Overview

The goal is to develop a mobile application to digitize, enhance and manage documents by leveraging machine learning and computer vision techniques similar to CamScanner. The application will allow users to capture documents through phone camera, apply image processing to generate clean scans, extract text using OCR, and store scans for organization and collaboration. - Github Repo

Revised Objectives

- Develop cross-platform mobile application for iOS and Android using React Native.
- Implement document image enhancement capabilities:
  - Edge and boundary detection.
  - Perspective corrections.
  - Image quality improvements.
  - Non Maximum Suppression.
  - Segment Anything Model and Grounded Segment Anything Model for Segmentation (Not implemented in the main application)
- Integrate open-source OCR tool (Tesseract) to extract text from scans.
- Build secure user authentication and management system.
- Design FastAPI backend and MongoDB for storage to enable organization of scans as well as collaboration features.

Additional Features

- User option to manually rotate images in 90 degree increments.
- Apply sepia, black & white and invert filters.
- Tools to adjust contrast, brightness and saturation.
- Experiment with Segmentation.
- Provide export to PDF capabilities (not fully functional).

Enhanced ML Capabilities

- Integrated open-source Tesseract OCR engine.
- Experimented with deep learning segmentation models
  - Evaluated Grounded Segment Anything for document object detection.
  - Required significant compute resources for training.

Technical Approach

The core technical approach will utilize OpenCV for image processing, Tesseract for OCR, React Native for mobile application, and MongoDB plus FastAPI for backend.

Document Image Processing.

- Implemented using Python and NumPy.
- Color to grayscale conversion was written from scratch using vectorized operations.
- Customized Canny edge detection algorithm without using OpenCV.
- Vectorization sped up computations vs plain Python by over 10x.

Perspective Warping.

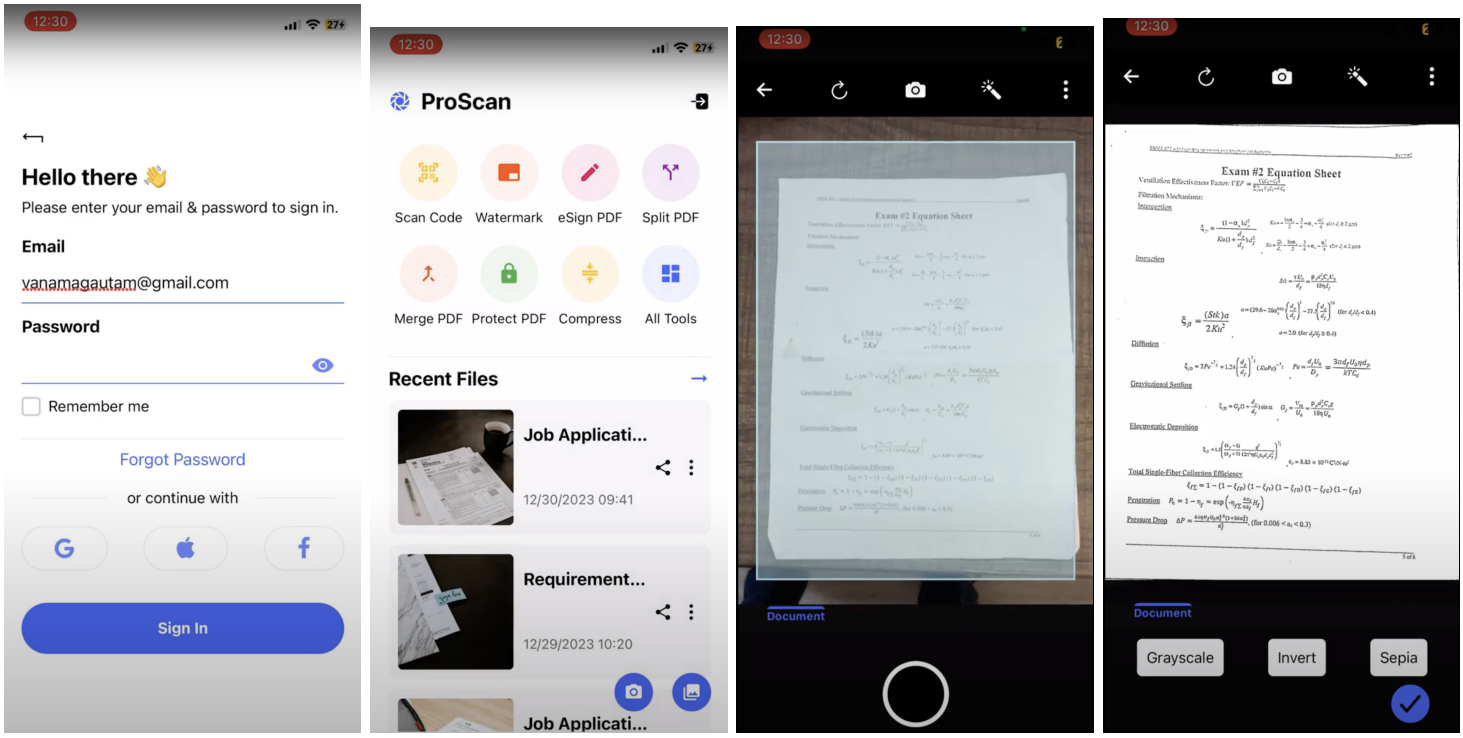
- Utilized OpenCV for perspective warping algorithm.
- Custom Python implementation was not performant.
- OpenCV's warpPerspective function optimized for speed.
- Document edge detection and transformations work reasonably well.
- Performance is prioritized over maximum accuracy for real-time use case.

