зміст роботи: Проведено аналіз алгоритмів комп’ютерного зору. Проаналізовано існуючі рішення проблеми. Розглянуто технології, які були використані при розробці програмного рішення. Розроблено мобільний застосунок для вирішення проблемі за допомогою бібліотеки OpenCV та мови програмування Kotlin. Проведено тести програмного рішення. Застосування розробленого рішення допомагає заощаджувати час при перевірці тестових бланків у навчальних процесах.

КЛЮЧОВІ СЛОВА: КОМП’ЮТЕРНИЙ ЗІР, МОБІЛЬНИЙ ЗАСТОСУНОК, ОПТИЧНЕ РОЗПІЗНАВАН, KOTLIN, ANDROID

ЗМІСТ

ВСТУП

Актуальність дослідження

Останніми роками технології комп'ютерного зору та машинного навчання знаходять дедалі ширше застосування у різних сферах життя, від медицини до транспортних засобів. Одним із перспективних напрямків використання цих технологій є автоматизація процесів оцінювання та перевірки тестових завдань. У навчальних закладах та на професійних іспитах перевірка тестів традиційно займає багато часу і вимагає значних людських ресурсів. Автоматизація цього процесу дозволяє суттєво скоротити час на перевірку, мінімізувати людський фактор та підвищити точність оцінювання.

Використання мобільних додатків для автоматичної перевірки тестів особливо актуальне в умовах зростаючої доступності смартфонів та планшетів. Ці пристрої можуть бути легко інтегровані у навчальний процес, забезпечуючи вчителів та екзаменаторів потужними інструментами для швидкої та точної перевірки результатів.

Мета і завдання роботи

Метою даної роботи є розробка мобільного додатку на платформі Android з використанням мови програмування Kotlin та бібліотеки OpenCV, здатного автоматично розпізнавати та порівнювати заповнені тестові бланки з еталонними відповідями, оцінюючи результати за 100-бальною шкалою.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Вивчити та проаналізувати існуючі методи і алгоритми комп'ютерного зору, що застосовуються для розпізнавання та обробки зображень.
2. Розробити архітектуру мобільного додатку, що включає всі необхідні компоненти для обробки зображень, розпізнавання зафарбованих кружечків та порівняння їх з еталонними відповідями.
3. Реалізувати додаток мовою Kotlin з використанням бібліотеки OpenCV, забезпечивши його коректну роботу на пристроях під керуванням Android.

4. Провести тестування та оцінку ефективності роботи додатку на різних тестових бланках, аналізуючи точність розпізнавання та порівняння.

Структура роботи

Дипломна робота складається з декількох розділів, кожен з яких присвячений певному аспекту розробки мобільного додатку для автоматичної перевірки тестів.

У першому розділі розглядаються теоретичні основи автоматичної перевірки тестів, включаючи методи та алгоритми комп'ютерного зору.

Другий розділ присвячений аналізу існуючих аналогів роботи.

У третьому розділі описується процес проектування мобільного додатку, включаючи визначення вимог, архітектуру та користувацький інтерфейс.

Четвертий розділ присвячений реалізації додатку мовою Kotlin з використанням OpenCV, включаючи інтеграцію бібліотеки та реалізацію основних функцій.

У п’ятому розділі проводиться тестування та оцінка ефективності додатку, аналізуються результати та робляться висновки про його точність і продуктивність.

Висновок містить узагальнення отриманих результатів та рекомендації щодо подальшого розвитку і вдосконалення додатку.

1 ТЕОРИТИЧНІ ОСНОВИ АВТОМАТИЗОВАНОЇ ПЕРЕВІРКІ ТЕСТІВ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Методи та алгоритми комп'ютерного зору

1.1.1 Вступ до комп'ютерного зору

Комп'ютерний зір – це галузь штучного інтелекту, яка займається автоматичним отриманням, обробкою та аналізом візуальної інформації з реального світу. Основною метою комп'ютерного зору є створення систем, які можуть розпізнавати та розуміти візуальні об'єкти так само, як це робить людське око. Комп'ютерний зір знаходить застосування в різних сферах, таких як медицина, робототехніка, автомобільна промисловість, безпека, а також в аналізі зображень та відео.

1.1.2 Загальний огляд алгоритмів розпізнавання об'єктів

Розпізнавання об'єктів є однією з ключових задач комп'ютерного зору. Існує багато різних методів та алгоритмів, які використовуються для розпізнавання об'єктів на зображеннях. Вони можуть бути умовно поділені на кілька груп:

- Методи, засновані на ознаках (feature-based methods): ці методи використовують різні характеристики зображень, такі як контури, кути, текстури та інші локальні ознаки. Серед популярних методів цієї групи можна виділити SIFT (Scale-Invariant Feature Transform), SURF (Speeded Up Robust Features) та ORB (Oriented FAST and Rotated BRIEF). Вони дозволяють знаходити унікальні точки на зображеннях, які можуть бути використані для порівняння та розпізнавання об'єктів.
- Методи, засновані на шаблонах (template matching): ці методи включають порівняння зображення або його частини з попередньо визначеним шаблоном. Хоча вони є менш гнучкими до змін масштабу, обертання та інших трансформацій об'єкта, їх простота та ефективність роблять їх корисними для певних задач.

- Методи машинного навчання: ці методи використовують алгоритми машинного навчання для навчання моделей на основі великих наборів даних. Вони можуть бути використані для класифікації об'єктів, виявлення облич, розпізнавання рукописного тексту та інших задач. Зокрема, методи машинного навчання можуть включати використання алгоритмів класифікації, таких як метод опорних векторів (SVM), дерева рішень, випадкові ліси та інші.
- Методи глибокого навчання: ці методи базуються на нейронних мережах, зокрема на згорткових нейронних мережах (CNN). Завдяки своїй високій точності та здатності обробляти великі обсяги даних, вони стали дуже популярними в останні роки. CNN використовуються для різних задач, включаючи класифікацію зображень, сегментацію об'єктів та розпізнавання обличь.

1.1.3 Фільтрація зображень

Фільтрація зображень – це процес, що включає зміну значень пікселів зображення для зменшення шуму, згладжування або виділення важливих характеристик, таких як краї або контури.

Gaussian Blur – це метод згладжування зображення за допомогою гаусового фільтра. Він зменшує шум і деталі, що робить зображення більш гладким. Гаусовий фільтр використовує нормальний розподіл для обчислення ваги пікселів навколо центрального пікселя.

Дискретний 2D гаусовий фільтр (ядро):

Зазвичай фільтр дискретизується для застосування до зображень. Наприклад, для $\sigma = 1$

$$\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Ця матриця представляє собою ядро 2D гаусового фільтра, яке використовується в обробці зображень для згладжування зображення. Основна мета такого фільтра — зменшення шуму та деталей на зображенні, що досягається шляхом згладжування пікселів.

Як це працює:

- **Пікселі**: Зображення складається з пікселів, кожен з яких має своє значення інтенсивності (для сірого зображення) або значення для кожного кольору (для кольорового зображення).
- **Згладжування**: Гаусовий фільтр використовується для згладжування зображення. Це досягається шляхом заміни значення кожного пікселя середньозваженим значенням його сусідніх пікселів.
- **Ядро фільтра**: Ваша матриця (ядро фільтра) визначає ваги для сусідніх пікселів під час згладжування. Наприклад, центральний елемент (значення 41) має найбільшу вагу, тоді як елементи на периферії мають менші ваги. Це відповідає принципу гаусового розподілу, де центр вагової функції має максимальне значення і зменшується до країв.

Коли цей фільтр застосовується до зображення, кожен піксель замінюється на середньозважене значення пікселів у сусідньому оточенні, зважене згідно з ядром фільтра.

Приклад застосування:

- Центральний піксель: Припустимо, що ви застосовуєте цей фільтр до центрального пікселя зображення.
- Вікно 5x5: Ви берете 5x5 вікно пікселів навколо цього центрального пікселя.
- Зважування: Помножуєте значення кожного пікселя в цьому вікні на відповідний елемент ядра фільтра.
- Сума: Сумуєте ці значення.

- Нормалізація: Ділите суму на загальну суму елементів ядра (273 у вашому випадку).
- Нове значення: Отримане значення встановлюєте як нове значення для центрального пікселя.

Це процес повторюється для кожного пікселя зображення (за винятком крайніх пікселів, де може бути застосоване додаткове оброблення для врахування меж зображення).

1.1.4 Виявлення контурів

Canny Edge Detection є одним з найпопулярніших методів для виявлення контурів на зображеннях. Він був розроблений Джоном Канні в 1986 році і є багатокроковим процесом, що забезпечує точне і стійке виявлення контурів. Алгоритм складається з кількох етапів, які забезпечують ефективне виділення країв об'єктів на зображенні:

- Згладжування зображення (Gaussian Blur): перед обчисленням градієнтів зображення згладжується за допомогою гаусового фільтра, щоб зменшити шум, який може викликати помилкові контури.
- Обчислення градієнтів зображення (Gradient Calculation)
- Після згладжування використовуються оператори Собеля для обчислення горизонтальних та вертикальних градієнтів зображення.
- Придушення немаксимальних значень (Non-Maximum Suppression)
  Цей етап полягає в тому, щоб зберегти лише локальні максимуми градієнтів, які знаходяться вздовж напрямку градієнта, і придушити всі інші значення. Це дозволяє отримати тонкі контури.
  Процедура:
    - Для кожного пікселя обчислити інтерполюючи значення градієнта в напрямку градієнта (θ).

- Якщо градієнт пікселя є локальним максимумом в своєму напрямку, залишити його; інакше придушити.

- Подвійний поріг (Double Thresholding)

  Для розділення сильних і слабких контурів застосовуються два пороги: верхній (Thigh) та нижній (Tlow).

  - Пікселі з градієнтною величиною вище Thigh вважаються сильними краями.
  - Пікселі з градієнтною величиною між Thigh вважаються слабкими краями.
  - Пікселі з градієнтною величиною нижче Tlow відкидаються.

- Гістерезисне відстеження (Hysteresis Thresholding)

  Слабкі краї включаються в контури, якщо вони з’єднані з сильними краями, формуючи завершені контури.

  Процедура:

  - Почати з сильних країв.
  - Якщо слабкий край з’єднаний з сильним, включити його в контур.
  - Повторювати, поки всі пікселі не будуть перевірені.

1.1.5 Використання бібліотеки OpenCV для розпізнавання об'єктів

OpenCV (Open Source Computer Vision Library) – це потужна бібліотека з відкритим вихідним кодом, яка призначена для розробки додатків у сфері комп'ютерного зору та машинного навчання. Вона надає безліч інструментів та функцій для обробки зображень, розпізнавання об'єктів, відстеження руху та інших завдань.

Основні можливості OpenCV:

- Обробка зображень: фільтрація, сегментація, обробка контурів, перетворення простору кольорів. Ці методи дозволяють підготувати зображення для подальшого аналізу та обробки.

- Розпізнавання об'єктів: алгоритми для детекції та розпізнавання облич, об'єктів, жестів. Наприклад, Haar Cascade Classifier, який використовується для детекції облич на зображеннях.
- Калібрування камери: методи для калібрування та вирівнювання зображень, що дозволяють покращити точність розпізнавання об'єктів та вимірювань.
- 3D-реконструкція: відновлення тривимірної форми зображених об'єктів за допомогою стерео зображень або руху камери.

1.1.6 Використання OpenCV для розпізнавання заповнених кружечків на тестових бланках

Для розпізнавання заповнених кружечків на тестових бланках використовуються наступні основні кроки:

Підготовка зображення:

- Перетворення зображення в градації сірого: цей крок дозволяє зменшити кількість інформації, що обробляється, зберігаючи лише інтенсивність пікселів.
- Застосування порогової обробки для виділення контурів: адаптивне порогування дозволяє виділити заповнені кружечки на зображенні, незалежно від умов освітлення.

Виділення контурів та фігур:

- Пошук контурів на зображенні: за допомогою функції findContours в OpenCV можна знайти всі контури на бінарному зображенні.
- Фільтрація контурів за розміром та формою для виявлення кружечків: після знаходження контурів необхідно відфільтрувати їх за формою та розміром, щоб залишити тільки ті, які відповідають заповненим кружечкам.

Аналіз заповнених кружечків:

- Обчислення площі заповнення кожного кружечка: для кожного знайденого контуру обчислюється площа заповнення.

- Визначення, чи є кружечок заповненим на основі порогового значення: якщо площа заповнення перевищує заданий поріг, кружечок вважається заповненим.

Код програми на Kotlin з використанням OpenCV, представлений у додатку, демонструє ці кроки та показує, як вони можуть бути реалізовані на практиці. В наступних розділах роботи детально описується кожен етап розробки та реалізації додатку для автоматичної перевірки тестових бланків.

1.1.7 Переваги та обмеження використання комп'ютерного зору для автоматичної перевірки тестів

Переваги:

- Швидкість: автоматичні системи перевірки тестів працюють набагато швидше за людину, що дозволяє обробляти великі обсяги тестових бланків за короткий час.
- Точність: використання комп'ютерного зору мінімізує можливість людської помилки, забезпечуючи високу точність перевірки.
- Автоматизація процесу: автоматичні системи дозволяють повністю автоматизувати процес перевірки тестів, що звільняє персонал від рутинної роботи.
- Гнучкість: системи на основі комп'ютерного зору можуть бути налаштовані для роботи з різними типами тестових бланків та умовами освітлення.

Обмеження:

- Залежність від якості зображення: якість зображення має велике значення для точності розпізнавання. Погана якість зображення може призвести до помилок у визначенні заповнених кружечків.
- Чутливість до умов освітлення: зміни в умовах освітлення можуть впливати на якість зображення, що вимагає застосування додаткових методів обробки для компенсації цих змін.

