DETECTOO - Complete Project Explanation

Made by: Sumit - 9527352323

Table of Contents

Project Overview

Detectoo is a web-based AI image detection tool built with React. It helps users identify whether an image is AI-generated or a real photograph by analyzing pixel patterns, color distribution, and other image characteristics.

Purpose: Combat misinformation and deepfakes by providing an easy-to-use tool that analyzes images at the pixel level.

Type: Educational project / Web Application

Development Team:

- Anisha Sathe
- Sneha Pawar
- Shravani Sonawane

- College of Engineering, Pune

What Does This Project Do?

Imagine you see a photo online and wonder, "Is this real or created by AI?" This tool helps you find out.

Main Functionality:

1. User uploads an image file (JPG, PNG, WebP, GIF)
2. The application analyzes the image using pixel-level algorithms
3. The tool displays:
    - Whether the image is AI-generated or real
    - Confidence percentage of the analysis
    - A heatmap showing which parts of the image are AI-generated
    - Detailed metrics about the image
    - A downloadable PDF report

Real-World Use Cases:

- Journalists verifying news images
- Social media users checking viral photos
- Researchers studying AI-generated content
- Students learning about image analysis

Technology Stack

Frontend Framework

- **React 18.2.0** - JavaScript library for building user interfaces
- **React DOM 18.2.0** - Connects React to the browser

Build Tool

- **React Scripts 5.0.1** - Handles all the complex build configuration

Deployment

- **Netlify** - Cloud hosting platform for web applications

Languages Used

- **JavaScript (99.1%)** - Main programming language
- **HTML (0.9%)** - Basic page structure
- **CSS (Inline)** - Styling written directly in JavaScript

Browser APIs Used

- **FileReader API** - Reads uploaded image files
- **Canvas API** - Processes and analyzes pixels
- **Image API** - Loads and manipulates images

Project Structure

```
detectoo/
│
├── public/                 # Static files folder
│   └── index.html          # Main HTML file
│
├── src/                    # Source code folder
│   ├── App.jsx             # Main application component (935 lines)
│   └── index.jsx           # Entry point that renders App
│
├── .gitignore              # Files to ignore in version control
├── package.json            # Project dependencies and scripts
└── netlify.toml            # Deployment configuration
```

How It Works - Baby Steps

Let me explain the entire process as if explaining to a beginner:

Step 1: User Opens the Application

- The browser loads `index.html`
- React takes over and renders the `App` component
- User sees a beautiful homepage with navigation menu

Step 2: User Navigates to Detector Page

- User clicks on "Detector" in the navigation menu
- The page changes to show an upload area
- User can drag-and-drop or click to select an image

Step 3: Image Upload Process

```
User selects image file
   ↓
Browser reads the file using FileReader
   ↓
File is converted to base64 data (text representation of image)
   ↓
Image is displayed on screen
   ↓
Analysis process begins automatically
```

Step 4: Image Analysis (The Magic Happens Here)

Phase 1: Overall Analysis

- The system generates a random determination (AI or Real)
- This is for demonstration purposes
- Calculates confidence percentage (55-90%)
- Records processing time

Phase 2: Region-Based Analysis

- Image is divided into 80x80 pixel squares (regions)
- Each region is analyzed separately
- For each region:
    - Extract pixel data
    - Calculate color uniformity
    - Determine if region looks AI-generated or real
    - Assign confidence score

Phase 3: Metric Calculation

- **Frequency Entropy:** Measures randomness in patterns

- **Color Variance:** Analyzes color distribution
- **Edge Consistency:** Examines edge sharpness
- **Noise Level:** Checks for camera sensor noise

Step 5: Results Display

- Shows verdict (AI Generated or Real Image)
- Displays confidence percentage
- Shows all metrics in a dashboard
- Provides "View Heatmap" option

Step 6: Heatmap Visualization (Optional)

- User clicks "View Heatmap"
- Creates a visual overlay on the image
- Red regions = AI-generated parts
- Green regions = Real photo parts
- Shows which specific areas are suspicious

Step 7: Download Report

- User can download a text report
- Contains all analysis details
- Lists all regions with their verdicts
- Includes file information

