DEEP SCAN

PROJECT REPORT

Submitted by

ABITHA M	**(714023104002)**
ANANDHI A	**(714023104006)**
DSHANTHINI R	**(714023104024)**
KAVITHA K	**(714023104048)**

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING IN

COMPUTER SCIENCE AND ENGINEERING

SRI SHAKTHI

INSTITUTE OF ENGINEERING ANDTECHNOLOGY.

An Autonomous Institution

Accredited by NAAC with "A"Grade

MAY 2025

BONAFIDE CERTIFICATE

Certified that this Report titled "**DEEP SCAN**" is the bonafide work of **"ABITHA M (714023104002), ANANDHI A (714023104006), DSHANTHINI R (714023104024), KAVITHA K (714023104048)**",who carried out the work under my supervision.

SIGNATURE

Mrs.M.MANIMEGALA

SUPERVISOR

Assistant Professor

Department of CSE,

Sri Shakthi Institute of

Engineering and Technology,

Coimbatore-641062.

SIGNATURE

Dr.K.E.KANNAMMAL

HEAD OF THE DEPARTMENT

Professor and Head

Department of CSE,

Sri Shakthi Institute of

Engineering and Technology

Coimbatore-641062.

Submitted for the project work viva-voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost, I would like to thank God Almighty for giving me the strength. Without his blessings this achievement would not have been possible.

We expressourdeepest gratitude to our Chairman **Dr.S.Thangavelu** for his continuous encouragement and support throughout our course of study.

We are thankful to our Secretary **Mr.T.Dheepan** for his unwavering support during the entire course of this project work.

We are also thankful to our Joint Secretary **Mr.T.Sheelan** for his support during the entire course of this project work.

We are highly indebted to Principal **Dr.N.K.Sakthivel, M.Tech, Ph.D** for his support during the tenure of the project.

We are deeply indebted to out Head of the Department, Computer Science and Engineering, **Dr.K.E.Kannammal,** for providing us with the necessary facilities.

It's a great pleasure to thank our Project Guide **Mrs.M.Manimegala** for her valuable technical suggestions and continuous guidance throughout this project work.

We solemnly extend out thanks to all the teachers and non-teaching staff of our department, family and friends for their valuable support.

ABITHA M

ANANDHI A

DSHANTHINI R

KAVITHA K

ABSTRACT

In today's digital-first environment, the verification of documents has become a mission-critical task for organizations across sectors such as education, finance, law, and government. With the growing sophistication of forgery techniques, traditional methods of document validation are no longer sufficient. DeepScan is a next- generation document authentication application designed to detect and prevent the use of fake or tampered documents through the integration of advanced image processing, deep learning, and optical character recognition (OCR).The application enables users to scan physical or digital documents using a mobile device or desktop interface. Once scanned, DeepScan performs a comprehensive analysis by comparing the document against trusted templates or verified datasets. It examines elements such as fonts, spacing, alignment, official seals, signatures, and embedded metadata to detect discrepancies. OCR technology is utilized to extract and analyze textual information, while machine learning models are employed to identify subtle signs of manipulation that might escape human detection. DeepScan's real-time processing capability ensures that users receive instant feedback on the authenticity of the document, accompanied by a detailed verification report. This significantly streamlines verification workflows for institutions that deal with high volumes of documentation, including universities, human resources departments, banks, and legal entities.The app also features modular integration with external verification APIs and government or institutional databases, allowing it to cross-reference data for added accuracy and legitimacy. Furthermore, its self-learning algorithm continuously evolves by recognizing and adapting to emerging forgery trends, thereby maintaining its effectiveness in a constantly changing threat landscape.By automating the document verification process and reducing the margin for human error, DeepScan enhances organizational security and operational efficiency. Its intuitive user interface, coupled with robust backend intelligence, makes it a reliable solution for ensuring document authenticity. DeepScan stands as a vital tool in the fight against document fraud, safeguarding trust and integrity in both digital and paper-based records.

TABLE OF CONTENT

CHAPTER -1

INTRODUCTION

1.1 INTRODUCTION TO DEEP SCAN

Document forgery is a growing problem in the digital age, impacting various sectors such as education, finance, legal services, employment, and government institutions. The rapid advancement of image editing software and desktop publishing tools has made it easier for individuals to create fake documents that are visually convincing and difficult to distinguish from original ones. This has led to a rise in the submission of counterfeit certificates, identity proofs, and other official documents, resulting in significant security, financial, and reputational risks for organizations.Traditional methods of verifying documents involve manual inspection by trained personnel, cross-checking details with databases, and contacting issuing authorities. These methods, while sometimes effective, are time-consuming, resource- intensive, and prone to human error.

Developed using Flutter, a cross-platform UI framework, the app is capable of running on both Android and iOS devices. It uses Optical Character Recognition (OCR) to extract text from scanned images and combines it with image processing and machine learning techniques to detect inconsistencies or signs of forgery.The application is designed to be simple, fast, and effective. Users can scan a document using their phone’s camera, and the system will analyze the image to determine its authenticity. Key indicators such as inconsistent fonts, irregular text alignment, unnatural edges, and signs of image manipulation are examined to detect tampering. The use of Flutter ensures a smooth user experience and broad accessibility, while the integrated technologies allow for real-time results.

CHAPTER -2

LITERATURE REVIEW

1. Introduction to Digital Document Verification in DeepScan

DeepScan is a cutting-edge fake document detection app developed to combat the growing threat of document fraud across various sectors. As organizations increasingly shift to digital workflows, the risk of forged certificates, tampered ID proofs, and counterfeit documents has risen sharply. DeepScan aims to provide a reliable and automated solution for document verification using intelligent technologies such as Optical Character Recognition (OCR), data validation, and AI- driven analysis. Studies highlight the critical role of such systems in enhancing institutional security, streamlining verification, and minimizing human error.