- Складність налаштування: для досягнення високої точності необхідно правильно налаштувати систему, що може вимагати значних зусиль та знань в області комп'ютерного зору.

1.2 Огляд порівняльних характеристик подібних програм

1.2.1 Моніторинг

Моніторинг - це серія однотипних вимірювань об'єкта дослідження з подальшим аналізом, оцінкою та порівнянням отриманих результатів, з метою виявлення певних закономірностей, тенденцій, змінних та їх динаміки.

Для того, щоб розпочати роботу над додатком та створити його структуру, необхідно провести конкурентний моніторинг аналогічних додатків. Конкурентний моніторинг - це дослідження однієї, декількох або багатьох ідентичних або схожих систем паралельно за єдиною методологією. Це дозволяє оцінити і порівняти продуктивність систем, виявити відмінності між системами, а також встановити сильні і слабкі сторони.

Майже всі основні програми дуже добре розроблені з точки зору психології сприйняття. Вони мають чітку і ясну структуру, щоб користувачі могли швидко знайти те, що їм потрібно. Ніхто не віддає перевагу марним і незручним додаткам.

У технологіях просування є таке поняття, як сайт для людей. Для нього характерно наступне:

- гарна, інтуїтивно зрозуміла структура додатку;
- легкий і простий пошук потрібної інформації;
- всі тексти добре читані та не перевантажені ключовими фразами;
- інформація цікава користувачеві через її конкретність без зайвої води;
- реклама не випирає з усіх боків, а м'яко доповнює зміст сторінки;
- оформлення додатку не викликає роздратування, а чітко вписується в його призначення.

На даний момент немає програми, яка б перевіряла саме шаблон із завданням, але якщо порівняти подібні додатки за цими критеріями, то можна отримати для себе достатню кількість корисних і потрібних речей.

1.2.1 CamScanner

CamScanner - додаток який може зробити фотографії документів, а потім перетворити їх на PDF-файли. Він також має функцію розпізнавання тексту, яка може бути корисною для перевірки текстових завдань. Основні функції CamScanner включають:

- Сканування документів: Ви можете використовувати камеру вашого смартфона для сканування будь-якого документа. Додаток автоматично виправляє перспективу та підвищує якість зображення.
- Редагування та покращення зображень: Після сканування ви можете редагувати зображення, обрізати та видалити зайві елементи.
- Редагування та покращення зображень: Після сканування ви можете редагувати зображення, обрізати, виправляти колірність та видалити зайві елементи.
- Створення PDF: Після сканування документу ви можете зберегти його у форматі PDF, що дозволяє зручно обмінюватися ним з іншими користувачами.
- Синхронізація та резервне копіювання: CamScanner пропонує можливості синхронізації з обліковими записами хмарних служб, таких як Dropbox, Google Drive, а також автоматичне резервне копіювання сканованих документів.

Ось деякі переваги та недоліки програми CamScanner:

Переваги

- Зручність: CamScanner забезпечує швидкий та простий спосіб сканувати документи безпосередньо з вашого смартфона. Це особливо зручно в ситуаціях, коли потрібно швидко зробити копію документа або нотаток.
- Якість сканування: Додаток забезпечує високоякісні сканування, які можуть бути порівняні зі сканерами, що використовуються на робочому місці. Він автоматично коригує перспективу та покращує якість

зображення.

- OCR (розпізнавання тексту): Функція OCR дозволяє розпізнавати текст на сканованих документах, що робить їх пошук та редагування більш зручними.
- Можливості редагування: Ви можете легко редагувати скановані документи, обрізаючи, виправляючи кольорові налаштування та видаляючи непотрібні елементи.
- Синхронізація та резервне копіювання: CamScanner пропонує можливості синхронізації з хмарними службами для зручного зберігання та обміну документами.

Недоліки:

- Вартість деяких функцій: Деякі розширені функції, такі як OCR та синхронізація з хмарними службами, можуть бути доступні тільки у платній версії програми.
- Пристрої з низькими характеристиками: На деяких старих або менш потужних смартфонах може спостерігатися погіршення продуктивності при використанні CamScanner, особливо при обробці великої кількості документів.
- Приватність даних: Як і в будь-якій програмі, яка працює з особистими даними, існує питання щодо приватності. Користувачі повинні бути обережні зі збереженням конфіденційних або чутливих документів у хмарних сервісах або на пристроях.

1.2.3 Photomath

Photomath - це додаток, який допомагає вирішувати математичні завдання за допомогою камери смартфона. Він дозволяє користувачам просто направити камеру на математичне завдання, і програма автоматично розпізнає текст і надає розв'язок.

Основні особливості Photomath включають:

- Розпізнавання тексту: Додаток може розпізнавати математичний текст на

фотографіях, зроблених камерою смартфона. Це означає, що ви можете просто зробити фотографію математичного завдання, і Photomath автоматично розпізнає його.

- Розв'язання крок за кроком: Після розпізнавання тексту програма відображає послідовні кроки для розв'язання завдання. Це допомагає користувачам зрозуміти процес розв'язання задачі та отримати необхідні пояснення.
- Підтримка різноманітних математичних завдань: Photomath може вирішувати різні типи математичних завдань, включаючи алгебру, геометрію, тригонометрію, калькулюс тощо. Він також має функцію розпізнавання рукописного тексту, що дозволяє вирішувати завдання, написані вручну.
- Освітній контент: У Photomath є бібліотека математичних понять та прикладів, які можна використовувати для вивчення та вдосконалення навичок в математиці. Вона містить пояснення до різних концепцій та вправ для практики.
- Доступність: Додаток доступний як на iOS, так і на Android, що робить його доступним для широкого кола користувачів.

Ось деякі переваги та недоліки додатка Photomath:

Переваги

- Зручність використання: Photomath надає простий та зручний спосіб вирішення математичних завдань за допомогою камери смартфона. Ви просто фотографуєте завдання, і програма надає відповіді та пояснення.
- Розпізнавання тексту: Додаток добре розпізнає текст на фотографіях, що дозволяє швидко та ефективно вирішувати завдання.
- Пояснення крок за кроком: Photomath надає послідовні кроки для розв'язання завдання, що допомагає зрозуміти процес та підвищує рівень розуміння матеріалу.
- Різноманітність математичних завдань: Додаток підтримує багато різних типів математичних завдань, включаючи алгебру, геометрію та тригонометрію.
- Доступність і безкоштовність: Photomath доступний як на iOS, так і на Android,

і більшість основних функцій є безкоштовними для використання.

Недоліки

- Залежність від технологій: Хоча Photomath є потужним інструментом, він може розглядатися як "коструватий" метод, який може зменшувати розвиток навичок розуміння та розв'язання математичних завдань самостійно.
- Недостатня точність: Деякі користувачі відзначають, що програма не завжди розпізнає текст на фотографіях правильно, що може призводити до неточних результатів.
- Недоступність у навчанні вищих математичних концепцій: Photomath чудово підходить для базових та середніх рівнів математики, але може бути обмеженим у вирішенні складних математичних завдань або вищих математичних концепцій.
- Необхідність інтернет-підключення для деяких функцій: Деякі функції, такі як перевірка та завантаження додаткового освітнього контенту, можуть вимагати доступу до Інтернету.

1.2.4 Microsoft Math Solver

Microsoft Math Solver - це додаток від Microsoft, який допомагає вирішувати математичні завдання та отримувати пояснення крок за кроком. Основні функції додатка включають:

- Розпізнавання тексту: Додаток дозволяє користувачам завантажувати або фотографувати математичні завдання, а потім автоматично розпізнавати текст на зображеннях.
- Розв'язання завдань: Після розпізнавання тексту додаток пропонує варіанти розв'язання завдання. Користувач може вибрати потрібний метод розв'язання та отримати відповідь.
- Пояснення крок за кроком: Крім розв'язання, Microsoft Math Solver надає пояснення кожного кроку розв'язку завдання. Це допомагає зрозуміти процес та логіку розв'язання.
- Додатковий матеріал: Додаток містить багато додаткового матеріалу

для вивчення та практики математики, такий як підручники, відеоуроки та вправи.

- Різноманітність математичних тем: Microsoft Math Solver підтримує широкий спектр математичних тем, включаючи алгебру, геометрію, тригонометрію, калькулюс та інші.
- Інтерактивність: Додаток дозволяє користувачам використовувати інтерактивні елементи для взаємодії з математичними об'єктами та концепціями.

Ось деякі переваги та недоліки Microsoft Math Solver:
Переваги

- Широкі можливості розв'язання: Додаток підтримує багато різних математичних тем та методів розв'язання, що дозволяє вирішувати різноманітні завдання.
- Пояснення крок за кроком: Microsoft Math Solver надає чіткі та докладні пояснення кожного кроку розв'язку, що допомагає зрозуміти процес та концепції.
- Додатковий освітній матеріал: Додаток містить велику кількість додаткового матеріалу, який може бути корисним для вивчення та практики математики.

Недоліки

- Необхідність Інтернет-підключення: Деякі функції додатка можуть вимагати доступу до Інтернету, що може бути не зручно в деяких ситуаціях.
- Обмеження використання безкоштовної версії: Деякі розширені функції можуть бути доступні лише у платній версії додатку.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО РІШЕННЯ

2.1 Вибір технологій

Для розробки мобільного застосунку було обрано Kotlin як основну мову програмування. Kotlin — це сучасна статично типізована мова програмування, розроблена компанією JetBrains. Вона створена для сумісності з Java та роботи на платформі Java Virtual Machine (JVM). Перша версія Kotlin була випущена у 2011 році, і з того часу мова отримала велику популярність серед розробників, особливо у сфері розробки Android-застосунків.

Основні особливості Kotlin:

- Лаконічність: Код на Kotlin зазвичай коротший та зрозуміліший порівняно з кодом на Java. Це досягається завдяки використанню сучасного синтаксису та усуненню зайвих елементів.
- Безпечність: Kotlin значно знижує кількість потенційних помилок, таких як NullPointerException, за рахунок впровадження концепції безпеки щодо null-посилань.
- Інтероперабельність з Java: Код на Kotlin може працювати разом з Java-кодом у одному проекті, що дозволяє розробникам поступово мігрувати з Java на Kotlin.
- Функціональне програмування: Kotlin підтримує функціональні парадигми програмування, включаючи лямбда-вирази, функції вищого порядку, каррінг та інші.
- Розширюваність: Мова дозволяє створювати розширення для існуючих класів без необхідності їх змінювати, що полегшує роботу з сторонніми бібліотеками.
- Сучасні конструкції мови: Kotlin підтримує сучасні конструкції, такі як data-класи, sealed-класи, деструктуризацію та інші, що спрощує розробку та підвищує продуктивність.

Для реалізації функцій комп’ютерного зору була обрана OpenCV.

Open Source Computer Vision Library — це відкрита бібліотека комп'ютерного зору та машинного навчання, яка була створена для забезпечення швидкого виконання програм на багатьох платформах. Вона була розроблена компанією Intel і вперше випущена у 2000 році. Основною метою OpenCV є надання інфраструктури для комп'ютерного зору, що включає в себе великий набір алгоритмів для обробки зображень і відео. Основні можливості OpenCV:

- Обробка зображень: OpenCV містить безліч функцій для обробки зображень, таких як фільтрація, перетворення кольорів, порогове значення, морфологічні операції та геометричні перетворення.
- Аналіз зображень: Бібліотека дозволяє проводити аналіз зображень, включаючи виявлення контурів, сегментацію, аналіз форм і розпізнавання об'єктів.
- Комп'ютерний зір: OpenCV містить алгоритми для виявлення облич, відстеження об'єктів, розпізнавання жестів, виявлення руху та багато іншого.
- Машинне навчання: В OpenCV включені алгоритми машинного навчання, такі як класифікатори на основі алгоритмів дерева рішень, нейронні мережі, підтримуючі векторні машини (SVM) та інші.
- Робота з відео: Бібліотека підтримує зчитування і запис відео файлів, а також роботу з потоковим відео.
- Підтримка різних мов програмування: OpenCV підтримує кілька мов програмування, включаючи C++, Python, Java і MATLAB, що робить її доступною для широкого кола розробників.

2.2 Архітектура застосунку

Архітектура мобільного застосунку складається з кількох основних компонентів:

- Інтерфейс користувача (UI): Забезпечує взаємодію користувача із застосунком.

- Модуль обробки зображень: Відповідає за захоплення та попередню обробку зображень.
- Модуль розпізнавання: Використовує алгоритми OpenCV для розпізнавання заповнених тестових бланків.
- Модуль оцінювання: Порівнює розпізнані відповіді з еталонними та виставляє оцінки.

2.2.1 Інтерфейс користувача

Інтерфейс користувача був розроблений з урахуванням простоти використання та інтуїтивності. Основні елементи інтерфейсу:

- Екран захоплення зображень: Дозволяє користувачу зробити фото тестового бланку.
- Кнопка для захоплення правильних відповідей.
- Екран для розпізнавання та оцінювання бланків у прямому ефірі.

2.2.2 Модуль обробки зображень

Цей модуль відповідає за захоплення зображень з камери телефону та їх попередню обробку. Основні етапи обробки включають:

- Перетворення в сірий колір: Знижує складність подальшої обробки.
- Бінаризація: Перетворює зображення в двокольорове, що дозволяє легше виявити заповнені кружечки.
- Масштабування: Забезпечує правильне співвідношення розмірів зображення для точного аналізу.

2.2.3 Модуль розпізнавання

Модуль розпізнавання використовує алгоритми OpenCV для виявлення та аналізу кружечків на тестовому бланку. Основні завдання цього модуля включають:

- Виявлення контурів: Алгоритм знаходить контури на бланку, які можуть відповідати заповненим чи незаповненим кружечкам.