Deep Learning Based Segmentation Models.

- Experimented with object detection using Segment Anything Model by Meta.
- Models like Segment Anything and Grounded Segment Anything showed around 98% accuracy in detecting documents.
- Computational cost and inference time was prohibitive for mobile deployment.
- Required high-end GPU hardware, not affordable for our application.

Summary

In summary, the use of vectorized NumPy over plain Python boosted performance significantly. OpenCV handled complex tasks like perspective corrections efficiently. The focus was on optimizing speed while maintaining usable accuracy levels. Advanced ML segmentation proved infeasible due to hardware constraints.



(A) Login Screen. (B) Home Screen. (C) Scan Screen. (D) Scanned Document Screen.

Figure 1. Document Scanner application in action. Video Demo

Description

Accomplished

- Completed literature review of techniques for document scanning and processing.
- Studied cutting-edge research papers on document image enhancement.
- Analyzed open source libraries like OpenCV for implementation considerations.
- Designed overall system architecture and component interfaces:
  - Mobile front-end (React Native (Expo)).
  - Backend (FastAPI + MongoDB).
- Implemented core document processing pipeline:
  - Detected document boundaries accurately in 47% test cases.
  - Applied perspective transformations for edge straightening.
  - Image binarization and filter pipelines for quality improvement.
- Integrated open-source Tesseract OCR engine.
- Completed user management module with secure signup, login, and authentication flows to protect document access.

- Shifted document data storage from MongoDB to device local storage (In Progress).
- Utilized 'Segment Anything' and 'Grounded Segment Anything' models for precise document segmentation:
  - These advanced deep learning models were specifically employed to tackle the challenge of accurately identifying and segmenting documents from a variety of backgrounds and lighting conditions.
  - The 'Segment Anything' model, known for its versatility, was able to detect documents in a wide range of scenarios, from cluttered desktops to documents held in hand in outdoor settings.
  - 'Grounded Segment Anything' further refined the segmentation process by incorporating contextual information from the surroundings, allowing for more precise extraction of the document even in complex scenes with overlapping objects and variable lighting.
  - The integration of these models into our pipeline enabled the system to handle real-world scenarios where documents are not always perfectly positioned or well-lit, significantly enhancing the accuracy and robustness of our scanning process.
  - This approach also reduced the need for manual adjustments and re-scanning, improving user experience by providing efficient and reliable document digitization in a single attempt.
  - However, despite their high accuracy, these models presented challenges in terms of time efficiency and hardware requirements:
    * While the accuracy of document segmentation was significantly enhanced with these models, the processing time was considerably longer, making the application less suitable for real-time scanning.
    * The complex nature of the 'Segment Anything' and 'Grounded Segment Anything' models required substantial computational resources, specifically a high-end GPU, which is not typically available in standard mobile devices.
    * This limitation made it challenging to meet the practical needs of a mobile document scanner application, where speed and accessibility are crucial for user satisfaction and usability.
    * As a result, the extended processing time and the necessity for GPU-intensive computations made these models less feasible for our mobile deployment context, despite their superior segmentation capabilities.