2. Technological Advancements in Fake Document Detection

DeepScan's architecture is built on a foundation of advanced technologies that have significantly evolved in recent years. OCR enables accurate extraction of textual data from scanned documents and images, while deep learning models assess structural integrity, font patterns, and visual inconsistencies. Research shows that integrating machine learning with rule-based analysis enhances detection capabilities, particularly in identifying subtle forgeries and edited content. In addition, cloud- based services and APIs allow DeepScan to cross-verify extracted information against trusted databases for real-time validation.

3. Impact on User Experience and Trust

User-centric design is essential for technology adoption, and DeepScan emphasiz peed, simplicity, and transparency in its operation. Literature indicates that automated document verification increases user trust, especially when results are delivered quickly and accompanied by clear explanations of detected issues. By reducing reliance on manual checks, DeepScan improves efficiency in high-volume scenarios such as university admissions, bank account openings, and government services. The app's ability to deliver accurate and unbiased

results enhances confidence among both institutional users and individuals.

4. Applications of Fake Document Detection Across Sectors

DeepScan's technology is widely applicable across domains where document authenticity is critical. In education, it verifies academic certificates and transcripts to prevent admission fraud. In the banking and financial sector, DeepScan checks KYC documents, pay slips, and bank statements. Government agencies use it to authenticate identity documents like voter IDs and birth certificates. Research supports the scalability of such tools in diverse sectors, noting significant reductions in processing time and fraud cases when digital verification is implemented.

5. Challenges and Limitations in Document Detection Systems

Despite its effectiveness, DeepScan and similar systems face several technical and operational challenges. Studies point to limitations in OCR when dealing with handwritten text, poor lighting, or low-quality scans. Some sophisticated forgeries may escape detection if they replicate formatting and typography with high precision. Additionally, access to up-to-date and authentic reference databases is crucial for accurate validation. Privacy and data security also remain key concerns, especially when handling sensitive personal documents prompting the need for strong encryption and compliance with data protection regulations.

6. Theoretical Perspectives on Technology Adoption

To understand adoption of DeepScan, models such as the Technology Acceptance Model (TAM) and the Unified Theory of Acceptance and Use of Technology (UTAUT) are often applied. These frameworks reveal that users are more likely to adopt such technologies if they perceive them as useful, easy to use, and secure. Studies also highlight the role of institutional trust, previous exposure to fraud, and regulatory requirements in shaping adoption trends across industries and user demographics.

7. Future Directions and Opportunities

The future of document fraud detection apps like DeepScan lies in continuous innovation. Integrating Natural Language Processing (NLP) for semantic analysis of content, using blockchain for immutable verification records, and applying federated learning for privacy-preserving AI training are some promising directions. As 5G and edge computing become widespread, DeepScan could provide faster and more localized processing. Researchers also see opportunities for expanding into cross- border document validation, multilingual support, and integration with global verification networks.

CHAPTER 3

MODULE DESCRIPTION

3.1 EXISTING MODULE

1. App Initialization

Purpose: Set up the environment for document scanning.

Real-Time Example: When a user launches DeepScan, it checks for camera access, initializes image processing libraries, and prepares the UI for scanning.

Key Elements:

Camera Access: Activates the rear camera to capture documents.

Permission Handling: Prompts for camera and storage access.

Environment Check: Ensures proper lighting and focus for accurate detection.

2. Document Detection

Purpose: Automatically detect and crop documents from camera input.

Real-Time Example: User holds up a driver's license—DeepScan detects the edges and outlines the document for cropping.

Key Elements:

Edge Detection: Identifies the borders of the document. Auto-

Cropping: Adjusts the frame to isolate the document.

Perspective Correction: Fixes skewed angles for proper analysis.

3. Fake Document Detection

Purpose: Verify the authenticity of scanned documents.

Real-Time Example: After scanning a diploma, DeepScan cross-references the design

with a database and flags mismatches.

Key Elements:

Template Matching: Compares with official templates. Font/Logo

Analysis: Detects altered text, logos, or seals. Metadata Checks:

Verifies dates, serial numbers, and formatting.

4. Interaction with Scan Results

Purpose: Allow users to review and act on results.

Real-Time Example: User sees a "Fake Detected" flag and can tap to see mismatched areas highlighted in red.

Key Elements:

Highlight Errors: Visually marks discrepancies.

Details Panel: Shows why a document was marked fake.

Save/Share Options: Allows storing or sending results.

5. **UI Integration**

Purpose: Provide an intuitive interface for users to scan and check documents.

Real-Time Example: The main screen has a big "Scan Now" button, history tab, and access to help.

Key Elements:

Navigation Bar: Home, Scan, History, Help. Search

Function: Find previous scans.

Dark/Light Mode: Enhances visibility in various lighting.

6. Performance Optimization

Purpose: Ensure fast and accurate scanning.

Real-Time Example: The app quickly processes a scanned image and shows results in under 3 seconds.

Key Elements:

Lightweight Models: Efficient AI models for real-time checking.

Multi-threading: Background processing for smooth UI. Compression:

Compresses scan data without losing quality.

7. Analytics and Insights

Purpose: Track usage to improve performance.

Real-Time Example: DeepScan logs how often certain types of documents are scanned and flagged.

Key Elements:

Usage Stats: Document types, scan times, error rates.

User Feedback: In-app surveys or thumbs up/down on scan accuracy

8. Multi-Platform Compatibility

Purpose: Work across Android and iOS.

Real-Time Example: A scan performed on an Android phone looks the same and functions the same as on iOS.

Key Elements:

Cross-platform UI Framework: Flutter or React Native.

Device Testing: Optimized for a range of smartphones and tablets

9. Multi-Document Scanning

Purpose: Scan and analyze multiple documents in a session.