- Фільтрація та класифікація: Відфільтровує непотрібні контури та класифікує знайдені об’єкти як заповнені або незаповнені кружечки на основі визначених критеріїв.
- Перетворення координат: Забезпечує правильне розміщення знайдених об’єктів відносно координат бланку для подальшого аналізу.

2.2.4 Модуль оцінювання

Модуль оцінювання відповідає за порівняння розпізнаних відповідей з еталонними відповідями та виставлення оцінок. Основні етапи включають:

- Завантаження еталонних відповідей: Програма отримує еталонні відповіді, з якими будуть порівнюватись результати.
- Порівняння: Порівнюються заповнені та незаповнені кружечки на тестовому бланку з еталонними відповідями.
- Розрахунок оцінки: Оцінка виставляється за шкалою від 0 до 100 балів, залежно від кількості правильних відповідей.

2.3 Тестування застосунку

2.3.1 Введення у тестування

Тестування - це процес перевірки програмного забезпечення на відповідність специфікаціям, вимогам та очікуванням. Це систематичне дослідження, яке здійснюється з метою виявлення помилок, дефектів чи недоліків у програмах. Основна мета тестування полягає в забезпеченні якості програмного продукту та впевненості у його правильному функціонуванні перед його впровадженням або випуском на ринок.

Тестування в IT сфері важливе з кількох причин:

- **Забезпечення якості продукту**: Тестування в IT - це важлива складова процесу розробки програмного забезпечення, яка спрямована на забезпечення його якості. Це не просто про виявлення помилок, а про перевірку відповідності продукту вимогам, специфікаціям та очікуванням користувачів. Якість програми визначається не лише її функціональністю, а

й надійністю, продуктивністю, безпекою та іншими аспектами. Тестування допомагає виявити недоліки в цих аспектах та виправити їх перед випуском продукту на ринок.

- **Економія часу та ресурсів**: Помилки в програмному забезпеченні можуть бути дорогими у виправленні, якщо вони виявляються після випуску продукту. Тестування на ранніх стадіях розробки дозволяє виявити та виправити ці помилки на етапі, коли вони ще не накопичилися або не мають серйозних наслідків. Це дозволяє заощадити час і ресурси, які інакше довелося б витратити на виправлення проблем пізніше.
- **Впевненість користувачів**: Користувачі очікують від програмного забезпечення стабільності та надійності. Правильно протестовані програми, які працюють безперебійно та відповідають їхнім потребам, сприяють позитивному досвіду використання та підвищують довіру користувачів до бренду чи компанії, що створила це програмне забезпечення.
- **Зменшення ризику втрати даних**: У сучасному цифровому світі безпека є одним з найважливіших аспектів розробки програмного забезпечення. Тестування на вразливості допомагає виявити можливі ризики та недоліки, які можуть призвести до втрати даних або порушення конфіденційності. Забезпечення безпеки програмного забезпечення шляхом тестування допомагає зберегти репутацію компанії та захистити інформацію користувачів.
- **Підтримка конкурентоспроможності**: У світі, де конкуренція на ринку IT продуктів є дуже великою, якість та надійність програмного забезпечення стають ключовими факторами успіху. Компанії, які інвестують у тестування та забезпечують високу якість своїх продуктів, здатні зберігати та залучати більше клієнтів, ніж їхні конкуренти.

2.3.2 Види тестування

Модульне тестування (Unit Testing) - це процес перевірки окремих компонентів або "одиниць" програмного коду для переконання, що вони працюють правильно. Одна "одиниця" - це найменша частина програмного коду, яка може

бути перевірена окремо, наприклад, функція, метод або клас. Під час модульного тестування ці одиниці тестуються в ізоляції від інших частин програми.

Основна мета модульного тестування - впевнитися, що кожна частина програми працює належним чином, відповідно до очікуваного результату. Це допомагає виявляти помилки та дефекти на ранніх етапах розробки, забезпечуючи високу якість програмного забезпечення.

Зазвичай модульні тести автоматизовані, що дозволяє їх легко виконувати при кожній зміні коду. Популярні інструменти для модульного тестування включають JUnit для Java, NUnit для .NET, pytest для Python, та інші.

Інтеграційне тестування (Integration Testing) - це процес перевірки взаємодії між різними компонентами або модулями програмного забезпечення, щоб забезпечити їх спільну роботу так, як очікується. Під час інтеграційного тестування перевіряється, як різні частини програми взаємодіють одна з одною, як вони обмінюються даними та як вони впливають одна на одну.

Основна мета інтеграційного тестування - виявлення помилок або невідповідностей у взаємодії між компонентами програмного забезпечення. Це може включати виявлення проблем з інтерфейсами, неправильним форматом передачі даних, або несправним спільним функціоналом.

Існують різні підходи до інтеграційного тестування, такі як верхнє-донизу (top-down), низьке-вгору (bottom-up), або гібридний підхід. У верхньому-донизу підході тестування починається з вищого рівня програми і поступово переходить до нижчих рівнів, тоді як у низькому-вгору підході тестування починається з низького рівня та поступово переходить до вищих рівнів. Гібридний підхід комбінує обидва підходи.

Системне тестування (System Testing) - це вид тестування програмного забезпечення, який оцінює поведінку і функціональність повної інтегрованої системи. У цьому типі тестування програма тестується як єдина сутність, а не окремі компоненти чи модулі.

Основна мета системного тестування - переконатися, що програма відповідає вимогам, включаючи як функціональні, так і нефункціональні аспекти. Це включає

в себе перевірку виконання функцій програми відповідно до її специфікації, а також оцінку таких аспектів, як продуктивність, безпека, надійність та зручність використання.

У системному тестуванні тести створюються на основі вимог до програми, таких як специфікації продукту, вимоги користувача та інші документи. Тестові сценарії охоплюють різні функціональність системи та ситуації використання, щоб впевнитися, що вона працює належним чином в реальних умовах. Результати системного тестування документуються для подальшого аналізу та виправлення виявлених проблем.

Тестування на прийняття (Acceptance Testing) - це вид тестування програмного забезпечення, який виконується з метою перевірки, чи відповідає розроблене програмне забезпечення вимогам та очікуванням замовника чи кінцевих користувачів перед його остаточним впровадженням.

Цей вид тестування може включати різні підходи, такі як:

- **Функціональне тестування**: Перевірка, чи працює програмне забезпечення згідно зі специфікованими функціональними вимогами. Це включає в себе тестування різних функцій, операцій та сценаріїв використання програми.
- **Непряме тестування (Usability Testing)**: Оцінка того, наскільки зручно користуватися програмним забезпеченням та як легко користувачі можуть виконувати різні завдання з його допомогою.
- **Валідація вимог (Requirements Validation)**: Перевірка, чи відповідає програмне забезпечення вимогам та очікуванням, що були встановлені на початкових етапах розробки.
- **Тестування сумісності (Compatibility Testing)**: Перевірка того, як програмне забезпечення працює на різних пристроях, платформах та середовищах.
- **Тестування з точки зору користувача (User Acceptance Testing, UAT)**: Це остаточний етап тестування, коли програмне забезпечення передається кінцевим користувачам або представникам замовника для використання та оцінки.

У цілому, тестування на прийняття допомагає забезпечити, що розроблене програмне забезпечення відповідає очікуванням та вимогам користувачів перед його впровадженням у виробниче середовище.

Регресійне тестування (Regression Testing) - це вид тестування програмного забезпечення, який здійснюється для переконання, що зміни, внесені до програмного коду під час розробки або покращення, не спричинили появу нових помилок або не вплинули негативно на існуючу функціональність.

Основна мета регресійного тестування - підтвердження того, що попередні функції та функціональність програмного забезпечення залишаються працездатними після внесення змін. Це важливо, оскільки в процесі розробки можуть виникати нові функції, зміни або виправлення помилок, які можуть навіть непомітно вплинути на існуючу функціональність.

Під час регресійного тестування зазвичай використовуються раніше створені тестові набори, які включають тестові випадки, які вже були виконані для попередніх версій програмного забезпечення. Ці тестові набори повторно виконуються, щоб переконатися, що нічого не зламалось після внесення змін.
Регресійне тестування часто автоматизується для ефективності та швидкості, особливо коли програмне забезпечення має великий обсяг тестових випадків. Це допомагає забезпечити швидке виявлення будь-яких негативних впливів нових змін на існуючу функціональність.

Тестування продуктивності (Performance Testing) - це вид тестування програмного забезпечення, спрямований на оцінку його швидкості, відгуку та стабільності під час різних умов навантаження. Основна мета тестування продуктивності - визначити, як швидко та ефективно працює програмне забезпечення при різних навантаженнях та обсягах даних.
Тестування продуктивності включає такі види тестів:

- **Навантажувальне тестування (Load Testing)**: Цей вид тестування оцінює, як програмне забезпечення веде себе під час нормального та пікового

навантаження. Його мета - визначити максимальне навантаження, яке програма може обробити, та як це впливає на її продуктивність.

- **Стрес-тестування (Stress Testing)**: Цей вид тестування оцінює стійкість програмного забезпечення під великими навантаженнями або екстремальними умовами. Його мета - визначити, як програмне забезпечення поводиться під час перевищення його максимальних можливостей та як швидко воно може відновитися після стресової ситуації.
- **Стійкість (Endurance Testing)**: Цей вид тестування перевіряє, як довго програмне забезпечення може працювати під навантаженням без виявлення важливих проблем з продуктивністю або стабільністю.
- **Тестування пропускної здатності (Throughput Testing)**: Цей вид тестування оцінює кількість даних або операцій, які програмне забезпечення може обробляти протягом певного часу, щоб визначити його пропускну здатність.

Тестування продуктивності допомагає виявити потенційні проблеми з продуктивністю програмного забезпечення та покращити його ефективність та стабільність перед випуском у виробництво.

Тестування навантаження (Load Testing) - це вид тестування програмного забезпечення, яке спрямоване на визначення, як програма веде себе під час роботи під нормальним та піковим навантаженням. Основна мета тестування навантаження - визначити, як програма реагує на певні рівні навантаження та чи зберігає вона свою продуктивність та стабільність.

Під час тестування навантаження, програма піддається різним рівням навантаження, які можуть включати в себе одночасних користувачів, обсяги даних або транзакцій, та періоди з високим трафіком. Вимірюються такі параметри, як час відгуку, обробка запитів, навантаження на сервер, та інші, щоб визначити, як програма реагує на ці умови.

Переваги тестування навантаження включають:

- Виявлення потенційних проблем з продуктивністю: Тестування навантаження допомагає виявити слабкі місця програми, які можуть виникнути при великому обсязі роботи або пікових навантаженнях.
- Впевненість в стабільності: Цей вид тестування допомагає переконатися, що програма залишається стабільною та працездатною навіть під час інтенсивного використання.
- Планування ємності: Результати тестування навантаження можуть бути використані для планування ресурсів та ємності, необхідних для підтримки програми в реальному середовищі.

Стрес-тестування (Stress Testing) - це вид тестування програмного забезпечення, який спрямований на визначення меж його стійкості та надійності в екстремальних умовах або при великих навантаженнях. Основна мета стрес-тестування - виявлення точок вразливості, перегрузок або інших проблем, які можуть виникнути під час інтенсивного використання програмного забезпечення.

Під час стрес-тестування, програма піддається великим обсягам даних, інтенсивним операціям або піковим навантаженням, які перевищують нормальні умови роботи. Мета полягає в тому, щоб визначити, як програма поводиться під цими стресовими умовами та як вона відновлюється після перевантаження.

Переваги стрес-тестування включають:

- Виявлення точок вразливості: Стрес-тестування допомагає виявити слабкі місця програми, які можуть призвести до аварій або відмов під час екстремальних умов.
- Підтвердження стійкості: Тестування дозволяє переконатися, що програмне забезпечення залишається стабільним та працездатним навіть під час стресових ситуацій.
- Покращення якості: Аналіз результатів стрес-тестування допомагає розробникам вдосконалити програму та виправити проблеми, які можуть виникнути в реальних умовах використання.

Тестування безпеки (Security Testing) - це процес оцінки вразливостей програмного забезпечення та інформаційних систем з метою забезпечення їх захищеності від несанкціонованого доступу, атак або втрати конфіденційності, цілісності та доступності даних.

Основна мета тестування безпеки - ідентифікація потенційних слабких місць у системі та застосунках, які можуть бути використані зловмисниками для виконання різних видів атак, таких як внедрення коду, витік даних, переповнення буфера та інші.

Процес тестування безпеки може включати такі етапи:

- **Аналіз загроз і загроз доходжень (Threat Modeling)**: Визначення потенційних загроз та оцінка їхнього впливу на систему.
- **Тестування на вразливість (Vulnerability Testing)**: Виявлення потенційних вразливостей у системі або програмному забезпеченні, які можуть бути використані для здійснення атак.
- **Тестування аутентифікації та авторизації (Authentication and Authorization Testing)**: Оцінка механізмів перевірки ідентичності користувачів та їх доступу до ресурсів.
- **Тестування шифрування та захисту даних (Encryption and Data Protection Testing)**: Перевірка ефективності шифрування та захисту конфіденційних даних.
- **Тестування безпеки мережі (Network Security Testing)**: Оцінка захищеності мережевих протоколів та мережевих пристроїв від атак на рівні мережі.
- **Тестування на проникнення (Penetration Testing)**: Спроба вдатися до системи або програми з метою виявлення вразливостей та поборення їхньої захищеності.

Тестування зручності використання (Usability Testing) - це процес оцінки того, наскільки легко та ефективно користувачі можуть взаємодіяти з програмним

забезпеченням, веб-сайтами, мобільними додатками або будь-якими іншими продуктами.