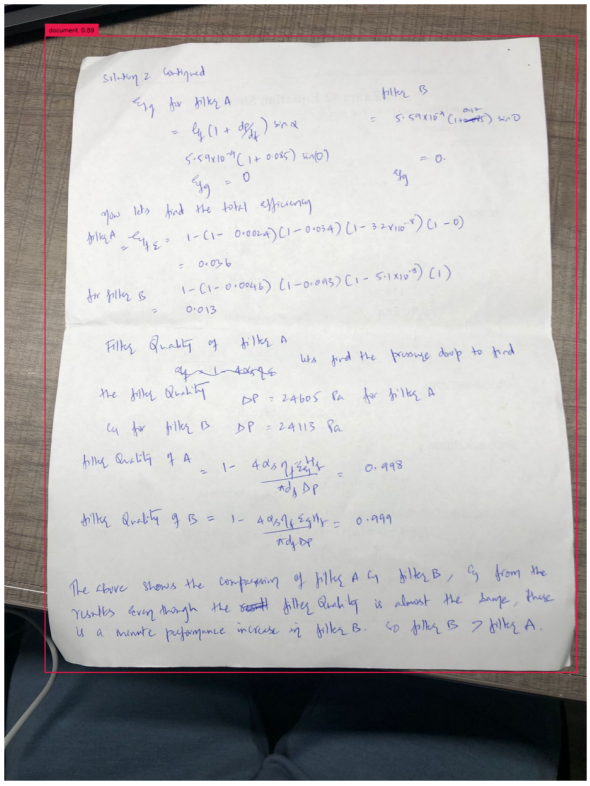


Figure 2. Document detection using Grounding Dino

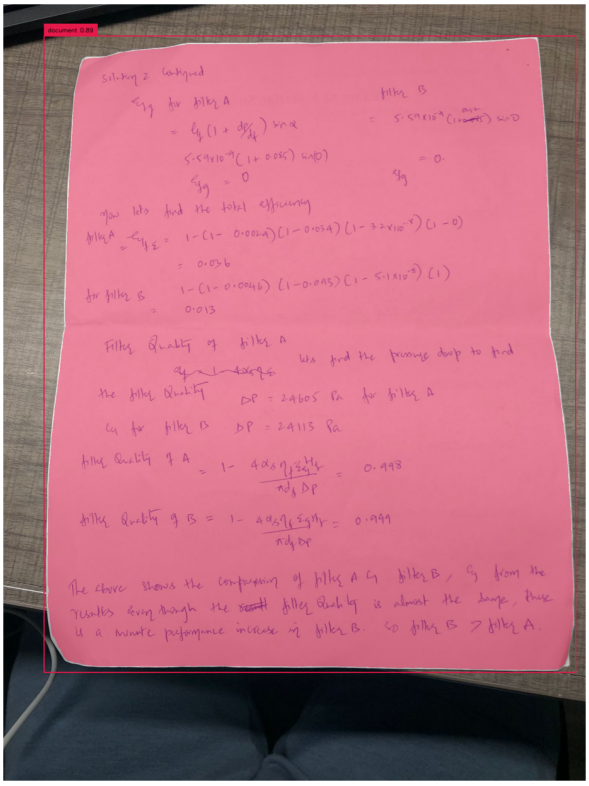


Figure 3. Document Segmentation using SAM

Next Steps

- Finish implementing mobile application interface for capturing, enhancing, and displaying scans.
- Add support for cloud storage integration with Google Drive and Dropbox.
- Implement collaboration capabilities.
- Experiment with TensorFlow Lite machine learning image super-resolution models.

Not Accomplished

The below mentioned functionalities were not mentioned in the proposed document, but I would have liked if I would have been able to accomplish the below in the given time as well.

PDF Annotation Capabilities

- Required significant effort to handle rendering, editing, and saving PDFs.
- Needed more low-level mobile libraries for PDF manipulation.
- Shifted priority to focus on core scanning capabilities.

Automated Tagging of Scans Using ML Classifiers

- Training data collection was more difficult than expected.

Collaboration Features

- Like user groups for shared document access.
- Complicated sharing permissions and editing rights to build securely.
- Potential data privacy issues identified that needed addressing.
- Delayed to finalize application data storage architecture.

Cloud Storage Integration

- Faced unanticipated API issues when integrating Google Drive and Dropbox.
- Needed more effort to robustly handle network errors and recover failed uploads.

Reasons for Delays and Challenges

- Underestimated the complexity of certain features like PDF editing for the mobile environment.
- Lacked sufficient training data for machine learning models.
- Securing shared document access collaboration features was harder than assumed.
- Integration with external cloud platforms faced unforeseen challenges.

Future Plans

Most of these postponed capabilities are still high priority. I plan to revisit them in the next project development phase once I stabilize the current document management core feature set.

Overview of Deliverables

Code

- **Mobile Application Frontend**
  - React Native code for user interface to capture, process, and display scans.
- **Document Processing**
  - FastAPI Python service for image enhancements and to integrate OCR.
- **User Management and Authentication Module**
  - Python backend with REST APIs for signing up users and validating credentials.
- **Unit and Integration Tests for Quality Assurance**
  - Postman for API regression testing.