Real-Time Example: User scans an ID and utility bill in sequence for KYC verification. Key

Elements:

Batch Scanning: Scan multiple docs without restarting.

Session Management: Group scans under one session ID

Auto-save: Save all scanned documents in history.

3.2 PROPOSED METHODOLOGY

1. Problem Definition and Scope

Design an AI-powered mobile application that assists users in detecting fake or altered documents using real-time scanning and verification techniques.

Challenges:

Accurate recognition and validation of document elements (e.g., text, seals, fonts, barcodes).

Real-time performance on various mobile devices. Supporting a

wide variety of document formats and layouts. Ensuring data

privacy and secure processing.

2. Technology Stack Selection

Development Platform: Android Studio or Flutter (for cross-platform support).

Programming Languages: Kotlin/Java (Android), Dart (Flutter), Python (for AI backend, if needed).

AI/ML Frameworks: TensorFlow Lite or OpenCV for image analysis, OCR (e.g., Tesseract).

Cloud Services: Firebase or AWS for database, analytics.

Database: SQLite (local), Firestore (cloud-based for user history or templates).

3. Environment Setup

1. Initialize camera access with real-time frame processing capabilities.

2. Configure the pipeline for image pre-processing (contrast enhancement, noise reduction, edge detection).

3. Set up OCR and document template-matching modules.

4. Secure user permissions for camera, storage, and internet.

4. Document Model Preparation

1. Collect and categorize official templates of common documents (e.g., IDs, licenses, certificates).

2. Define key fields for each type (e.g., name, date of birth, ID number, seal position).

3. Train machine learning models to recognize tampering indicators like font mismatches, forged signatures, or incorrect layouts**.**

5. User Interface (UI) Design

Scan Button: Prominent access to start a scan.

Document Type Selector: Dropdown for selecting the expected document type. Result

Screen: Displays analysis results, mismatches, and authenticity status.

History Tab: Allows access to previous scans and reports.

Customization Options: Theme toggle (light/dark), offline/online mode toggle, etc.

6. Document Scanning and Interaction

1. Use the device camera to capture the document image.

2. Detect and correct perspective distortion.

3. Allow users to adjust the crop manually if automatic etection fails.

4. Apply OCR and compare text, fonts, and layout to known standards.

5. Highlight mismatched or suspicious areas interactively.

7. Lighting and Quality Handling

1. Implement brightness and sharpness checks to guide users during scanning.
2. Notify users if glare, blur, or poor contrast affects scan quality.
3. Use adaptive filters based on environment (low-light, overexposure, etc.).

8. Advanced Features Development

Real-Time Comparison: Compare scanned documents to official templates in real- time.

Field Matching: Automatically compare names, numbers, and dates between documents.

Report Generation: Auto-generate a verification report (PDF/JSON).

Multi-Document Session: Enable scanning multiple documents in one session for KYC or legal workflows.

9. Performance Optimization

1. Optimize image processing and OCR performance using native libraries.
2. Compress image data to reduce memory usage.
3. Minimize latency by processing images on-device with fallback to cloud if needed.
4. Conduct device-specific profiling to maintain frame rate and avoid crashes.

10. Testing and Validation

1. Conduct unit and integration testing for document analysis and OCR modules.
2. Perform validation tests with real and fake samples to measure accuracy, precision, and recall.
3. Test across multiple device models and OS versions for compatibility.

4. Perform user testing with different age groups and literacy levels for usability.

11. Deployment and Maintenance

1. Build the app for Android (and iOS if cross-platform).
2. Publish to Google Play Store and/or App Store.
3. Monitor crash logs, user feedback, and analytics.
4. Release updates for new document formats, model improvements, and bug fixes.

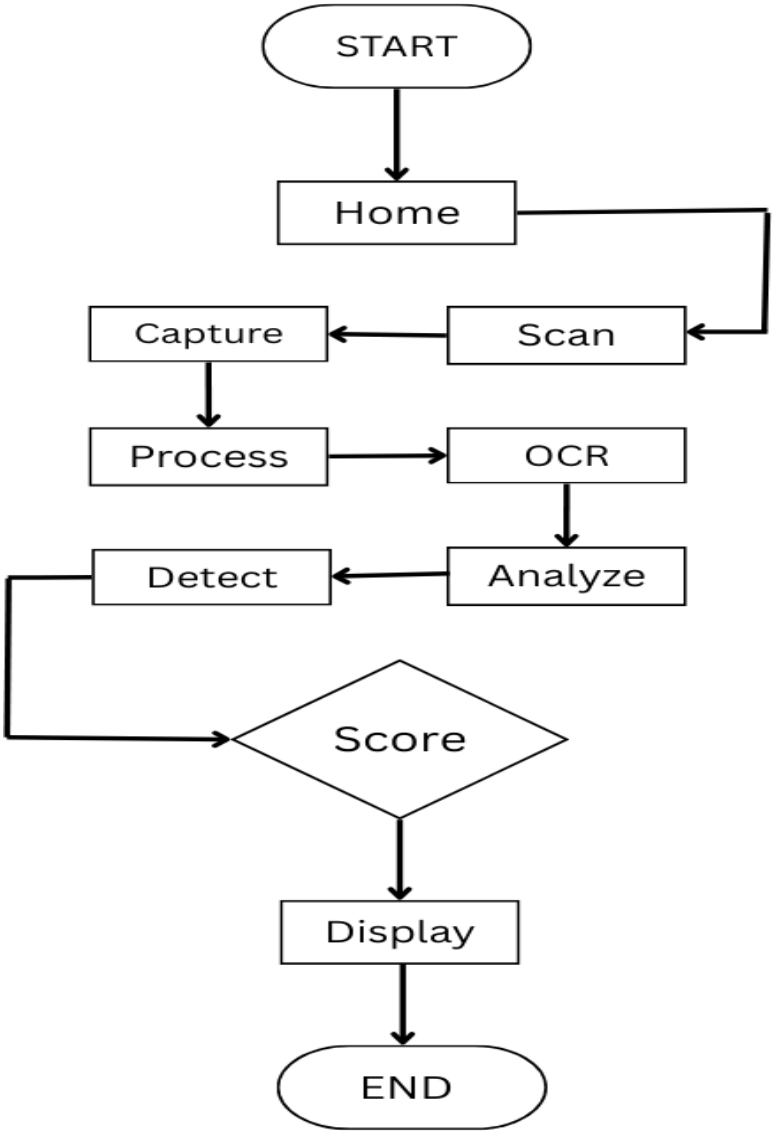
12. Analytics and Insights

Purpose: Improve user experience and fraud detection accuracy over time.