Основна мета тестування зручності використання - визначити, наскільки продукт задовольняє потреби та очікування кінцевих користувачів щодо його інтерфейсу та функціональності. Це включає оцінку таких аспектів, як:

- **Легкість навігації**: Чи легко користувачі можуть знаходити потрібні функції та інформацію.
- **Чіткість і зрозумілість інтерфейсу**: Чи зрозумілі та інтуїтивно зрозумілі мітки, кнопки, меню та інші елементи інтерфейсу.
- **Ефективність використання**: Наскільки швидко та ефективно користувачі можуть виконувати завдання з використанням продукту.
- **Задоволення користувача**: Як користувачі реагують на взаємодію з продуктом, чи вони відчувають задоволення від користування ним.
- **Адаптивність та доступність**: Чи легко використовувати продукт людям з різними рівнями знань та здатностей, включаючи людей з обмеженими можливостями.

Тестування зручності використання може включати різні методи, такі як фокус-групи, спостереження за користувачами, анкетування, а також тестування на основі сценаріїв використання. Результати цього виду тестування допомагають розробникам покращити дизайн та функціональність продукту з метою забезпечення оптимального досвіду користувача.

Тестування сумісності (Compatibility Testing) - це процес перевірки, як програмне забезпечення або додаток взаємодіє з різними пристроями, операційними системами, браузерами, мережами або іншими компонентами середовища, на якому воно має працювати. Основна мета тестування сумісності - переконатися, що програмне забезпечення або додаток може працювати на різних платформах та конфігураціях без проблем та відповідати вимогам користувачів.
Під час тестування сумісності виконуються такі дії:

- **Тестування на різних пристроях**: Перевірка роботи програмного забезпечення на різних моделях та типах пристроїв, таких як комп'ютери, смартфони, планшети тощо.
- **Тестування на різних операційних системах**: Перевірка сумісності програмного забезпечення з різними операційними системами, такими як Windows, macOS, Linux, Android, iOS тощо.
- **Тестування на різних браузерах**: Перевірка роботи веб-сайтів або веб-додатків на різних веб-браузерах, таких як Google Chrome, Mozilla Firefox, Microsoft Edge, Safari тощо.
- **Тестування на різних мережах**: Перевірка роботи програмного забезпечення в різних мережних умовах, таких як LAN, Wi-Fi, мобільні мережі тощо.
- **Тестування на різних версіях і компонентах середовища**: Перевірка роботи програмного забезпечення з різними версіями додатків, бібліотек, плагінів тощо, які можуть використовуватися на специфічних конфігураціях.

Результати тестування сумісності допомагають забезпечити, що програмне забезпечення або додаток працює на різних платформах та середовищах без проблем і відповідає очікуванням користувачів.

Дослідницьке тестування (Exploratory Testing) - це метод тестування програмного забезпечення, в якому тестер виконує тестування без попередньо підготовлених скриптів або детального плану. Замість цього тестер досліджує програмне забезпечення, виконуючи тестові сценарії та тест-кейси на основі свого досвіду, інтуїції та знань про домен.

Основні принципи дослідницького тестування включають:

- **Активне дослідження**: Тестер активно вивчає програмне забезпечення, спробується зрозуміти його функціональність, можливі сценарії використання та потенційні точки вразливості.

- **Гнучкість**: Тестувальник вільно вибирає, які тест-кейси виконувати, в залежності від виявлених відхилень або специфічних областей, які потребують додаткового тестування.
- **Багатократні ітерації**: Тестер може повторно виконати тестові сценарії з різними варіаціями або в різних послідовностях, щоб виявити нові або приховані проблеми.
- **Реактивне тестування**: Тестер реагує на виявлені проблеми або незвичайність, змінюючи свій підхід до тестування та перевіряючи області, які можуть бути пов'язані з цими проблемами.

Дослідницьке тестування особливо ефективне в ситуаціях, коли недостатньо інформації для попереднього розроблення детальних тест-кейсів або коли важко передбачити всі можливі сценарії використання. Цей підхід дозволяє тестувальникам ефективно виявляти дефекти та проблеми, що можуть залишитися непоміченими в інших методах тестування.

2.3.3 Обрані методи тестування

У процесі планування мною було обрано два основних методи тестування застосунку: тестування сумісності та тестування зручності використання. Ці методи забезпечать комплексну перевірку функціональності застосунку, а також його відповідність очікуванням користувачів.

3 РОЗРОБКА ЗАСТОСУНКА

У цій главі детально розглядається процес розробки та впровадження мобільного застосунку для автоматичної перевірки тестів. Ми зосередимося на кожному етапі розробки: від початкового планування та проектування до тестування та запуску в експлуатацію. Також обговоримо використані інструменти та технології, а також труднощі, з якими довелося зіткнутися, і як вони були подолані.

3.1 Планування проекту

Планування є важливим етапом розробки, який визначає успіх усього проекту. Включає визначення цілей, вимог, часових рамок і ресурсів.

Визначення цілей та вимог:

Функціональні вимоги — це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку: обчислення даних, маніпулювання даними, обробка даних та інші специфічні функції, які має виконувати система.

Відповідно до поставленого завдання необхідно розробити android-застосунок для перевірки екзаменаційних тестових бланків. Користувач повинен мати можливість:

- Надавати додатку доступ до камери
- За допомогою камери фіксувати бланк з правильними відповідями
- За допомогою камери перевіряти співпадіння похідного бланку з бланком правильних відповідей.

Нефункціональні вимоги — це вимоги до програмного забезпечення, які задають критерії для оцінки якості його роботи. На відміну від функціональних вимог, які визначають що система повинна робити, нефункціональні вимоги визначають якою система повинна бути. Отже

- Працювати “з коробки”.
- Мати мінімалістичний та зрозумілий інтерфейс.

- Користувач має чітко бачити, які відповіді вважаються програмою правильними, а які - ні.

Для досягнення цих цілей необхідно врахувати кілька ключових аспектів, таких як зручність користування, швидкість роботи застосунку, сумісність з різними типами мобільних пристроїв та забезпечення високої точності розпізнавання відповідей.

Часові рамки та ресурси:

Розробка проекту була поділена на кілька етапів:

- Дослідження та підготовка (1 тиждень): Аналіз ринку, збір вимог, створення технічного завдання.
- Проектування архітектури (1 тиждень): Визначення архітектури застосунку, вибір технологій.
- Розробка та інтеграція основних модулів (2 тижні): Реалізація основного функціоналу.
- Тестування та налагодження (1 тиждень): Тестування застосунку, виправлення помилок.

3.2 Проектування архітектури

Проектування архітектури мобільного застосунку включає вибір структурних елементів і їх взаємодію.

3.2.1 Модель-Вид-Контролер (MVC)

Архітектура застосунку була побудована на основі паттерну MVC:

Модель: Управляє даними застосунку. Включає моделі для тестових бланків, відповідей, користувачів тощо.

Вид: Відповідає за відображення даних. Інтерфейс користувача, що включає екрани для сканування, перегляду результатів тощо.

Контролер: Взаємодіє з моделлю та видом, обробляє події користувача, управляє потоком даних.

3.3 Вибір інструментів та технологій

Kotlin як основна мова програмування, що має ряд переваг:

- Безпека типів: Kotlin значно знижує кількість помилок, пов'язаних із типізацією.
- Інтероперабельність: Kotlin повністю сумісний з Java, що дозволяє використовувати вже існуючі бібліотеки.
- Сучасний синтаксис: Спрощує процес розробки та підвищує читабельність коду.

OpenCV була обрана для реалізації функцій комп'ютерного зору завдяки її широким можливостям та підтримці мобільних платформ. OpenCV надає інструменти для обробки зображень, що є необхідними для розпізнавання та аналізу тестових бланків.

Android Studio. Це інтегроване середовище розробки (IDE) для створення мобільних додатків під операційну систему Android. Розроблене компанією Google, Android Studio надає розробникам широкий набір інструментів та ресурсів для створення якісних та ефективних додатків. Ось деякі з його основних переваг:

- Інтуїтивний інтерфейс. Android Studio має зручний та дружелюбний інтерфейс, що полегшує роботу розробників та дозволяє швидко оволодіти середовищем.
- Широкий набір інструментів. У склад Android Studio входять різноманітні інструменти для розробки, від дизайну інтерфейсу до тестування та налагодження додатків.
- Емулятор Android. Android Studio має вбудований емулятор Android, який дозволяє розробникам тестувати свої додатки на різних пристроях та версіях операційної системи Android без необхідності в реальних пристроях.
- Підтримка мов програмування. Android Studio підтримує різні мови

програмування, зокрема Java та Kotlin, що дає розробникам можливість обирати найзручніший варіант для своїх потреб.

- Інтеграція з іншими сервісами Google. Android Studio має вбудовану підтримку сервісів Google, таких як Firebase, що дозволяє розробникам швидко і легко інтегрувати різноманітні функціональні можливості у свої додатки. Активна спільнота користувачів. Як продукт Google, Android Studio користується підтримкою великої та активної спільноти розробників, що забезпечує швидке розв'язання проблем та доступ до великої кількості ресурсів та підказок.

3.4 Розробка модулів

Процес розробки включає створення окремих модулів, які згодом інтегруються у повноцінний застосунок.

3.4.1 Модуль обробки зображень

Основні завдання цього модуля включають захоплення зображення з камери та його попередню обробку:

- **Перетворення в сірий колір**: Знижує складність обробки та підвищує швидкість роботи. Для перетворення зображення в градації сірого використовується метод Imgproc.cvtColor, який є частиною бібліотеки OpenCV. Цей метод перетворює зображення з одного кольорового простору в інший. В даному випадку, зображення перетворюється з кольорового (RGB) простору в сірий. Метод Imgproc.cvtColor здійснює перетворення кольорового зображення в сіре шляхом обчислення вагового середнього значення для кожного пікселя. Це дозволяє зменшити кількість каналів з трьох (R, G, B) до одного. Як це працює:
  - Вхід: кольорове зображення (RGB).
  - Кожний піксель зображення обробляється так: $Gray=0.299\times R+0.587\times G+0.114\times B$
  - Вихід: зображення у відтінках сірого.
- **Бінаризація:** Перетворює зображення в двокольорове, що полегшує

виявлення заповнених кружечків. Для бінаризації (порогової обробки) зображення використовується метод Imgproc.adaptiveThreshold. Цей метод застосовує адаптивне порогове значення до зображення, що дозволяє розділити його на бінарне (чорно-біле). Для кожного пікселя обчислюється середнє значення інтенсивності в його околі розміром THRESHOLD_BLOCK_SIZE. Якщо інтенсивність пікселя нижче середнього на певну константу THRESHOLD_CONSTANT, піксель стає білим, інакше - чорним.

Метод Imgproc.adaptiveThreshold приймає три основні параметри:

- **adaptiveMethod** - визначає, як обчислюється порогове значення:
  - Imgproc.ADAPTIVE_THRESH_MEAN_C: Порогове значення є середнім значенням сусідньої області мінус константа C.
  - Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C: Порогове значення є гаусівською зваженою сумою значень сусідніх пікселів мінус константа C.
- **blockSize** - визначає розмір сусідньої області.
- **C** - константа, яка віднімається від середнього або зваженого значення сусідніх пікселів.

Також для простого порогового значення використовується функція Imgproc.threshold. Перший аргумент — це вихідне зображення у градаціях сірого. Другий аргумент — порогове значення для класифікації пікселів. Третій аргумент — максимальне значення для пікселів, що перевищують поріг. Четвертий параметр визначає тип порогового значення. Основний тип порогового значення — Imgproc.THRESH_BINARY. Інші типи:

- Imgproc.THRESH_BINARY
- Imgproc.THRESH_BINARY_INV
- Imgproc.THRESH_TRUNC
- Imgproc.THRESH_TOZERO
- Imgproc.THRESH_TOZERO_INV

Метод повертає два результати: перший — це використане порогове значення, а другий — зображення після застосування порогового значення.

Вихід: бінарне (чорно-біле) зображення.

- **Фільтрація шуму**: Видаляє зайві об'єкти з зображення, забезпечуючи більш точне розпізнавання. Для фільтрації шуму використовується метод заповнення областей за межами детекційної зони, щоб видалити зайві області, які не є частиною форми. Це досягається за допомогою заповнення цих областей чорним кольором. Як це працює:
    - Вхід: бінарне зображення.
    - Обчислюються координати зон, що знаходяться за межами бланка.
    - Ці зони заповнюються чорним кольором.
    - Вихід: бінарне зображення з відфільтрованим шумом.

3.4.2 Модуль розпізнавання відповідей

3.4.2.1 Формат бланка

Перш, ніж приступити до розпізнавання відповідей, треба вирішитися з форматом екзаменаційного бланка. На рисунку 3.1 зображений формат, на який далі буде орієнтуватися програма.