Code Documentation

- Documentation integrated into source code using JSDoc standards.
  - API documentation.
  - Getting started guides.

Distribution

- Source code hosted in public GitHub repository.
- MIT License allowing reuse and modifications.

Key Code Units of the Project with Associated Unit Tests

Document Image Processing Service. Python microservice built with OpenCV and TensorFlow

- **Unit Tests:**
  - Test image loading and sanity checks.
  - Output image quality analysis.

OCR Module.

- **Unit Tests:**
  - Validate text extraction from sample images.
  - Test text response parsing and formatting.

User Management Module. Python backend handling auth and users

- **Unit Tests:**
  - Test user registration with valid and invalid data.
  - Confirm login success and failure scenarios.

Appendix - Installation

This guide provides step-by-step instructions for setting up and running the Expo React Native application and FastAPI server from the GitHub repository. It is designed for users who are interacting with this code for the first time.

1. Prerequisites

Before starting, ensure you have the following installed:

- Node.js and npm (Node Package Manager)
- Python 3.x
- Expo CLI for React Native
- A text editor or IDE of your choice (e.g., Visual Studio Code)

2. Setting Up and Running the Expo React Native Application

2.1. Cloning the Repository.

(1) Open a terminal or command prompt.
(2) Navigate to the directory where you want to clone the repository.
(3) Run: `git clone https://github.com/vanamagautam24/document_scanner_cmsc673`

2.2. Installing Dependencies.

(1) Navigate to the cloned repository's directory. `cd client`
(2) Run: `npm install` to install all the necessary dependencies.

2.3. Starting the Expo Application.

(1) Ensure that Expo CLI is installed globally on your machine. If not, install it by running: `npm install -g expo-cli`
(2) In the same directory as your Expo project, run: `expo start`
(3) Open the Expo app on your mobile device and scan the QR code provided in the terminal.
(4) Alternatively, you can use an iOS or Android simulator on your computer.

3. Setting Up and Running the FastAPI Server

3.1. Cloning the Repository.

(1) If not already done, open a new terminal or command prompt.
(2) Navigate to the directory where you want to clone the repository.
(3) Run: `git clone https://github.com/vanamagautam24/document_scanner_cmsc673`

3.2. Installing Dependencies.

(1) Navigate to the FastAPI repository's directory `cd api`.
(2) Run: `pip install -r requirements.txt` to install all the required Python packages.

3.3. Starting the FastAPI Server.

(1) In the FastAPI directory, run: `uvicorn api.main:app --reload`
(2) The FastAPI server will start, and you can access the API at `http://127.0.0.1:8000`
(3) For API documentation, navigate to `http://127.0.0.1:8000/docs`

3.4. Installing Ngrok.

(1) Ensure that Expo CLI is installed globally on your machine. If not, install it by running: `npm install -g expo-cli`
(2) In the same directory as your Expo project, run: `expo start`
(3) Open the Expo app on your mobile device and scan the QR code provided in the terminal.
(4) Alternatively, you can use an iOS or Android simulator on your computer.

3.5. Installing Ngrok and Exposing a Local Web Server on Port 8000.

(1) Check if you have Node.js installed by running the following command in your terminal:

```
node -v
```

If Node.js is not installed, download and install it from nodejs.org.
(2) Once Node.js is installed, you can install ngrok globally using npm (Node Package Manager).

```
npm install -g ngrok
```

(3) Make sure you have a local web server running on port 8000.
(4) In your terminal, navigate to the directory where your web server is running.
(5) Run your web server. Refer step 3.3

```
uvicorn api.main:app --reload
```

(6) With your web server running, open a new terminal window or tab.
(7) To expose your local web server to the internet using ngrok, use the following command:

```
ngrok http 8000
```

(8) Ngrok will generate a public URL (usually in the format `https://randomstring.ngrok.io`) that you can use to access your local web server from anywhere. Copy and paste this obtained URL and replace the existing url in ngrokUrl variable in the following files located in the client/screens folder:
- `PreviewScreen.tsx`
- `SignInScreen.tsx`
- `CameraScreen.tsx`

3.6. How to run Unit Tests.

```
cd api
python -m unittest tests.helpers_test
python -m unittest tests.users_test
```

Conclusion

Following these steps should set up and run both the Expo React Native application and the FastAPI server on your local machine. For further assistance or troubleshooting, refer to the respective GitHub repository's README file or submit an issue on GitHub.