Steps:

1. Track scan frequency, document types, detection accuracy, and usage time.
2. Analyze failure rates or user corrections to refine models.
3. Collect anonymized feedback on misclassified documents to improve datasets.
4. Use trend data to prioritize new features (e.g., most requested document types).

3.3 FLOWCHART



3.4 ADVANTAGES

1. Real-Time Document Verification

Deep Scan enables instant verification of documents directly on a mobile device. Users can scan and analyze documents in real-time, which is crucial for environments like interviews, admissions, or legal checks where quick verification is essential.

2. Cross-Platform Compatibility

Developed using the Flutter framework, Deep Scan is a cross-platform application, meaning it can run on both Android and iOS devices. This ensures that users across different operating systems can utilize the same application without needing separate development for each platform.

3. Integrated OCR and Image Analysis

Unlike many existing document scanning apps that focus solely on OCR, Deep Scan combines text extraction and forgery detection. It not only scans and extracts text but also examines the document's visual elements for inconsistencies, such as altered text, blurred areas, and irregular fonts, providing a more comprehensive analysis.

4. Offline Functionality

Deep Scan is capable of working offline, meaning users can still scan and verify documents without needing a constant internet connection. This is particularly beneficial in remote areas, regions with low connectivity, or when traveling where internet access may be limited.

5. User-Friendly Interface

The app is designed with a clean and simple interface that requires minimal training or technical knowledge. Its intuitive design ensures that even people with limited technical expertise can easily scan and verify documents on the go.

6. Cost-Effective Solution

The application provides an affordable alternative to traditional document verification tools, which often require expensive scanners, specialized hardware, or third-party services. Deep Scan is a cost-effective solution for small businesses, educational institutions, and individuals, eliminating the need for high-cost verification equipment.

7. Improved Data Privacy and Security

As all document processing is performed locally on the user's device, sensitive information is not uploaded to external servers. This ensures enhanced privacy and security, addressing concerns about data leaks and unauthorized access.

8. Quick Feedback and Results

Once a document is scanned and processed, users receive immediate feedback on its authenticity. The results are displayed in a user-friendly format, showing whether the document is genuine or contains signs of tampering, allowing users to act quickly without waiting for manual verification.

9. Lightweight and Mobile Optimized

Deep Scan is optimized for mobile devices, ensuring it runs smoothly even on lower- end smartphones. The app uses lightweight processing techniques that allow it to function efficiently without taxing device resources, making it accessible to a wide range of users with varying device capabilities.

10. Scalability and Flexibility

Deep Scan can be easily scaled to incorporate additional features such as machine learning models for advanced forgery detection, signature recognition, or integration with cloud databases. The modular nature of the system allows for future upgrades, ensuring it can evolve with advancements in technology.

3.5 DISADVANTAGES

1. Accuracy Relies on Image Quality

The app's OCR and forgery detection performance depend heavily on the clarity and resolution of the scanned image. Blurry, poorly lit, or skewed images can lead to incorrect text extraction and may prevent the system from detecting alterations effectively.

2. Limited Advanced Forgery Detection Without AI

Without incorporating deep learning techniques, the app may not detect forgeries that are highly sophisticated or involve subtle changes, such as micro-adjustments in font size, line spacing, or background patterns.

3. Cannot Verify Actual Content Validity

While Deep Scan can detect signs of tampering, it cannot verify whether the information in a document (e.g., a certificate's marks, an ID number) is truthful unless integrated with a trusted database or official source.

4. Lack of Signature and Stamp Authentication

Identifying whether a signature or official seal is real or forged requires biometric or forensic tools. This functionality is not included in standard OCR-based systems, making the app insufficient for high-level verification cases.

5. Performance Issues on Low-End Devices

Devices with limited processing power or low RAM may struggle to handle image processing, OCR, and analysis tasks. This can result in application crashes, slow performance, or incomplete scans.

6. No Legal Validity of Results

The analysis and results provided by the app are not recognized as legally certified. Therefore, the app cannot replace human validation or certified forensic tools in official or legal scenarios.

7. Offline Mode Lacks External Validation

While offline functionality is helpful, it also limits the app's ability to validate document details against external databases (e.g., university records, government registries), which is often necessary for authentic verification.

8. Language and Font Limitations

OCR engines like Tesseract may not accurately read documents in regional languages, complex scripts, or uncommon fonts. This limits the app's usefulness for multilingual document verification, especially in diverse regions.

9. Model Dependency in AI-Based Versions

If a machine learning model is used for forgery detection, its accuracy depends on the quality and diversity of the training dataset. Incomplete or biased datasets can cause false positives or missed forgeries.

10. Unable to Detect Skilled Forgeries Visually Identical to Originals

Some forgeries are created with such precision that they visually mimic real documents perfectly. Detecting these requires forensic analysis beyond the capabilities of basic image processing or OCR-based tools.