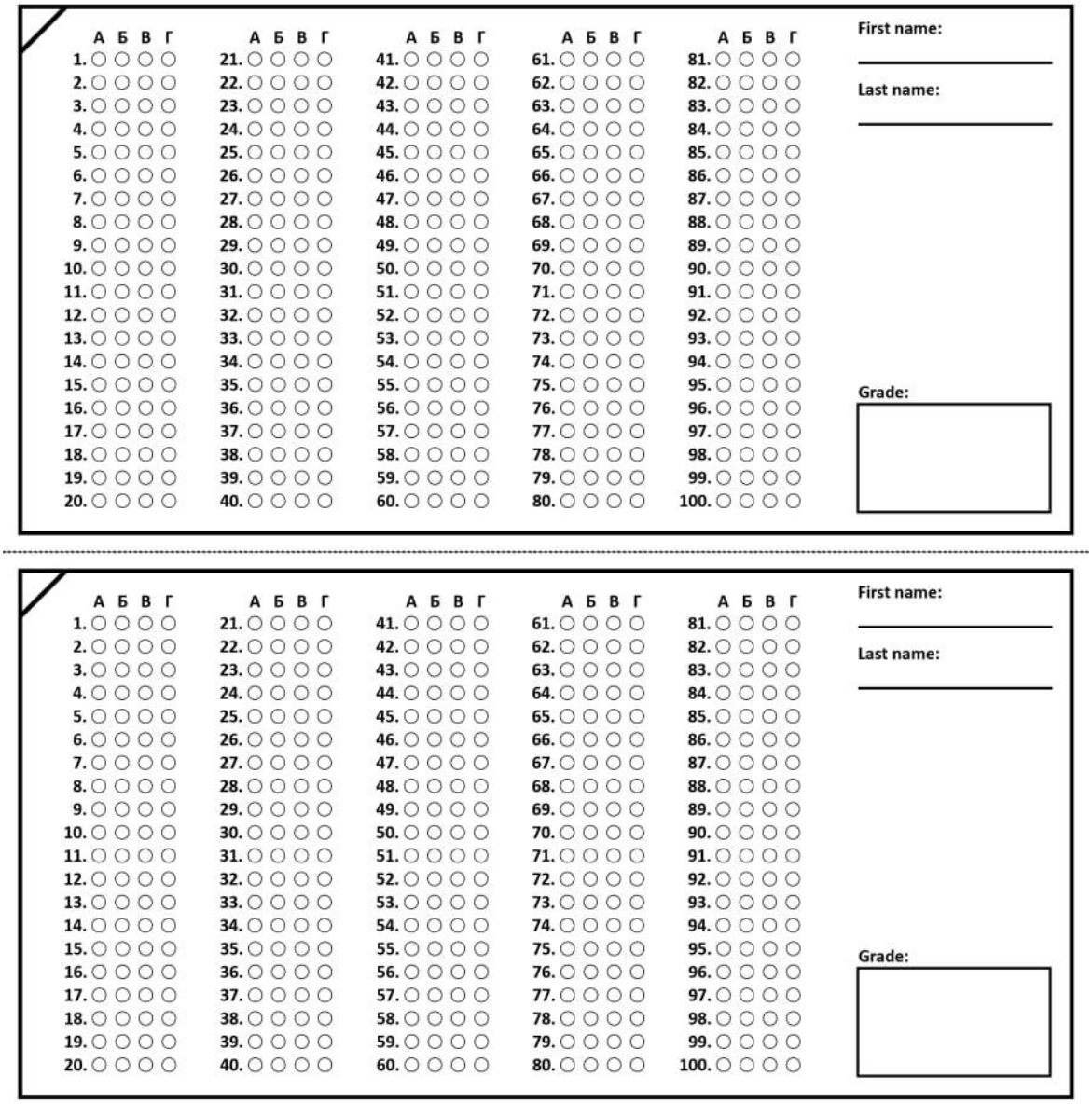


Рисунок 3.1 - формат екзаменаційного тестового бланку

3.4.2.2 Процес розпізнавання

Цей модуль використовує алгоритми OpenCV для розпізнавання заповнених та незаповнених відповідей на тестових бланках:

- Виявлення контурів: Алгоритм знаходить контури, які можуть відповідати заповненим чи незаповненим кружечкам. Використовуючи методи OpenCV, такі як findContours, програма аналізує бінарне зображення, яке отримане на попередньому етапі.
- Фільтрація та класифікація: Відфільтровує непотрібні контури та класифікує знайдені об’єкти на заповнені або незаповнені кружечки. Після фільтрації

контурів програма класифікує знайдені об'єкти на дві категорії: заповнені кружечки та незаповнені кружечки.

- Це може бути зроблено на основі інтенсивності пікселів у контурі або на основі площі контуру.
- Перетворення координат: Забезпечує правильне розміщення знайдених об’єктів відносно координат бланку для подальшого аналізу. Знайдені контури потрібно коректно розмістити відносно координат бланку для подальшого аналізу. Це виконується шляхом перетворення координат контурів у відповідності з координатами бланку. В результаті кожен контур відображає своє положення на бланку, що спрощує подальший аналіз.

3.4.3 Модуль оцінювання

Модуль оцінювання відповідає за порівняння розпізнаних відповідей з еталонними відповідями та виставлення оцінок. Основні етапи включають:

- Завантаження еталонних відповідей: Програма отримує еталонні відповіді, з якими будуть порівнюватися результати.
- Порівняння: Порівнюються заповнені та незаповнені кружечки на тестовому бланку з еталонними відповідями.
- Розрахунок оцінки: Оцінка виставляється за шкалою від 0 до 100 балів, залежно від кількості правильних відповідей.

3.4.4 Модуль користувацького інтерфейсу

Цей модуль відповідає за взаємодію з користувачем та включає:

- Інтерфейс для сканування: Забезпечує зручний спосіб сканування тестових бланків за допомогою камери смартфона.
- Інтерфейс для перегляду результатів: Дозволяє користувачам переглядати результати тестування у зрозумілому вигляді.

На рисунку 3.2 зображений інтерфейс перегляду відповідей на етапі оцінювання

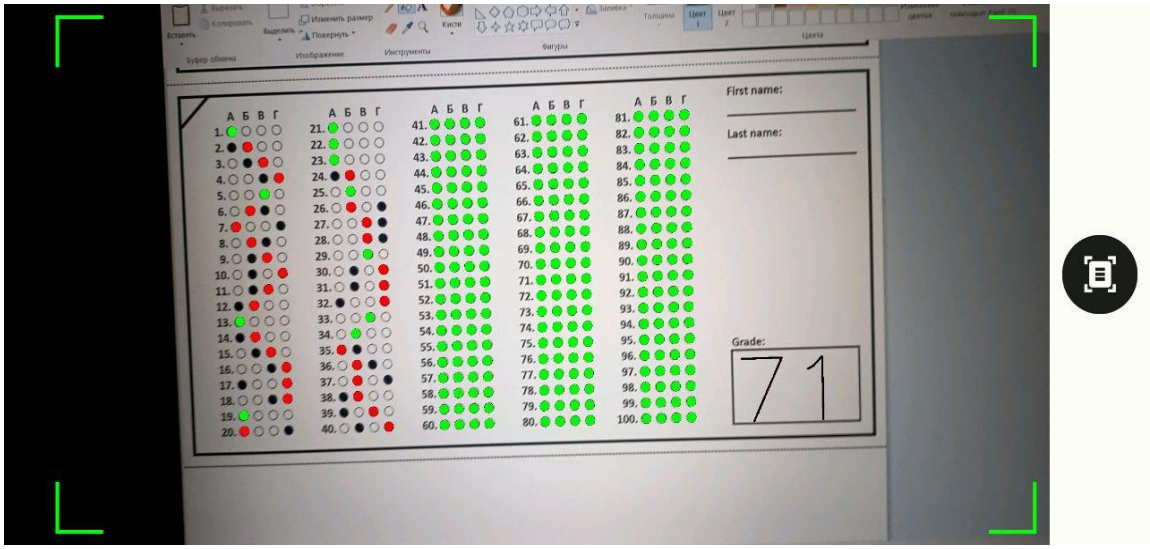


Рисунок 3.2 - інтерфейс перегляду результатів

3.5 Опис окремих елементів програми

Окремими елементами програми є класи, методи та функції.

Contour.kt

Contour.kt визначає клас Contour, який інкапсулює контур OpenCV MatOfPoint. Давайте розберемо його функціональність:

- У класі є властивість area, яка обчислює абсолютну площу контура.
- Є приватне властивість orientedArea, яка обчислює орієнтовану площу контура (площа з урахуванням орієнтації).
- Функція detectCorners() використовується для виявлення кутів контура. Вона виконує наступні кроки:
    - Перетворює MatOfPoint в MatOfPoint2f (тип матриці для операцій з плаваючою комою).
    - Обчислює периметр контура.
    - Апроксимує контур до багатокутника за допомогою алгоритму Рамера-Дугласа-Пекера (Imgproc.approxPolyDP).
    - Повертає список точок, що представляють виявлені кути. Порядок точок залежить від того, чи контур проти годинникової стрілки, чи ні. Якщо контур проти годинникової стрілки, то точки перевертаються.

ContourTree.kt

ContourTree - клас, який представляє дерево контурів для обробки зображень за допомогою бібліотеки OpenCV. Його основні елементи:

- Клас ContourTree має приватні властивості: список контурів (contours) та матрицю ієрархії контурів (contourHierarchy).
- Метод asList() повертає список контурів.
- Метод getChildren(contour: Contour) повертає список дочірніх контурів для вказаного контуру.
- Методи getSiblings(contour: Contour) та iterateContourIndices() використовуються для отримання списку сусідніх контурів та ітерації по індексах контурів у відповідності.

Клас має статичний метод createFrom(mat: Mat), який ініціалізує об'єкт ContourTree з матриці Mat. Він використовує функцію Imgproc.findContours для знаходження контурів у зображенні та їх ієрархії.

DetectionArea.kt

DetectionArea - клас, який представляє область виявлення на зображенні для аналізу форм. Основні аспекти цього коду:

- Клас DetectionArea має властивості, які представляють координати, ширину та висоту області виявлення, а також горизонтальний і вертикальний відступи.
- У статичному блоку companion object визначений метод calculate, який обчислює параметри області виявлення на основі ширини та висоти рамки зображення.
- Цей метод використовує константи MARGIN_FRACTION та PADDING_FRACTION для визначення відступів та величини відступів від країв рамки зображення.

- Розрахунок параметрів області виявлення залежить від того, чи обмежена область вертикально, або горизонтально, з урахуванням співвідношення сторін форми (значення FormMetrics.RATIO).

Отже, цей код допомагає автоматично розраховувати область виявлення на зображенні для подальшого аналізу форми.

DetectionAreaExt.kt

Цей файл розширює функціональність класу DetectionArea, додаючи кілька корисних властивостей та методів перетворення типів даних. Ось що робить кожний з них:

- Функція asInt() конвертує область виявлення з типів даних з плаваючою комою в цілочисельний тип. Вона використовує функцію ceil() для округлення до більшого цілого числа.
- Функція asFloat() конвертує область виявлення з типів даних з плаваючою комою в тип даних з плаваючою комою меншої точності (Float).
- Властивості paddedWidth і paddedHeight повертають ширину та висоту області виявлення без відступів.
- Властивості area і paddedArea обчислюють площу області виявлення з і без відступів відповідно.
- Властивості x2 і y2 повертають координати правого нижнього кута області виявлення (x + width, y + height) для цілочисельного та з плаваючою комою.

Detector.kt

Detector – об’єкт, який містить метод detect(), що виявляє форму на зображенні та повертає її позицію. Основні етапи алгоритму:

- Обчислення області виявлення за допомогою DetectionArea.calculate().
- Побудова дерева контурів з використанням ContourTree.createFrom().
- Пошук контуру форми, який містить в собі чотири кути.

- Для кожного такого контуру визначаються трикутні контури, що йому відповідають.
- Визначення, чи правильний розмір трикутного контуру за порівнянням з очікуваним розміром.
- Визначення найближчого кута контуру форми до трикутного контуру.
- Перевірка, чи відстань між найближчим кутом та трикутним кутом не перевищує максимальну допустиму відстань.
- Корекція орієнтації форми на основі найближчого кута та повернення його координат.

FormPosition.kt

FormPosition – це клас, який представляє позицію форми на зображенні за допомогою чотирьох кутових точок. Його основні елементи:

- Клас містить чотири точки (topLeft, topRight, bottomRight, bottomLeft) та список усіх кутів (corners).
- Існує вторинний конструктор, який дозволяє створювати об'єкт FormPosition з масиву точок, але перевіряє, що кількість точок дорівнює 4.
- Метод getBoundingBox() обчислює та повертає обмежуючий прямокутник (Rect), що охоплює форму, за допомогою мінімальних та максимальних координат кутових точок.
- Об'єкт-супутник Empty представляє порожню позицію форми, яка містить чотири нульові точки.

Цей клас корисний для представлення та роботи з позиціями форм на зображенні, зокрема для визначення їх розташування та обмежуючого прямокутника

Evaluator.kt

Evaluator - об’єкт, який відповідає за оцінку заповнених форм зображень. Основні етапи алгоритму:

- Метод evaluate() приймає зображення форми і розраховує результати оцінки для кожного блоку форми, перебираючи блоки та їх групи опцій.
- Метод resolveAnswer() вирішує, яким чином оцінювати результат для кожної групи опцій, перевіряючи, скільки опцій в групі було відмічено.
- Метод isOptionChecked() визначає, чи позначена певна опція в групі, обчислюючи відношення площі не нульових пікселів на площу круга, яка відповідає опції.
- Використовуються константи для толерантності при оцінці та масштабування додаткового простору для виявлення опцій.

Form.kt

Form - клас, який представляє форму та її відповіді.

- Клас Form має список відповідей (answers), який може бути порожнім за замовчуванням.
- Об'єкт-супутник Empty є пустою формою, що містить порожній список відповідей.
- Внутрішній інтерфейс Answer визначає типи можливих відповідей. Це:
    - Option, яка представляє вибрану опцію з номером.
    - Ambiguous, яка позначає, що відповідь неоднозначна.
    - Empty, яка позначає відсутність відповіді.

Цей клас призначений для моделювання структури форми та її відповідей у програмі, яка робить аналіз анкет або документів.

FormMetrics.kt

FormMetrics - клас, який відповідає за розміри та розташування елементів на формі. Основні елементи цього класу:

- Конструктор FormMetrics приймає ширину реального зображення форми та використовує її для обчислення масштабу та розмірів елементів.

- Клас містить властивості та константи, які визначають розміри і зсуви різних елементів форми.
- Властивості blocks та gradeBox створюються лінивим чином за допомогою функцій createBlocks() та createGradeBox() відповідно.
- Метод createBlocks() створює блоки форми разом з їхніми групами опцій та опціями.
- Клас також містить об'єкт-супутник Triangle, який визначає статичні параметри трикутника, такі як площа та відношення площі до площі форми.
- Об'єкт companion містить статичний метод fromActual(), який створює екземпляр FormMetrics на основі реальної ширини форми.
- Клас також визначає декілька вкладених класів для представлення блоків, груп опцій, опцій та оцінювальних рамок на формі.

Цей клас допомагає моделювати та працювати з розмірами та розташуванням елементів на формі.

FormPreprocessor.kt

FormPreprocessor - об’єкт який містить метод preprocess, що відповідає за попередню обробку зображень форм перед подальшим аналізом. Основна інформація про цей файл:

1. Метод preprocess(form: Mat) приймає матрицю Mat, яка представляє зображення форми, та виконує різні операції попередньої обробки на ній.
2. Внутрішній метод removeEmptyOptions() використовує морфологічні операції ерозії та дилатації для видалення порожніх опцій на формі. Для цього використовується ядро операцій, яке представлене квадратною структурою розміром CLEAN_KERNEL_SIZE x CLEAN_KERNEL_SIZE.

Цей код допомагає очищати та попередньо обробляти зображення форм перед подальшим аналізом, щоб покращити якість та точність результатів.

FramePreprocessor.kt

FramePreprocessor - це об’єкт, який містить метод preprocess, призначений для попередньої обробки кадрів перед аналізом. Інформація про цей об’єкт:

- Метод preprocess(frame: Mat) приймає матрицю Mat, яка представляє кадр, та виконує різні операції попередньої обробки на ній.
- Метод scale() масштабує кадр, щоб забезпечити мінімальний розмір сторони кадру як SCALING_MINIMUM_FRAME_SIDE. Повертає фактор масштабування.
- Метод toGrayscale() перетворює кадр у відтінки сірого.
- Метод toBlackAndWhite() використовує адаптивний поріг для перетворення кадру у чорно-білий зображення.
- Метод fillBeyondDetectionArea() заповнює області поза областю виявлення (визначеною за допомогою DetectionArea) чорним кольором.

Цей код допомагає підготувати кадри для подальшого аналізу, зокрема для виявлення областей інтересу та виявлення контурів на зображенні.

MatExt.kt

Цей файл містить код кількох розширених функцій для роботи з матрицями OpenCV (Mat). Призначення цих функцій:

- Функція submat приймає обмежуючий прямокутник Rect і функцію transform, яка застосовується до підматриці, створеної за цим обмежуючим прямокутником. Підматриця автоматично вивільняється після виконання функції transform.
- Функція copy створює копію поточної матриці та застосовує до неї функцію block. Після виконання функції block копія матриці вивільняється.
- Функція temporaryMat створює тимчасову порожню матрицю та застосовує до неї функцію block. Після виконання функції block тимчасова матриця вивільняється.

PreprocessingResult.kt

Клас PreprocessingResult використовується для збереження результатів попередньої обробки кадрів у об'єкті. Він містить одне поле:

- scaleFactor: представляє фактор масштабування, застосований до кадру під час попередньої обробки. Це число подається як значення типу Double. Цей фактор масштабування може бути використаний для відновлення реальних розмірів об'єктів на зображенні після масштабування.

ScanResult.kt

Клас ScanResult використовується для представлення результатів сканування форми. Він містить два поля:

- form: представляє результати аналізу самої форми із зібраними відповідями. Це поле типу Form, яке за замовчуванням має значення Form.Empty, що вказує на те, що форма порожня.
- formPosition: представляє позицію форми на зображенні. Це поле типу FormPosition, яке також має значення за замовчуванням FormPosition.Empty, що вказує на те, що позиція форми не була знайдена або не була визначена.
- Крім того, клас має об'єкт-компаньйон Empty, який представляє порожній об'єкт ScanResult для зручності використання в коді.

Scanner.kt

У цьому файлі визначений код об'єкта Scanner, який має метод scan, що виконує сканування форми на зображенні. Основна інформація про цей об’єкт:

- Метод scan(frame: Mat): ScanResult? приймає матрицю frame, яка представляє кадр зображення, і повертає результат сканування форми у вигляді об'єкта ScanResult або null, якщо форма не була знайдена на зображенні.
- У методі виконується наступне:
    - Кадр попередньо обробляється за допомогою об'єкта FramePreprocessor.

- Використовуючи об'єкт Detector, визначається позиція форми на зображенні.
- Після знаходження форми, вона підготовлюється за допомогою методу prepareForm, в якому виконується корекція перспективи, попередня обробка форми та її оцінка.
- Потім оцінений результат форми повертається як об'єкт ScanResult, в якому також вказується оригінальна позиція форми у вихідному масштабі зображення.

Цей код здійснює повний процес сканування форми на зображенні, включаючи виявлення форми, корекцію перспективи та оцінку її вмісту.

3.6 Тестування застосунку

3.6.1 Опис використаних методів тестування

У процесі розробки мною було обрано два основних методи тестування застосунку: тестування сумісності та тестування зручності використання. Ці методи забезпечили комплексну перевірку функціональності застосунку, а також його відповідність очікуванням користувачів.

Тестування сумісності спрямоване на перевірку роботи застосунку на різних пристроях та версіях операційної системи Android. Метою цього методу є забезпечення коректного функціонування застосунку незалежно від специфікацій пристрою, на якому він використовується.

Процес тестування сумісності включав такі етапи:

- **Вибір тестових пристроїв**:
  - Відібрано різні моделі смартфонів з різними версіями Android (від Android 11 до Android 14).
  - Враховано різноманітність технічних характеристик (розмір екрану, роздільна здатність, потужність процесора, об'єм оперативної пам'яті).
- **Інсталяція та запуск застосунку**:

- Перевірено процес інсталяції застосунку на кожному з обраних пристроїв.
- Проведено тестування первинного запуску застосунку для оцінки часу завантаження та відсутності помилок.

- **Перевірка основних функцій**:
  - Тестування можливості захоплення зображень екзаменаційних бланків камерою пристрою.
  - Оцінка коректності обробки та розпізнавання тестів на зображеннях.
  - Перевірка стабільності роботи застосунку при різних умовах освітлення та якості зображень.

Тестування зручності використання (Usability Testing) було спрямоване на оцінку інтерфейсу застосунку з точки зору кінцевих користувачів. Метою цього методу є забезпечення інтуїтивної зрозумілості та легкості використання застосунку, що підвищує задоволеність користувачів і сприяє більш широкому його прийняттю.

Процес тестування зручності використання включав такі етапи:

- **Визначення цільової аудиторії**:
  - Визначено основні категорії користувачів (вчителі, студенти, адміністратори освітніх установ).
- **Розробка сценаріїв використання**:
  - Створено типові сценарії використання застосунку, що включають реєстрацію, захоплення зображення бланку, обробку та отримання результатів.
  - Передбачено сценарії для користувачів з різним рівнем технічної підготовки.
- **Проведення тестових сесій**:

- Організовано тестові сесії з участю обраних представників цільової аудиторії.
- Кожен учасник виконував завдання за визначеними сценаріями, водночас спостерігали за їхніми діями та збирали зворотній зв'язок.

- **Аналіз результатів**:
  - Оцінено зручність інтерфейсу, зрозумілість навігації та швидкість виконання основних операцій.

3.6.2 Результати тестування

Результати тестування сумісності показали, що застосунок працює стабільно на більшості пристроїв і версій Android, що забезпечує його доступність для широкої аудиторії користувачів.

На пристроях, що мають невеликий об’єм оперативної пам’яті та слабкий процесор, робота програми може супроводжуватися сильними шумами на екрані, але це не заважає застосунку правильно та якісно оцінювати бланки.

Результати тестування зручності використання дозволили значно покращити інтерфейс та функціональність застосунку, забезпечивши його зручність і інтуїтивність для кінцевих користувачів.

ВИСНОВКИ

1. В роботі проведено аналіз існуючих методів та алгоритмів комп'ютерного зору, що застосовуються для розпізнавання та обробки зображень.
2. Розроблено архітектуру мобільного додатку, що включає всі необхідні компоненти для обробки зображень, розпізнавання зафарбованих кружечків та порівняння їх з еталонними відповідями.
3. Реалізовано додаток мовою Kotlin з використанням бібліотеки OpenCV, забезпечивши його коректну роботу на пристроях під керуванням Android.
4. Проведено тестування та оцінку ефективності роботи додатку на різних тестових бланках, аналізуючи точність розпізнавання та порівняння.
5. В результаті виконання дипломного проекту був розроблений мобільний додаток для пристроїв на базі Android, за допомогою якого користувачі можуть перевіряти та оцінювати тестові екзаменаційні бланки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Android Developers. [Електронний ресурс]. – https://developer.android.com/
2. Bubble sheet multiple choice scanner and test grader using OMR, Python, and OpenCV [Електронний ресурс]. – https://pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/
3. Griffiths, D., Phillips, B., & Smith, C. (2021). Head First Android Development, 3rd Edition.** O'Reilly Media.
4. Kaehler, A., & Bradski, G. (2017). Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library** (1st ed.). Packt Publishing Ltd.
5. Kotlin Teach. [Електронний ресурс]. – https://kotlinlang.org/education/
6. Laurence, P.-O., & Hinchman-Dominguez, A. (2019). Programming Android with Kotlin. Communications of the ACM
7. Laurence P.-O., Hinchman-Dominguez A. Programming Android with Kotlin. 1st Ed. // Communications of the ACM. - 2019.
8. OpenCV. [Електронний ресурс]. – https://opencv.org/
9. Optical mark recognition (OMR). Вікіпедія. [Електронний ресурс]. – https://en.wikipedia.org/wiki/Optical_mark_recognition