CHAPTER 4

LANGUAGE DESCRIPTION

Dart (Frontend - Flutter)

Dart, used with Flutter, enables fast and efficient cross-platform app development. It creates smooth, responsive user interfaces for both mobile and web platforms. In the fake document detection app, Dart handles the frontend, enabling easy document uploads, scans, and real-time feedback. The Flutter framework allows for a consistent, high-performance user experience, ensuring the app runs efficiently on multiple devices while maintaining a clean, intuitive design. Dart's asynchronous programming and rich libraries ensure seamless interaction between users and backend processing, making it an ideal choice for a responsive frontend in the app.

Python (Backend Processing & Machine Learning)

Python powers backend processing in the fake document detection app. It uses libraries like **OpenCV** for image analysis and **TensorFlow** for machine learning, identifying potential document forgeries. Python processes scanned documents, analyzes text and images, and runs algorithms to detect irregularities. Its flexibility and vast library ecosystem make Python perfect for backend development, ensuring accurate results. The app leverages Python's simplicity and power to handle document analysis tasks, providing real-time feedback and reliable document verification, crucial for distinguishing fake documents from genuine ones.

Node.js (Backend & Asynchronous Processing)

Node.js handles backend operations in the fake document detection app, managing document uploads and asynchronous tasks. Its non-blocking, event-driven architecture supports multiple requests simultaneously, ensuring the app is scalable and responsive. Node.js processes user requests, interacts with Python scripts, and sends real-time updates to the frontend. By using frameworks like Express.js, Node.js builds RESTful APIs, enabling smooth communication between the frontend and backend. This ensures fast document verification, real-time alerts, and efficient data handling, improving user experience and app

performance.

MongoDB (Database Storage)

MongoDB is used to store document metadata, verification history, and user data in the app. Its flexible, document-oriented structure handles diverse document formats and large volumes of data efficiently.NoSQL nature of MongoDB allows dynamic, schema- less storage, making it ideal for the app's varied data. It enables fast querying and easy modification of document records, supporting the app’s scalable backend as it grows. MongoDB’s ability to handle high data throughput,maintain performance ensures the app operates while managing increasing user and document data.

Flutter (Cross-platform Development)

Flutter, powered by Dart, enables the creation of fast, natively compiled applications for mobile, web, and desktop from a single codebase. It provides a smooth, responsive UI for the fake document detection app, ensuring a seamless experience across platforms. Flutter’s widget-based architecture allows for easy customization, offering users an intuitive interface to upload documents and receive verification results. The framework’s integration with backend services like Python ensures efficient processing and real-time feedback, making Flutter ideal for building scalable and user-friendly applications.

CHAPTER 5

RESULT AND ANALYSIS

5.1 RESULT

The Fake Document Detection App integrates advanced technologies like Dart (with Flutter for the frontend), Python (for backend processing and machine learning), Node.js (for asynchronous handling), and MongoDB (for data storage) to deliver a highly effective document verification system. The app's frontend, built using Flutter, offers a smooth, responsive, and consistent experience across mobile and web platforms. Users can easily upload and scan documents, with results displayed in real- time. Python's powerful image processing libraries, such as OpenCV, and machine learning models, trained on large datasets, allow the app to detect various document forgeries, analyzing text, images, and document features for authenticity.

On the backend, Node.js ensures the app remains scalable and responsive by handling asynchronous tasks like document uploads and processing without delays. Its event-driven architecture supports multiple users simultaneously, making the app suitable for large-scale applications. MongoDB's flexible, document-oriented database stores verification results and document metadata efficiently, enabling quick queries. The app performs well in real-time document verification, providing users with accurate feedback on whether a document is genuine or forged, thanks to the sophisticated image and text analysis conducted by Python. This combination of technologies ensures the app is both fast and reliable for detecting fake documents.

5.2 FUTURE WORKS

While the current version of the **Fake Document Detection App** is highly effective, there are several avenues for improvement to make the app even more robust and accurate. A key area for future development is improving the **machine learning models** used for document verification. This can be achieved by expanding the training datasets to include

more varied document types, fraudulent patterns, and other sophisticated forgery techniques. Enhanced models will provide better accuracy and be able to detect increasingly complex forgeries.Additionally, the app could benefit from the integration of **Natural Language Processing (NLP)** techniques. By analyzing the linguistic structure of documents, the app could flag irregularities in text, such as inconsistent language, grammar mistakes, or unnatural sentence structures that often appear in forged documents. NLP could be especially useful in detecting issues in documents like contracts, resumes, or reports, where language is crucial to authenticity.

Another significant enhancement would be the ability to scan documents directly using the **camera** in the app, providing users with more convenience and accessibility. Cloud-based processing could be employed to offload intensive tasks, such as image and document analysis, which would improve processing speed and allow for better performance, particularly when dealing with large documents or high traffic from multiple users.

To ensure the app has global appeal, future versions could introduce **multi-language support**, enabling users to verify documents further expanding its utility across various countries and industries.Additionally, **real-time notifications** could be implemented to keep users informed about the verification process, provide immediate alerts in case of potential fraud. Finally, integrating **OCR (Optical Character Recognition)** advancements to improve text recognition in scanned documents would further boost the app's accuracy and effective.

5.3 CONCLUSION

The Deep Scan – Fake Document Detector project represents a modern, efficient, and highly practical solution to the increasing threat of document forgery. In an age where digital manipulation tools are readily available, the need for intelligent document verification has become more critical than ever. This project addresses that need by combining several technologies—image processing, Optical Character Recognition (OCR), and artificial intelligence—within a mobile- friendly application developed using Flutter.