ДОДАТОК А

Лістинг програми:

```
package com.formscanner.detector

import org.opencv.core.CvType
import org.opencv.core.MatOfPoint
import org.opencv.core.MatOfPoint2f
import org.opencv.core.Point
import org.opencv.imgproc.Imgproc
import kotlin.math.abs

@JvmInline
internal value class Contour(private val mat: MatOfPoint) {
    val area: Double
        get() = abs(orientedArea)

    private val orientedArea: Double
        get() = Imgproc.contourArea(mat, true)

    fun detectCorners(): List<Point> {
        val contour2f = MatOfPoint2f()
        mat.convertTo(contour2f, CvType.CV_32F)

        val perimeter = Imgproc.arcLength(contour2f, true)
        val approximation = MatOfPoint2f()
        Imgproc.approxPolyDP(contour2f, approximation, perimeter * 0.04, true)
        val points = approximation.toList()

        val isCounterClockwise = orientedArea < 0
```

```
        return if (isCounterClockwise) points.reversed()
        else points
    }
}
package com.formscanner.detector

import org.opencv.core.Mat
import org.opencv.core.MatOfPoint
import org.opencv.imgproc.Imgproc

internal class ContourTree private constructor(
    private val contours: List<Contour>,
    private val contourHierarchy: Mat
) {
    fun asList() = contours

    fun getChildren(contour: Contour): List<Contour> {
        val contourIndex = contours.indexOf(contour)
        val childContourIndex = getRelativeIndex(contourIndex, Relative.CHILD)
        if (childContourIndex == -1) return emptyList()
        return getSiblings(contours[childContourIndex])
    }

    private fun getSiblings(contour: Contour): List<Contour> {
        val contourIndex = contours.indexOf(contour)
        return iterateContourIndices(contourIndex) {
            getRelativeIndex(it, Relative.NEXT_SIBLING)
        }.map { contours[it] }
```

```
}

private fun iterateContourIndices(
    startIndex: Int,
    computeNextIndex: (currentIndex: Int) -> Int
): List<Int> {
    require(startIndex != -1)

    val indices = mutableListOf<Int>()
    var currentIndex = startIndex
    while (currentIndex != -1) {
        indices.add(currentIndex)
        currentIndex = computeNextIndex(currentIndex)
    }
    return indices
}

private fun getRelativeIndex(contourIndex: Int, relative: Relative): Int =
    contourHierarchy.get(0, contourIndex)[relative.index].toInt()

private enum class Relative(val index: Int) {
    NEXT_SIBLING(0),
    PREV_SIBLING(1),
    CHILD(2),
    PARENT(3)
}
```

```
    companion object {
        fun createFrom(mat: Mat): ContourTree {
            val contours = mutableListOf<MatOfPoint>()
            val contourHierarchy = Mat()
            Imgproc.findContours(
                mat, contours, contourHierarchy,
                Imgproc.RETR_TREE, Imgproc.CHAIN_APPROX_SIMPLE
            )
            return ContourTree(
                contours.map { Contour(it) },
                contourHierarchy
            )
        }
    }
}
package com.formscanner.detector

import com.formscanner.FormMetrics

private const val MARGIN_FRACTION = 0.015
private const val PADDING_FRACTION = 0.15

data class DetectionArea<T : Number>(
    val x: T,
    val y: T,
    val width: T,
    val height: T,
    val horizontalPadding: T,
```

```
    val verticalPadding: T
) {
    companion object {
        fun calculate(frameWidth: Int, frameHeight: Int): DetectionArea<Double> {
            require(MARGIN_FRACTION in 0.0..50.0)
            require(PADDING_FRACTION in 0.0..50.0)

            val paddedWidth = frameWidth - frameWidth * MARGIN_FRACTION * 2.0
            val paddedHeight = frameHeight - frameHeight * MARGIN_FRACTION * 2.0

            val isVerticallyConstrained = paddedWidth / paddedHeight >
FormMetrics.RATIO

            return if (isVerticallyConstrained) DetectionArea(
                x = (frameWidth - paddedHeight * FormMetrics.RATIO) / 2.0,
                y = frameHeight * MARGIN_FRACTION,
                width = paddedHeight * FormMetrics.RATIO,
                height = paddedHeight,
                verticalPadding = paddedHeight * PADDING_FRACTION,
                horizontalPadding = paddedWidth * PADDING_FRACTION
            ) else DetectionArea(
                x = frameWidth * MARGIN_FRACTION,
                y = (frameHeight - paddedWidth / FormMetrics.RATIO) / 2.0,
                width = paddedWidth,
                height = paddedWidth / FormMetrics.RATIO,
                verticalPadding = paddedHeight * PADDING_FRACTION,
                horizontalPadding = paddedWidth * PADDING_FRACTION
            )
```

```
        }
    }
}
package com.formscanner.detector

import kotlin.math.ceil

fun DetectionArea<Double>.asInt() =
    DetectionArea(
        ceil(x).toInt(),
        ceil(y).toInt(),
        ceil(width).toInt(),
        ceil(height).toInt(),
        ceil(horizontalPadding).toInt(),
        ceil(verticalPadding).toInt()
    )

fun DetectionArea<Double>.asFloat() =
    DetectionArea(
        x.toFloat(),
        y.toFloat(),
        width.toFloat(),
        height.toFloat(),
        horizontalPadding.toFloat(),
        verticalPadding.toFloat()
    )

val DetectionArea<Double>.paddedWidth
```

```
    get() = width - horizontalPadding * 2.0

val DetectionArea<Double>.paddedHeight
    get() = height - verticalPadding * 2.0

val DetectionArea<Double>.area
    get() = width * height

val DetectionArea<Double>.paddedArea
    get() = paddedWidth * paddedHeight



val DetectionArea<Float>.x2
    get() = x + width

val DetectionArea<Float>.y2
    get() = y + height



val DetectionArea<Int>.x2
    get() = x + width

val DetectionArea<Int>.y2
    get() = y + height
package com.formscanner.detector

import com.formscanner.FormMetrics
import org.opencv.core.Mat
```

```
import org.opencv.core.Point
import java.util.Collections
import kotlin.math.abs
import kotlin.math.pow
import kotlin.math.sqrt

private const val FORM_CORNER_COUNT = 4
private const val TRIANGLE_CORNER_COUNT = 3
private const val TRIANGLE_AREA_TOLERANCE = 0.40
private const val TRIANGLE_TO_CORNER_MAXIMUM_DISTANCE = 10.0

internal object Detector {
    fun detect(frame: Mat): FormPosition? {
        val detectionArea = DetectionArea.calculate(
            frame.width(), frame.height()
        )
        val contourTree = ContourTree.createFrom(frame)

        val formCorners = contourTree.asList()
            .firstNotNullOfOrNull { contour ->
                contour.asFormCornersOrNull(contourTree, detectionArea)
            } ?: return null
        return FormPosition(*formCorners.toTypedArray())
    }

    private fun Contour.asFormCornersOrNull(
        contourTree: ContourTree,
```

```
        detectionArea: DetectionArea<Double>
    ): List<Point>? {
        val isOutsideDetectionArea =
            !(area > detectionArea.paddedArea && area < detectionArea.area)
        if (isOutsideDetectionArea) return null

        val boxCorners = detectCorners()
        if (boxCorners.size != FORM_CORNER_COUNT) return null

        val triangleCorners = contourTree.getChildren(this)
            .firstNotNullOfOrNull { contour ->
                contour.asTriangleCornersOrNull(detectionArea)
            } ?: return null

        val (cornerWithTriangle, distance) =
            boxCorners.closestToTriangle(triangleCorners)

        val isTriangleMisplaced = distance >
TRIANGLE_TO_CORNER_MAXIMUM_DISTANCE
        if (isTriangleMisplaced) return null

        return correctFormOrientation(boxCorners, cornerWithTriangle)
    }

    private fun Contour.asTriangleCornersOrNull(
        detectionArea: DetectionArea<Double>
    ): List<Point>? {
        val isWrongSize = !isRightSizeTriangle(this, detectionArea)
```

```
        if (isWrongSize) return null
        val corners = detectCorners()
        return if (corners.size == TRIANGLE_CORNER_COUNT) corners
        else null
    }

    private fun isRightSizeTriangle(
        contour: Contour,
        detectionArea: DetectionArea<Double>
    ): Boolean {
        val expectedArea = detectionArea.area *
FormMetrics.Triangle.TO_FORM_AREA_RATIO
        val areaDifference = abs(contour.area - expectedArea)
        return areaDifference < expectedArea * TRIANGLE_AREA_TOLERANCE
    }

    private fun List<Point>.closestToTriangle(
        triangleCorners: List<Point>
    ): Pair<Point, Double> =
        flatMap { qPoint ->
            triangleCorners.map { tPoint ->
                qPoint to tPoint.distance(qPoint)
            }
        }.minBy { it.second }

    private fun Point.distance(other: Point): Double =
        sqrt((x - other.x).pow(2) + (y - other.y).pow(2))
```

```
    private fun correctFormOrientation(
        boxCorners: List<Point>,
        closestCorner: Point
    ): List<Point> {
        val oriented = boxCorners.toList()
        Collections.rotate(oriented, -oriented.indexOf(closestCorner))
        return oriented
    }
}
package com.formscanner.detector

import org.opencv.core.Point
import org.opencv.core.Rect

data class FormPosition(
    val topLeft: Point,
    val topRight: Point,
    val bottomRight: Point,
    val bottomLeft: Point
) {
    val corners: List<Point> =
        listOf(topLeft, topRight, bottomRight, bottomLeft)

    constructor(vararg corners: Point) : this(corners[0], corners[1], corners[2], corners[3])
{
        require(corners.size == 4) { "The number of points should be strictly 4." }
    }
```

```
    fun getBoundingBox(): Rect {
        val x1 = corners.minOf { it.x }.toInt()
        val y1 = corners.minOf { it.y }.toInt()
        val x2 = corners.maxOf { it.x }.toInt()
        val y2 = corners.maxOf { it.y }.toInt()
        return Rect(x1, y1, x2 - x1, y2 - y1)
    }

    companion object {
        val Empty = FormPosition(Point(), Point(), Point(), Point())
    }
}
package com.formscanner

import com.formscanner.FormMetrics.Option
import org.opencv.core.Core
import org.opencv.core.Mat
import org.opencv.core.Rect
import kotlin.math.PI
import kotlin.math.pow

private const val OPTION_CHECKED_TOLERANCE = 0.3
private const val OPTION_DETECTION_EXTRA_SCALE = 0.3

internal object Evaluator {
    fun evaluate(form: Mat): Form {
        val formMetrics = FormMetrics.fromActual(form.width())
```

```
    val answers = formMetrics.blocks.flatMap { block ->
        block.optionGroups.map { optionGroup ->
            resolveAnswer(form, optionGroup)
        }
    }
    return Form(answers)
}

private fun resolveAnswer(
    form: Mat,
    optionGroup: FormMetrics.OptionGroup
): Form.Answer {
    val checkedOptionNumbers = optionGroup.options
        .withIndex()
        .filter { isOptionChecked(form, it.value) }
        .map { it.index + 1 }

    return when (checkedOptionNumbers.size) {
        0 -> Form.Answer.Empty
        1 -> Form.Answer.Option(checkedOptionNumbers.single())
        else -> Form.Answer.Ambiguous
    }
}

private fun isOptionChecked(
    form: Mat,
    option: Option
```

```
    ): Boolean {
        val extraSize = option.size * OPTION_DETECTION_EXTRA_SCALE
        val optionBoundary = Rect(
            (option.position.x - extraSize).toInt(),
            (option.position.y - extraSize).toInt(),
            (option.size + extraSize * 2.0).toInt(),
            (option.size + extraSize * 2.0).toInt()
        )
        val nonZeroArea = form.submat(optionBoundary) { optionMat ->
            Core.countNonZero(optionMat)
        }
        val optionArea = PI * (option.size / 2.0).pow(2)
        val areaDifference = optionArea - nonZeroArea
        return areaDifference < optionArea * OPTION_CHECKED_TOLERANCE
    }
}
package com.formscanner

data class Form(
    val answers: List<Answer> = emptyList()
) {
    companion object {
        val Empty = Form()
    }

    sealed interface Answer {
        data class Option(val number: Int) : Answer
        data object Ambiguous : Answer
```

```
        data object Empty : Answer
    }
}
package com.formscanner

import com.formscanner.BuildConfig.BLOCK_COUNT
import com.formscanner.BuildConfig.BLOCK_GROUP_OFFSET_X
import com.formscanner.BuildConfig.BLOCK_GROUP_OFFSET_Y
import com.formscanner.BuildConfig.BLOCK_OFFSET
import com.formscanner.BuildConfig.FORM_HEIGHT
import com.formscanner.BuildConfig.FORM_WIDTH
import com.formscanner.BuildConfig.GRADE_BOX_HEIGHT
import com.formscanner.BuildConfig.GRADE_BOX_OFFSET_X
import com.formscanner.BuildConfig.GRADE_BOX_OFFSET_Y
import com.formscanner.BuildConfig.GRADE_BOX_WIDTH
import com.formscanner.BuildConfig.OPTION_COUNT
import com.formscanner.BuildConfig.OPTION_GROUP_COUNT
import com.formscanner.BuildConfig.OPTION_GROUP_OFFSET
import com.formscanner.BuildConfig.OPTION_OFFSET
import com.formscanner.BuildConfig.OPTION_SIZE
import com.formscanner.BuildConfig.TRIANGLE_LEG_LENGTH

class FormMetrics private constructor(
    actualWidth: Int
) {
    private val scale = actualWidth / FORM_WIDTH.toDouble()
    private val blockCount = BLOCK_COUNT
    private val optionGroupCount = OPTION_GROUP_COUNT
```

```
private val optionCount = OPTION_COUNT
private val blockOffset = BLOCK_OFFSET to scale
private val blockGroupOffsetX = BLOCK_GROUP_OFFSET_X to scale
private val blockGroupOffsetY = BLOCK_GROUP_OFFSET_Y to scale
private val optionGroupOffset = OPTION_GROUP_OFFSET to scale
private val optionOffset = OPTION_OFFSET to scale
private val optionSize = OPTION_SIZE to scale
private val gradeBoxOffsetX = GRADE_BOX_OFFSET_X to scale
private val gradeBoxOffsetY = GRADE_BOX_OFFSET_Y to scale
private val gradeBoxWidth = GRADE_BOX_WIDTH to scale
private val gradeBoxHeight = GRADE_BOX_HEIGHT to scale

val blocks by lazy { createBlocks() }
val gradeBox by lazy { createGradeBox() }

private infix fun Int.to(scale: Double): Double = this * scale

private fun createBlocks(): List<Block> =
    (0..<blockCount)
        .map { index -> index * blockOffset }
        .map { horizontalOffset ->
            val position = Position(
                blockGroupOffsetX + horizontalOffset,
                blockGroupOffsetY,
            )
            val optionGroups = createOptionGroups(position)
            Block(position, optionGroups)
```

```
        }

private fun createOptionGroups(
    parentPosition: Position
): List<OptionGroup> =
    (0..<optionGroupCount)
        .map { index -> index * optionGroupOffset }
        .map { verticalOffset ->
            val position = Position(
                parentPosition.x,
                parentPosition.y + verticalOffset
            )
            val options = createOptions(position)
            OptionGroup(position, options)
        }

private fun createOptions(
    parentPosition: Position
): List<Option> =
    (0..<optionCount)
        .map { index -> index * optionOffset }
        .map { horizontalOffset ->
            val position = Position(
                parentPosition.x + horizontalOffset,
                parentPosition.y
            )
            Option(position, optionSize)
```

```
        }

    private fun createGradeBox(): GradeBox {
        val position = Position(
            gradeBoxOffsetX,
            gradeBoxOffsetY
        )
        return GradeBox(position, gradeBoxWidth.toInt(), gradeBoxHeight.toInt())
    }

    companion object {
        const val RATIO = FORM_WIDTH / FORM_HEIGHT.toDouble()
        const val AREA = FORM_WIDTH * FORM_HEIGHT

        fun fromActual(width: Int): FormMetrics = FormMetrics(width)
    }

    object Triangle {
        private const val AREA = TRIANGLE_LEG_LENGTH *
TRIANGLE_LEG_LENGTH / 2.0
        const val TO_FORM_AREA_RATIO = AREA / FormMetrics.AREA
    }

    data class Block(
        val position: Position,
        val optionGroups: List<OptionGroup>
    )
```

```
    data class OptionGroup(
        val position: Position,
        val options: List<Option>
    )

    data class Option(
        val position: Position,
        val size: Double
    )

    data class GradeBox(
        val position: Position,
        val width: Int, val height: Int
    )

    data class Position(
        val x: Double, val y: Double
    )
}
package com.formscanner

import org.opencv.core.Mat
import org.opencv.core.Size
import org.opencv.imgproc.Imgproc

private const val CLEAN_KERNEL_SIZE = 5.0
```

```
internal object FormPreprocessor {
    fun preprocess(form: Mat) = with(form) {
        removeEmptyOptions()
    }

    private fun Mat.removeEmptyOptions() {
        val kernel = Imgproc.getStructuringElement(
            Imgproc.MORPH_RECT,
            Size(CLEAN_KERNEL_SIZE, CLEAN_KERNEL_SIZE)
        )
        Imgproc.erode(this, this, kernel)
        Imgproc.dilate(this, this, kernel)
    }
}
package com.formscanner

import com.formscanner.detector.DetectionArea
import com.formscanner.detector.asInt
import com.formscanner.detector.x2
import com.formscanner.detector.y2
import org.opencv.core.Mat
import org.opencv.core.Rect
import org.opencv.core.Scalar
import org.opencv.core.Size
import org.opencv.imgproc.Imgproc
import kotlin.math.max

private const val SCALING_MINIMUM_FRAME_SIDE = 500.0
```

```
private const val THRESHOLD_MAX_VALUE = 255.0
private const val THRESHOLD_BLOCK_SIZE = 25
private const val THRESHOLD_CONSTANT = 50.0

internal object FramePreprocessor {
    fun preprocess(frame: Mat): PreprocessingResult = with(frame) {
        val scaleFactor = scale()
        toGrayscale()
        toBlackAndWhite()
        fillBeyondDetectionArea()

        PreprocessingResult(scaleFactor = scaleFactor)
    }

    private fun Mat.scale(): Double {
        val horizontalScale = SCALING_MINIMUM_FRAME_SIDE / width()
        val verticalScale = SCALING_MINIMUM_FRAME_SIDE / height()
        val scale = max(horizontalScale, verticalScale)
        Imgproc.resize(this, this, Size(), scale, scale)
        return scale
    }

    private fun Mat.toGrayscale() {
        Imgproc.cvtColor(this, this, Imgproc.COLOR_RGB2GRAY)
    }

    private fun Mat.toBlackAndWhite() {
```

```
        Imgproc.adaptiveThreshold(
            this, this,
            THRESHOLD_MAX_VALUE,
            Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C,
            Imgproc.THRESH_BINARY_INV,
            THRESHOLD_BLOCK_SIZE, THRESHOLD_CONSTANT
        )
    }

    private fun Mat.fillBeyondDetectionArea() {
        val detectionArea = DetectionArea.calculate(width(), height()).asInt()
        val sections = with(detectionArea) {
            listOf(
                Rect(0, 0, width(), y),
                Rect(0, y2, width(), y),
                Rect(0, 0, x, height()),
                Rect(x2, 0, x, height())
            )
        }
        val blackColor = Scalar(0.0, 0.0, 0.0, 0.0)
        for (section in sections) {
            Imgproc.rectangle(this, section, blackColor, -1)
        }
    }
}
package com.formscanner

import org.opencv.core.Mat
```

```
import org.opencv.core.Rect
import kotlin.contracts.ExperimentalContracts
import kotlin.contracts.InvocationKind
import kotlin.contracts.contract

@OptIn(ExperimentalContracts::class)
fun <T> Mat.submat(
    rect: Rect,
    transform: (submat: Mat) -> T
): T {
    contract {
        callsInPlace(transform, InvocationKind.EXACTLY_ONCE)
    }

    val submat = submat(rect)
    val result = transform(submat)
    submat.release()
    return result
}

@OptIn(ExperimentalContracts::class)
fun <T> Mat.copy(block: (mat: Mat) -> T): T {
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }

    val mat = Mat()
```

```
    copyTo(mat)
    val result = block(mat)
    mat.release()
    return result
}

@OptIn(ExperimentalContracts::class)
fun <T> temporaryMat(block: (mat: Mat) -> T): T {
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }

    val mat = Mat()
    val result = block(mat)
    mat.release()
    return result
}
package com.formscanner

internal data class PreprocessingResult(
    val scaleFactor: Double
)
package com.formscanner

import com.formscanner.detector.FormPosition

data class ScanResult(
    val form: Form = Form.Empty,
```

```
    val formPosition: FormPosition = FormPosition.Empty
) {
    companion object {
        val Empty = ScanResult()
    }
}
package com.formscanner

import com.formscanner.detector.Detector
import com.formscanner.detector.FormPosition
import org.opencv.core.Mat
import org.opencv.core.MatOfPoint2f
import org.opencv.core.Point
import org.opencv.core.Size
import org.opencv.imgproc.Imgproc

object Scanner {
    fun scan(frame: Mat): ScanResult? {
        val preprocessingResult = FramePreprocessor.preprocess(frame)
        val formPosition = Detector.detect(frame) ?: return null
        val form = temporaryMat { form ->
            prepareForm(frame, form, formPosition)
            Evaluator.evaluate(form)
        }
        val originalFormPosition = formPosition
            .toOriginalScale(preprocessingResult.scaleFactor)
        return ScanResult(form, originalFormPosition)
    }
```

```
    private fun prepareForm(
        frame: Mat, form: Mat,
        formPosition: FormPosition
    ) {
        val formSize = calculateFormSize(formPosition)
        form.create(formSize, frame.type())
        correctPerspective(frame, form, formPosition)
        FormPreprocessor.preprocess(form)
    }

    private fun calculateFormSize(formPosition: FormPosition): Size {
        val formBoundingBox = formPosition.getBoundingBox()
        val isVerticallyConstrained =
            formBoundingBox.width / formBoundingBox.height.toDouble() >
FormMetrics.RATIO

        val width: Double
        val height: Double
        if (isVerticallyConstrained) {
            width = formBoundingBox.width.toDouble()
            height = formBoundingBox.width / FormMetrics.RATIO
        } else {
            width = formBoundingBox.height * FormMetrics.RATIO
            height = formBoundingBox.height.toDouble()
        }
        return Size(width, height)
    }
```

```
    private fun correctPerspective(
        frame: Mat, form: Mat,
        formPosition: FormPosition
    ) {
        val sourceCoordinates = MatOfPoint2f(*formPosition.corners.toTypedArray())
        val destinationCoordinates = MatOfPoint2f(
            Point(0.0, 0.0),
            Point(form.width().toDouble(), 0.0),
            Point(form.width().toDouble(), form.height().toDouble()),
            Point(0.0, form.height().toDouble())
        )
        val transformation = Imgproc.getPerspectiveTransform(
            sourceCoordinates, destinationCoordinates
        )
        Imgproc.warpPerspective(frame, form, transformation, form.size())
    }

    private fun FormPosition.toOriginalScale(scaleFactor: Double): FormPosition {
        val scaledCorners = corners.map {
            Point(it.x / scaleFactor, it.y / scaleFactor)
        }
        return FormPosition(*scaledCorners.toTypedArray())
    }
}
```