The primary goal of the project was to create a system that could assist in detecting fake or tampered documents in real time, directly from a mobile device. The system follows a structured methodology beginning with document image capture, followed by preprocessing, text extraction, visual analysis, and result generation

One of the key achievements of this project is the integration of multiple verification techniques within a single app. Unlike traditional document scanners that focus solely on text recognition, Deep Scan adds critical layers of analysis such as font irregularity detection, layout consistency checking, and basic tampering forensics. This allows users to not only scan documents but also determine the authenticity of their content, making it far more useful in real-world scenarios.Furthermore, the choice of Flutter for app development ensures that the application is lightweight, fast, and cross- platform compatible.This significantly reduces development time while reaching a broader user base across both Android and iOS devices. The inclusion of offline functionality ensures that users can verify documents in areas with poor or no internet connectivity. Features such as machine learning-based forgery detection, cloud-based verification, and biometric signature analysis could be incorporated in the next phases.

In conclusion, Deep Scan is a promising technological innovation that empowers individuals and organizations to combat document fraud more effectively. It simplifies the verification process, makes it more accessible, and demonstrates how modern mobile technology can be used to solve real-world security challenges.

6. APPENDICES

SOURCE CODE:

main.dart

```
import 'package:flutter/material.dart';
  void main() {
    runApp(const MyApp());
  }
  class MyApp extends StatelessWidget {
    const MyApp({super.key});
    // This widget is the root of your application.
    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          // This is the theme of your application.
          //
          // TRY THIS: Try running your application with "flutter run". You'll see
          // the application has a purple toolbar. Then, without quitting the app,
          // try changing the seedColor in the colorScheme below to Colors.green
          // and then invoke "hot reload" (save your changes or press the "hot
          // reload" button in a Flutter-supported IDE, or press "r" if you used
          // the command line to start the app).
          //
          // Notice that the counter didn't reset back to zero; the application
          // state is not lost during the reload. To reset the state, use hot
          // restart instead
```

```
//
        // This works for code too, not just values: Most code changes can be
        // tested with just a hot reload.
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});
  // This widget is the home page of your application. It is stateful, meaning
  // that it has a State object (defined below) that contains fields that affect
  // how it looks.
  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget) and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  final String title;
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    setState(() {
      // This call to setState tells the Flutter framework that something has
```

```
    // changed in this State, which causes it to rerun the build method below
    // so that the display can reflect the updated values. If we changed
    // _counter without calling setState(), then the build method would not be
    // called again, and so nothing would appear to happen.
    _counter++;
  });
}
@override
Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(
      // TRY THIS: Try changing the color here to a specific color (to
      // Colors.amber, perhaps?) and trigger a hot reload to see the AppBar
      // change color while the other colors stay the same.
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      // Here we take the value from the MyHomePage object that was created by
      // the App.build method, and use it to set our appbar title.
      title: Text(widget.title),
    ),
    body: Center(
      // Center is a layout widget. It takes a single child and positions it
      // in the middle of the parent.
      child: Column(
```

```
    // Column is also a layout widget. It takes a list of children and
    // arranges them vertically. By default, it sizes itself to fit its
    // children horizontally, and tries to be as tall as its parent.
    //
    // Column has various properties to control how it sizes itself and
    // how it positions its children. Here we use mainAxisAlignment to
    // center the children vertically; the main axis here is the vertical
    // axis because Columns are vertical (the cross axis would be
    // horizontal).
    //
    // TRY THIS: Invoke "debug painting" (choose the "Toggle Debug Paint"
    // action in the IDE, or press "p" in the console), to see the
    // wireframe for each widget.
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      const Text(
        'You have pushed the button this many times:',
      ),
      Text(
        '$_counter',
        style: Theme.of(context).textTheme.headlineMedium,
      ),
    ],
  ),
),
floatingActionButton: FloatingActionButton(
  onPressed: _incrementCounter,
  tooltip: 'Increment',
  child: const Icon(Icons.add),
```

```
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

pubspec.yaml

```
name: document_verifier
description: "A new Flutter project."
# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as
versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used
as CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistK
eyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
```

```
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
  sdk: '>=3.3.4 <4.0.0'

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter
  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^3.0.0
```

 # For information on the generic Dart part of this file, see the
 # following page: https://dart.dev/tools/pub/pubspec

 # The following section is specific to Flutter packages.
flutter:

 # The following line ensures that the Material Icons font is
 # included with your application, so that you can use the icons in
 # the material Icons class.
uses-material-design: true

 # To add assets to your application, add an assets section, like this:
 # assets:
 # - images/a_dot_burr.jpeg
 # - images/a_dot_ham.jpeg

 # An image asset can refer to one or more resolution-specific "variants", see
 # https://flutter.dev/assets-and-images/#resolution-aware

 # For details regarding adding assets from package dependencies, see
 # https://flutter.dev/assets-and-images/#from-packages

 # To add custom fonts to your application, add a fonts section here,
 # in this "flutter" section. Each entry in this list should have a
 # "family" key with the font family name, and a "fonts" key with a
 # list giving the asset and other descriptors for the font. For
 # example:
 # fonts:

```
  #  - family: Schyler
  #    fonts:
  #      - asset: fonts/Schyler-Regular.ttf
  #      - asset: fonts/Schyler-Italic.ttf
  #        style: italic
  #  - family: Trajan Pro
  #    fonts:
  #      - asset: fonts/TrajanPro.ttf
  #      - asset: fonts/TrajanPro_Bold.ttf
  #        weight: 700
```

AndroidManifest.xml

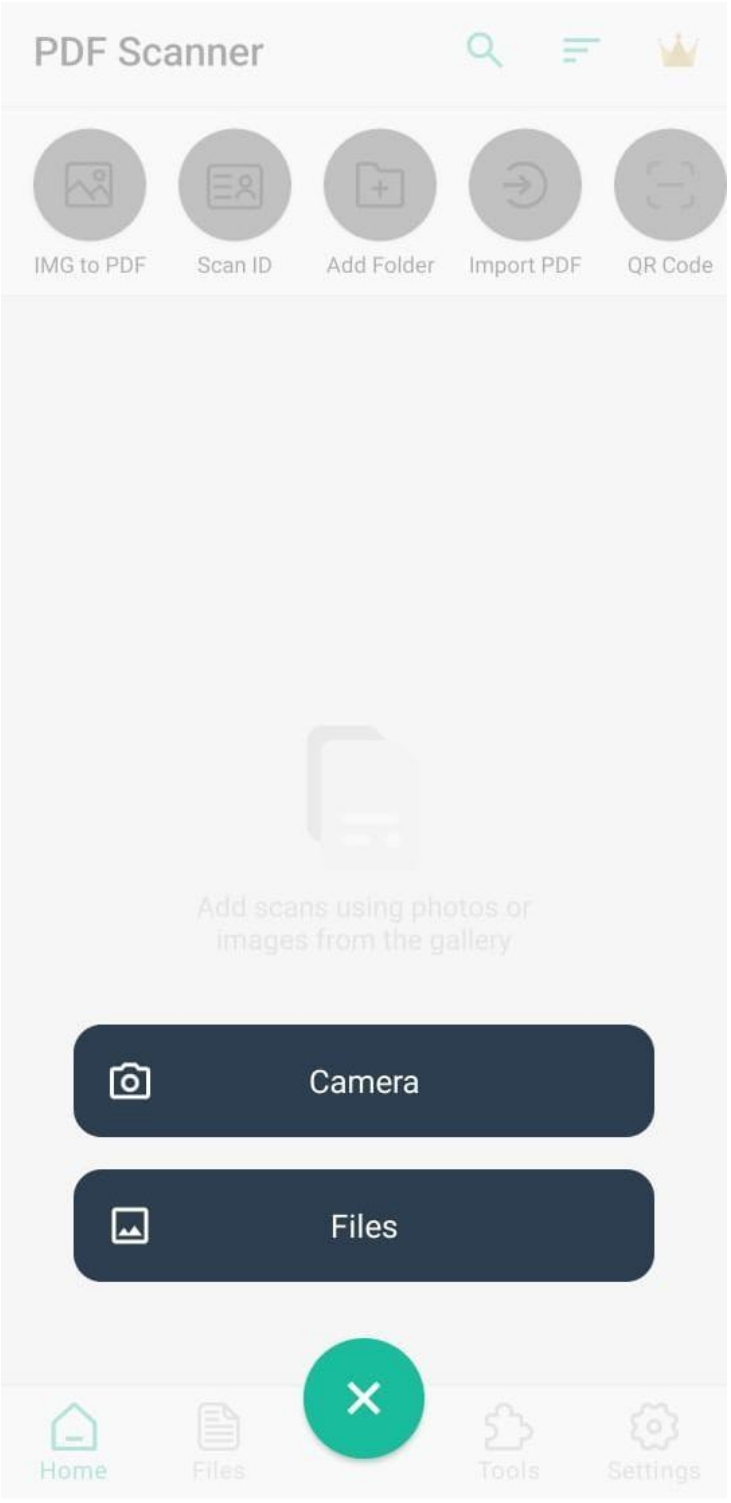
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.document_verifier">

    <!-- Required permission to access the camera -->
    <uses-permission android:name="android.permission.CAMERA" />

    <application
        android:label="document_verifier"
        android:icon="@mipmap/ic_launcher">

        <!-- Main activity configuration -->
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:launchMode="singleTop"
```

```
        android:theme="@style/LaunchTheme"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|l
ocale|layoutDirection|fontScale|screenLayout|density|uiMode"
        android:hardwareAccelerated="true"
        android:windowSoftInputMode="adjustResize">

        <!-- Specifies an Android theme to apply to this Activity as soon as the Android
process has started. -->


        <!-- Intent filter to launch the app -->
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>

      <!-- Don't delete the meta-data below. Flutter plugin generation -->
      <meta-data
        android:name="flutterEmbedding"
        android:value="2" />

    </application>
    <uses-feature android:name="android.hardware.camera" android:required="false" />

    <!-- Required to query activities that can process text -->
    <queries>
      <intent>
```

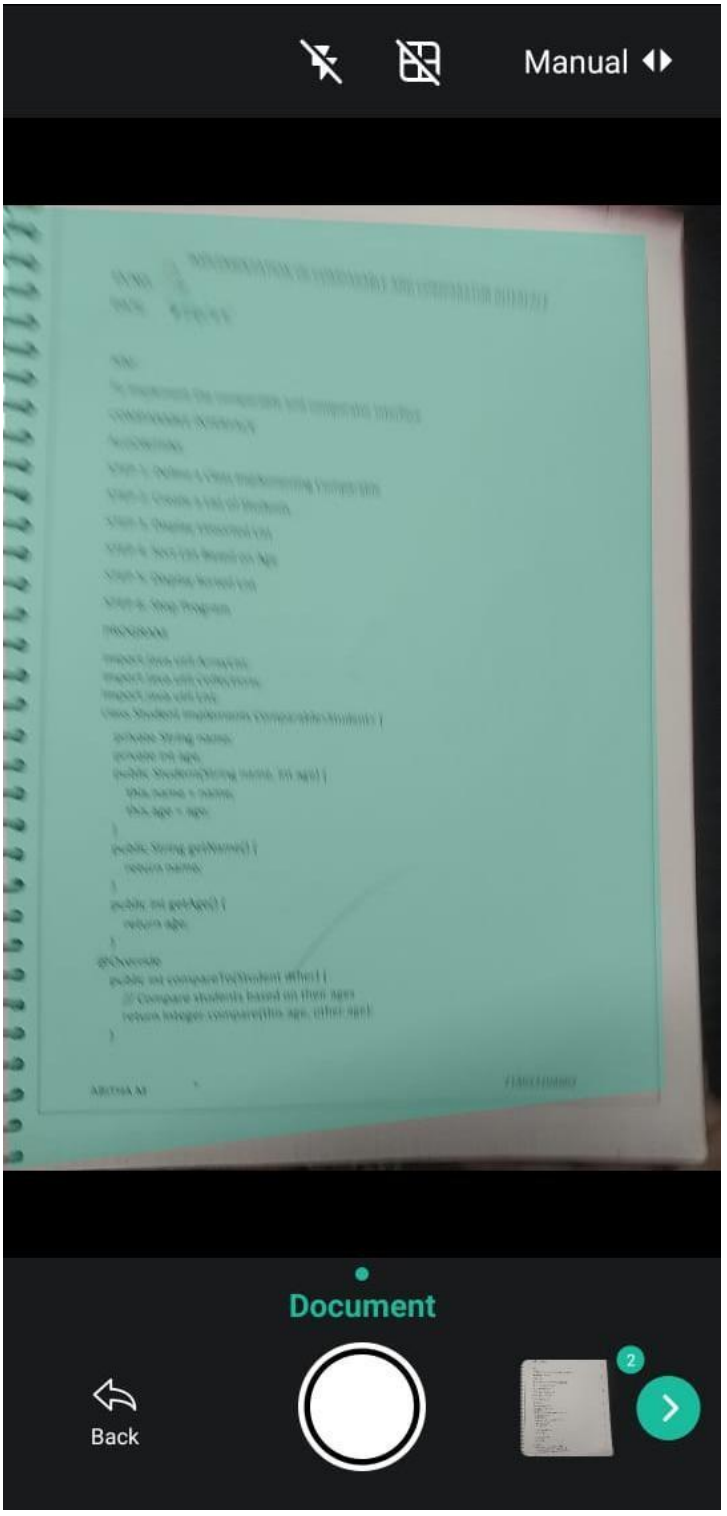
```
            <action android:name="android.intent.action.PROCESS_TEXT" />
            <data android:mimeType="text/plain" />
        </intent>
    </queries>

</manifest>
```

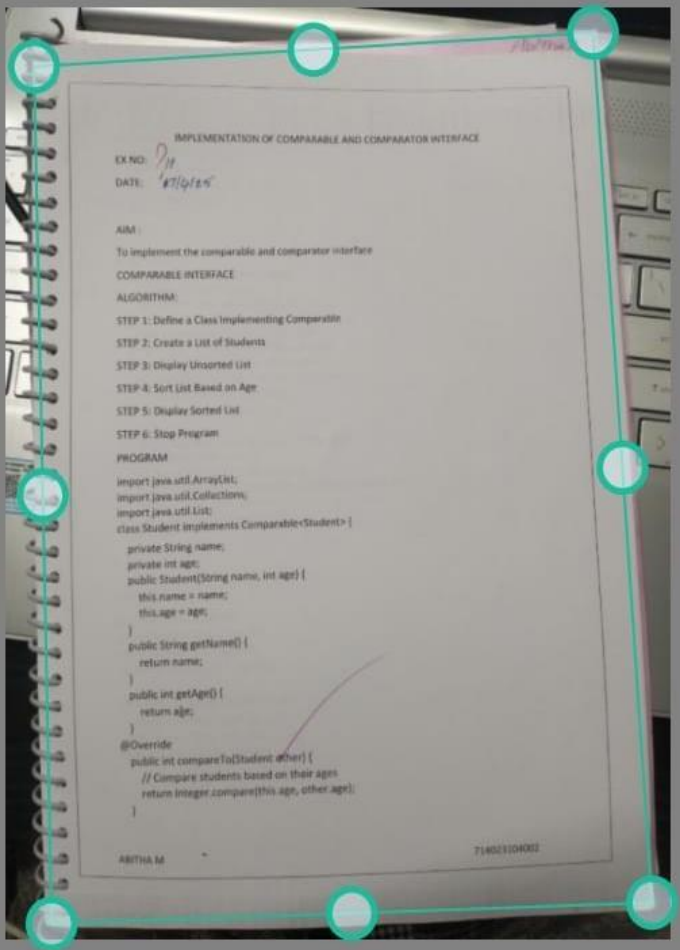
OUTPUT:

Home page:


PDF Scanner
IMG to PDF
Scan ID
Add Folder
Import PDF
QR Code
Add scans using photos or images from the gallery
Camera
Files
Home
Files
Tools
Settings

Scanner:





IMPLEMENTATION OF COMPARABLE AND COMPARATOR INTERFACE

EX NO: 11

DATE: 27/04/25

AIM :

To implement the comparable and comparator interface

COMPARABLE INTERFACE

ALGORITHM:

STEP 1: Define a Class Implementing Comparable

STEP 2: Create a List of Students

STEP 3: Display Unsorted List

STEP 4: Sort List Based on Age

STEP 5: Display Sorted List

STEP 6: Stop Program

PROGRAM

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Student implements Comparable<Student> {
  private String name;
  private int age;
  public Student(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
@Override
  public int compareTo(Student other) {
    // Compare students based on their ages
    return Integer.compare(this.age, other.age);
  }
```

ARITHA M

No crop Left Right NEXT

REFERENCES:

1. Document Scanner Application Using Python **:** https://www.irjmets.com/uploadedfiles/paper/volume2/issue_5_may_2020/1205/1628083027.pdf

2. Efficiency Redefined: https://www.researchgate.net/publication/388908237_Efficiency_Redefined_The_Document_Scanner_App_f

3. DocScanner: https://arxiv.org/abs/2110.14968

4. Deep Reader: https://arxiv.org/abs/1812.04377

5. LayoutLM: https://arxiv.org/abs/1912.13318

6. Adobe Scan https://www.adobe.com/acrobat/mobile/scanner-app.html