ДОДАТОК Б

Демонстраційні матеріали

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Розробка Android-застосунку для автоматизації перевірки тестувальних бланків у навчальних процесах

Виконав студент 4 курсу, групи КНД-41 Колосовський А.О.

Керівник: д.т.н., професор Ільїн О.О.

Київ 2024

Проблема

Перевірка великої кількості тестувальних бланків вручну є трудомістким, часозатратним і схильним до людських помилок процесом, що призводить до неефективності, упередженості в оцінюванні та затримок у наданні зворотного зв'язку студентам.

Вирішення проблеми

Вирішенням цієї проблеми може стати мобільний додаток на платформі Android, який за допомогою камери буде здатний автоматично розпізнавати та порівнювати заповнені тестові бланки з еталонними відповідями, оцінюючи результати за 100-бальною шкалою, тим самим заощаджувати час та сили перевіряючої особи.

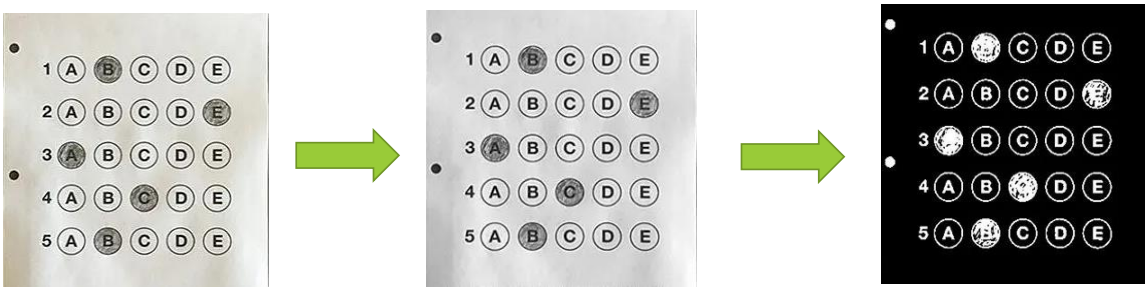
Основні завдання роботи

- Проаналізувати існуючі алгоритми та методи комп'ютерного зору, що застосовуються для обробки зображень
- Розробити архітектуру мобільного додатку, що включає всі необхідні компоненти для обробки зображень.
- Реалізувати додаток мовою програмування Kotlin з використанням бібліотеки OpenCV

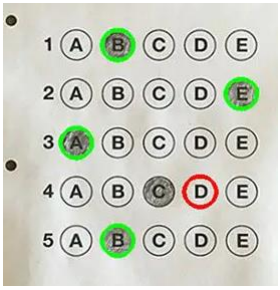
Комп'ютерний зір

Комп'ютерний зір – це галузь штучного інтелекту, яка займається автоматичним отриманням, обробкою та аналізом візуальної інформації з реального світу. Основною метою комп'ютерного зору є створення систем, які можуть розпізнавати та розуміти візуальні об'єкти так само, як це робить людське око. Комп'ютерний зір знаходить застосування в різних сферах, таких як медицина, робототехніка, автомобільна промисловість, безпека, а також в аналізі зображень та відео.

Алгоритм розпізнавання



Логіка трекінгу правильних відповідей



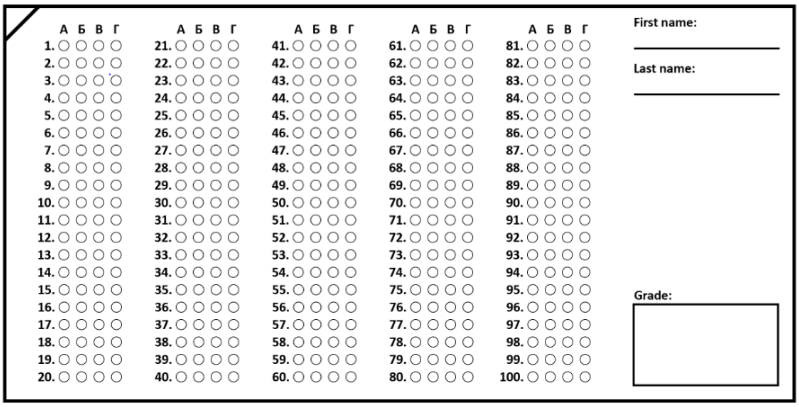
Інструменти розробки

android studio

OpenCV

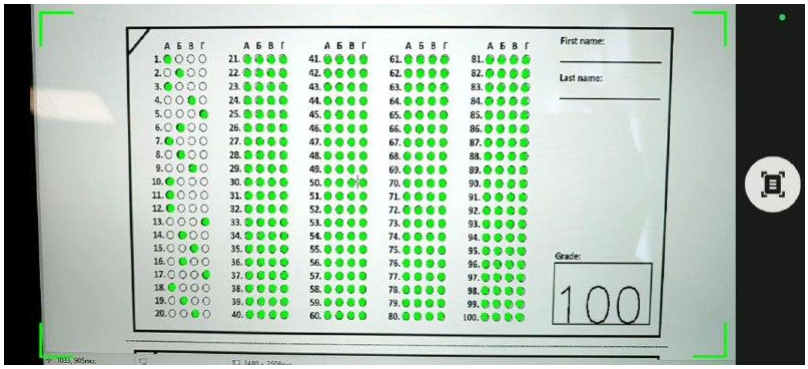
Kotlin

Формат бланку



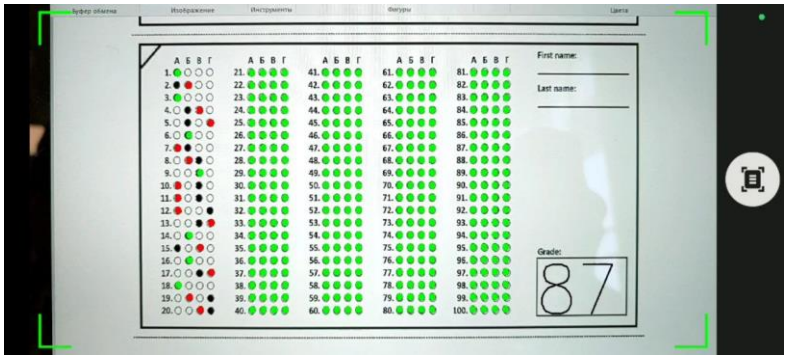
Експлуатація програми. Крок 1

Користувач має навести камеру на бланк. Коли білі кути змінять свій колір на зелений(це буде означати, що форма розпізнана), натиснути кнопку, що буде означати фіксацію правильних відповідей для подальших порівнянь.



Експлуатація програми. Крок 2

Користувач має навести камеру на бланк, який хоче перевірити так, щоб кути знову стали зеленими. Програма автоматично зафіксує обрані відповіді в прямому відеопотоці, та підсвітить правильні та неправильні відповіді. Оцінка знаходиться у прямокутній формі правого нижнього кута.



Висновки

- В цьому дослідженні представлено розробку мобільного застосунку на базі Android, метою створення якого є перевірка бланків з тестами
- Проведено дослідження особливостей алгоритмів оброки та аналізу зображень методами комп’ютерного зору
- Реалізовано додаток мовою Kotlin з використанням бібліотеки OpenCV, забезпечивши його коректну роботу на пристроях під керуванням Android
- Проведено тестування та оцінку ефективності роботи додатку на тестових бланках, аналізуючи точність розпізнавання та порівняння