Code Breakdown - File by File

1. index.html (Entry Point)

```
html
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#667eea" />
    <meta name="description" content="Detectoo - Advanced AI Image Detection Tool" />
    <title>Detectoo - AI Image Detection</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

What This Does:

- Creates the basic HTML structure
- Sets metadata for search engines
- Creates a `div` with id="root" where React will inject the app
- Sets theme color for mobile browsers
- Defines page title and description

---

2. index.jsx (React Entry Point)

javascript

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

What This Does:

- Imports React and ReactDOM libraries

- Imports the main App component
- Finds the HTML element with id="root"
- Renders the App component inside that element
- StrictMode helps catch potential problems during development

---

3. package.json (Project Configuration)

```json
{
  "name": "detectoo",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

What This Does:

- Defines project name and version
- Lists required libraries (React and React-DOM)
- Defines commands:
  - `npm start` - Runs development server
  - `npm build` - Creates production-ready files
  - `npm test` - Runs tests
  - `npm eject` - Removes build tool abstraction

---

4. netlify.toml (Deployment Settings)

```toml
```

```
[build]
  command = "npm run build"
  publish = "build"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

What This Does:

- Tells Netlify how to build the project
- `command` - Run `npm run build` to create production files
- `publish` - Serve files from the "build" folder
- `redirects` - Send all routes to index.html (for single-page app routing)

---

5. App.jsx (Main Application - 935 lines)

This is the brain of the application. Let me break it down section by section.

State Management (Lines 1-12)

```javascript
const [currentPage, setCurrentPage] = useState('home');
const [image, setImage] = useState(null);
const [result, setResult] = useState(null);
const [loading, setLoading] = useState(false);
const [error, setError] = useState(null);
const [history, setHistory] = useState([]);
const [showHeatmap, setShowHeatmap] = useState(false);
const [heatmapCanvas, setHeatmapCanvas] = useState(null);
const [regions, setRegions] = useState([]);
```

State Variables Explained:

Variable	Type	Purpose
currentPage	String	Tracks which page is displayed (home/detector/about/contact/support)
image	String	Stores uploaded image as base64 data
result	Object	Stores analysis results
loading	Boolean	Shows loading spinner during analysis
error	String	Stores error messages
history	Array	Keeps last 10 analysis results
showHeatmap	Boolean	Controls heatmap visibility
heatmapCanvas	String	Stores heatmap image data
regions	Array	Stores individual region analysis data

Key Features Explained

Feature 1: Image Upload System

How It Works:

1. User clicks on upload area or drags file
2. Browser's FileReader API reads the file
3. File is validated (must be an image)
4. File is converted to base64 string
5. Image is displayed on screen
6. Analysis begins automatically

Code Flow:

```
handleImageUpload()
  → Validates file type
  → Reads file with FileReader
  → Sets image state
  → Calls analyzeImage()
```

Feature 2: Region-Based Analysis

What Are Regions?

- The image is divided into a grid of 80x80 pixel squares
- Each square is analyzed independently
- This allows detecting AI manipulation in specific areas

Analysis Process:

1. Load image into invisible canvas
2. Loop through image in 80x80 pixel chunks
3. For each chunk:
    - Extract all pixel color data
    - Calculate "uniformity" (how similar pixels are)
    - Formula: Sum of color differences between adjacent pixels
    - If uniformity < 20, mark as AI (too perfect/smooth)
    - If uniformity >= 20, mark as Real (natural variation)
4. Store each region's verdict and confidence

Why This Works:

- AI-generated images often have unnaturally smooth gradients
- Real photos have natural noise and variation
- Different parts of an image may have different origins

Feature 3: Heatmap Visualization

Purpose: Visual representation of which parts are AI-generated

How It's Created:

1. Create a new canvas matching image dimensions
2. Draw the original image on canvas
3. Loop through all analyzed regions
4. For each region:
    - If AI: Draw semi-transparent red rectangle

- If Real: Draw semi-transparent green rectangle
- Add border for clarity

5. Convert canvas to downloadable image

Color Coding:

- Red overlay = AI-generated region
- Green overlay = Real photo region
- Opacity set to 50% so original image is still visible

Feature 4: Downloadable Report

Report Contents:

```
==========================================
DETECTOO AI IMAGE DETECTION REPORT
==========================================

Overall Verdict: AI GENERATED / REAL IMAGE
Confidence: 87%
File Name: example.jpg
File Size: 256 KB
Processing Time: 1543ms

Analysis Methods:
- Pixel Pattern Analysis
- Frequency Analysis
- Edge Detection

DETAILED METRICS:
----------------
Frequency Entropy: 65.3
Color Variance: 72.8
Edge Consistency: 45.2
Noise Level: 23.7

REGION ANALYSIS:
----------------
Region 1 (0, 0): AI GENERATED - 78% confidence
Region 2 (80, 0): REAL - 65% confidence
[... all regions listed ...]
```

Algorithm Explanation

Main Detection Algorithm (Simplified)

Current Implementation (Demonstration):

```javascript
// Generate random result for demonstration
const isAI = Math.random() > 0.45;
const confidence = Math.random() * 0.25 + (isAI ? 0.65 : 0.55);
```

What This Means:

- 55% chance image is classified as AI
- Confidence ranges from 55-90%
- This is a DEMO implementation for educational purposes

Region Analysis Algorithm (Actual Analysis):

```
javascript

// For each 80x80 pixel region:
1. Extract RGB data for all pixels
2. Calculate uniformity:
   uniformity = 0
   For each pixel:
     uniformity += |Red - Green| + |Green - Blue|
   uniformity = uniformity / total_pixels

3. Classify region:
   If uniformity < 20: AI-generated (too smooth)
   If uniformity >= 20: Real (natural variation)

4. Assign confidence:
   If AI: 70-100%
   If Real: 30-70%
```

Real-World Algorithm (What Would Be Used in Production):

A production system would use:

1. **Convolutional Neural Networks (CNN)** - Deep learning model trained on thousands of AI vs real images
2. **Frequency Domain Analysis** - Check patterns in DCT/FFT coefficients
3. **EXIF Data Analysis** - Check camera metadata
4. **Noise Pattern Analysis** - Detect sensor noise patterns
5. **Artifact Detection** - Look for compression artifacts typical of AI generation

---

User Interface Pages

1. Home Page

Components:

- Hero section with gradient background
- "Get Started" button
- Feature cards:
    - Regional Detection
    - Advanced Analysis
    - Instant Results
    - Free & Open Source
- Statistics display (optional)
- Call-to-action section

Design Features:

- Gradient backgrounds (purple to blue)
- Card hover animations
- Responsive layout
- Modern, clean design

---

2. Detector Page (Main Feature)

Layout Sections:

Upload Area:

- Drag-and-drop zone
- Click to upload button
- File type indication
- Visual upload icon

Analysis Display:

- Uploaded image preview
- Loading spinner during processing
- Results dashboard with:
    - Verdict badge (AI/Real)
    - Confidence meter
    - Detailed metrics grid

- File information

Action Buttons:

- View Heatmap
- Download Report
- Analyze Another Image

History Sidebar:

- Last 10 analyses
- Quick access to previous results
- Timestamp for each analysis

---

3. About Page

Content:

- Project mission and vision
- Technology explanation
- How it works section
- Development team information
- Open source commitment
- Educational focus

Key Sections:

- Mission statement
- Technical approach
- Team credits
- Future roadmap

---

4. Contact Page

Features:

- Contact form with fields:
    - Name input

    - Email input
    - Message textarea
- Submit button
- Team contact information
- GitHub repository link
- Response time indication

Form Validation:

- Checks for empty fields
- Validates email format
- Shows success/error messages

---

5. Support/FAQ Page

FAQ Topics Covered:

1. **How accurate is Detectoo?**
    - 85% accuracy claim
    - Educational disclaimer
    - Methodology explanation
2. **Is my data safe?**
    - Client-side processing
    - No server uploads
    - Privacy guarantee
3. **Is it really free?**
    - 100% free confirmation
    - No hidden costs
    - Educational mission
4. **What formats are supported?**
    - JPG/JPEG, PNG, WebP, GIF
    - 25MB file size limit
5. **What do metrics mean?**
    - Frequency Entropy explanation

- Color Variance details
- Edge Consistency info
- Noise Level description

6. **Can I analyze multiple images?**
    - Unlimited analysis
    - History feature
7. **What are the limitations?**
    - Cannot detect AI-assisted editing
    - Compression affects accuracy
    - Screenshots may give false positives
    - Emerging AI generators not covered

Tips Section:

- Use original images
- Higher resolution = better accuracy
- Check heatmap
- Consider confidence scores

Important Functions

1. handleImageUpload()

Purpose: Handles when user selects an image file

Steps:

1. Gets file from input element
2. Validates it's an image file
3. Creates FileReader to read the file
4. Converts file to base64 string
5. Sets image state
6. Automatically starts analysis

Code Signature:

```
javascript
```

```
const handleImageUpload = async (e) => {
  // Get file from input
  // Validate file type
  // Read file
  // Set image state
  // Call analyzeImage()
}
```

---

2. analyzeImage()

Purpose: Performs the AI detection analysis

Process:

1. Sets loading state to true
2. Clears previous results
3. Generates overall verdict (random for demo)
4. Calls generateRegions() for detailed analysis
5. Calculates metrics
6. Creates result object with all data
7. Updates history
8. Sets loading to false

Timing: Takes 2 seconds (simulated processing time)

---

3. generateRegions()

Purpose: Analyzes image region by region

Algorithm:

1. Creates temporary canvas
2. Loads image onto canvas
3. Divides image into 80x80 regions
4. For each region:
    - Extract pixel data using getImageData()
    - Calculate color uniformity

- Determine if AI or Real
- Calculate confidence percentage

5. Returns array of region objects

Output Format:

```javascript
{
  id: 0,
  x: 0,
  y: 0,
  width: 80,
  height: 80,
  isAI: true,
  confidence: 87
}
```

---

4. createHeatmapCanvas()

Purpose: Generates visual heatmap overlay

Process:

1. Creates canvas matching image size
2. Draws original image
3. Overlays colored rectangles on regions:
   - Red (with opacity) for AI regions
   - Green (with opacity) for Real regions
4. Adds borders to rectangles
5. Converts canvas to image data URL
6. Stores for display

---

5. downloadReport()

Purpose: Creates downloadable text report

Report Structure:

1. Header with title
2. Overall verdict and confidence
3. File information
4. Analysis methods used
5. Detailed metrics
6. Region-by-region breakdown
7. Disclaimer

File Format: Plain text (.txt) **Download Method:** Creates blob and triggers download

---

6. handleReset()

Purpose: Clears all data for new analysis

Actions:

- Clears image
- Clears results
- Clears errors
- Hides heatmap
- Resets regions
- Returns to initial state

How to Run the Project

Prerequisites

You need to have installed:

1. **Node.js** (version 14 or higher)
    - Download from nodejs.org
    - Includes npm (package manager)
2. **Text Editor**
    - VS Code (recommended)
    - Sublime Text
    - Any code editor

3. **Web Browser**
    - Chrome, Firefox, Safari, or Edge

Installation Steps

Step 1: Clone or Download the Project

```bash
# If using Git:
git clone https://github.com/sumitbolla7/detectoo.git

# Or download ZIP from GitHub and extract
```

Step 2: Navigate to Project Folder

```bash
cd detectoo
```

Step 3: Install Dependencies

```bash
npm install
```

This command:

- Reads package.json
- Downloads React and React-DOM
- Downloads React Scripts
- Installs all required packages
- Creates node_modules folder

Step 4: Start Development Server

```bash
npm start
```

This command:

- Starts a local web server

- Opens browser automatically
- Runs on http://localhost:3000
- Enables hot-reloading (changes reflect immediately)

Step 5: Access the Application

- Browser opens automatically
- If not, manually visit http://localhost:3000
- Application is now running!

Building for Production

Create Optimized Build:

```bash
npm run build
```

This creates:

- Optimized production files
- Minified JavaScript
- Compressed CSS
- Ready-to-deploy files in `build/` folder

Deployment Configuration

Netlify Deployment

Configuration File: netlify.toml

Build Settings:

```toml
[build]
  command = "npm run build"  # Command to build the project
  publish = "build"          # Folder to serve
```

Routing Configuration:

```toml
```

```
[[redirects]]
  from = "/*"          # All routes
  to = "/index.html"     # Redirect to index.html
  status = 200         # Success status
```

Why This Redirect?

- React is a Single Page Application (SPA)
- All routing happens client-side
- Server must serve index.html for all routes
- React Router then handles the actual page

Deployment Steps (Netlify)

1. **Push code to GitHub**
2. **Connect to Netlify:**
   - Go to netlify.com
   - Click "New site from Git"
   - Select GitHub repository
3. **Configure build:**
   - Build command: `npm run build`
   - Publish directory: `build`
4. **Deploy:**
   - Netlify automatically builds and deploys
   - Provides a unique URL
   - Auto-deploys on every Git push

Limitations

Current Implementation Limitations

1. Demonstration Algorithm

- Uses random generation for overall verdict
- Not a real AI detection model
- For educational/demonstration purposes only

2. Region Analysis Simplicity

- Simple uniformity calculation
- Doesn't use machine learning
- May not be accurate for real-world use

3. No Backend

- All processing happens in browser
- No database for storing results
- No user authentication

4. File Size Limits

- Very large images may slow down browser
- No optimization for huge files

5. No EXIF Analysis

- Doesn't check camera metadata
- Misses important detection clues

What a Production System Would Need

1. Machine Learning Model

- CNN trained on millions of images
- Transfer learning from pre-trained models
- Regular updates as AI generators evolve

2. Backend Infrastructure

- Server-side processing for heavy computations
- Database for result storage
- API for mobile apps

3. Advanced Algorithms

- Frequency domain analysis
- GAN fingerprint detection
- Multi-model ensemble

- EXIF metadata parsing

4. Security Features

- Rate limiting
- API authentication
- Input sanitization

5. User Features

- User accounts
- Analysis history
- Batch processing
- API access

Understanding the Code Flow

Complete User Journey

```
User Opens App
    ↓
index.html loads
    ↓
index.jsx renders App component
    ↓
App.jsx displays Home page
    ↓
User clicks "Detector" in navigation
    ↓
currentPage state changes to 'detector'
    ↓
renderDetector() function executes
    ↓
Upload area is displayed
    ↓
User selects/drops image file
    ↓
handleImageUpload() triggered
    ↓
FileReader reads the file
```

↓

Image converted to base64

↓

Image state updated

↓

Image displayed on screen

↓

analyzeImage() automatically called

↓

Loading state set to true (spinner shows)

↓

2-second delay (simulating processing)

↓

generateRegions() called

↓

Image divided into 80x80 regions

↓

Each region analyzed for uniformity

↓

Region verdicts calculated

↓

Overall verdict generated (random)

↓

Metrics calculated

↓

Result object created

↓

Result state updated

↓

History updated (keeps last 10)

↓

Loading state set to false

↓

Results displayed to user

↓

User can:

- View heatmap visualization
- Download text report
- Analyze another image
- Check previous analyses in history

Technical Concepts Used

1. React Hooks

useState:

- Manages component state
- Re-renders component when state changes
- Example: `const [image, setImage] = useState(null);`

2. Canvas API

Purpose: Pixel-level image manipulation

Key Methods:

- `getContext('2d')` - Get drawing context
- `drawImage()` - Draw image on canvas
- `getImageData()` - Extract pixel data
- `fillRect()` - Draw rectangles
- `toDataURL()` - Convert to image data

3. FileReader API

Purpose: Read file contents in browser

Process:

```javascript
const reader = new FileReader();
reader.onload = (event) => {
  // event.target.result contains file data
};
reader.readAsDataURL(file); // Triggers onload when done
```

4. Base64 Encoding

What It Is:

- Way to represent binary data as text
- Images converted to long strings
- Can be embedded directly in HTML/CSS

Example:

```
...
```

5. Inline Styling

Why Used:

- All styles in JavaScript
- No separate CSS files
- Dynamic styling based on state
- Component-scoped styles

Example:

```javascript
const styles = {
  container: {
    padding: '20px',
    background: '#fff'
  }
};
```

Performance Considerations

Optimizations Used

1. Lazy Loading

- Only one page rendered at a time
- Conditional rendering based on currentPage

2. Region Size

- 80x80 pixels is a good balance
- Smaller = more accurate but slower
- Larger = faster but less detailed

3. History Limit

- Only keeps last 10 results
- Prevents memory bloat

4. Simulated Delay

- 2-second setTimeout
- Gives user feedback
- Prevents instant results (feels more real)

Potential Improvements

1. Web Workers

- Move heavy computation off main thread
- Keep UI responsive during analysis

2. Image Compression

- Resize very large images
- Faster processing

3. Progressive Results

- Show results as regions complete
- Better user experience

4. Caching

- Cache analyzed images
- Instant re-analysis

Security & Privacy

Privacy Features

1. Client-Side Processing

- All analysis happens in browser
- Images never leave user's device
- No server uploads

2. No Data Storage

- Results stored only in memory
- Cleared when page is closed
- No cookies or local storage

3. No Tracking

- No analytics
- No user tracking
- Completely private

Security Best Practices

1. Input Validation

- Checks file type before processing
- Prevents non-image uploads

2. Error Handling

- Try-catch blocks for safety
- Graceful error messages

3. No External Dependencies

- Only React libraries
- Reduces attack surface

Future Enhancements

Possible Improvements

1. Real AI Model Integration

- Integrate TensorFlow.js
- Use pre-trained CNN model
- Real detection accuracy

2. Batch Processing

- Analyze multiple images at once
- Progress bar for batches

3. API Integration

- Backend API for advanced features
- Database for results
- User accounts

4. Advanced Visualizations

- Interactive heatmap with tooltips
- Zoom functionality
- 3D visualizations

5. Mobile App

- React Native version
- Camera integration
- On-device processing

6. Export Features

- PDF reports with charts
- CSV data export
- Share results

7. Comparison Mode

- Compare multiple images side-by-side
- Highlight differences

8. Educational Mode

- Step-by-step analysis explanation
- Interactive tutorials
- Visual algorithm explanation

Common Issues & Solutions

Issue 1: "npm: command not found"

Solution: Install Node.js from nodejs.org

Issue 2: Port 3000 already in use

Solution: Kill process on port 3000 or use different port

```bash
# Kill process (Mac/Linux)
lsof -ti:3000 | xargs kill

# Use different port
PORT=3001 npm start
```

Issue 3: Build fails

Solution: Delete node_modules and reinstall

```bash
rm -rf node_modules
npm install
npm start
```

Issue 4: Heatmap not showing

Solution:

- Ensure image is fully loaded
- Check browser console for errors
- Try smaller image

Issue 5: Image upload not working

Solution:

- Check file size (should be < 25MB)
- Ensure file is an image format
- Try different image

Conclusion

Detectoo is an educational web application that demonstrates AI image detection concepts using React and browser-based image processing. While the current implementation uses simplified algorithms for demonstration, the project structure and UI provide a solid foundation for integrating real machine learning models.

The project showcases:

- Modern React development patterns
- Client-side image processing
- Responsive UI design
- Single-page application architecture
- Cloud deployment with Netlify

Educational Value:

- Learn React fundamentals
- Understand Canvas API
- Practice file handling
- Learn about AI detection concepts
- Experience full-stack project structure

Real-World Applications:

- Journalism verification
- Social media fact-checking
- Content moderation
- Academic research
- Digital forensics

Made by: Sumit - 9527352323

Project Credits:

- Development Team: Anisha Sathe, Sneha Pawar, Shravani Sonawane
- Institution: College of Engineering, Pune

- Purpose: Educational & Research
- License: Open Source

Additional Resources

Learn More About:

1. **React:**
    - Official Docs: https://react.dev
    - React Hooks: https://react.dev/reference/react
2. **Canvas API:**
    - MDN Guide: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
3. **AI Image Detection:**
    - Research Papers on GAN Detection
    - CVPR Conference Papers
    - ArXiv.org for latest research
4. **Image Processing:**
    - Digital Image Processing by Gonzalez
    - OpenCV for advanced techniques

End of Documentation

This comprehensive guide explains every aspect of the Detectoo project from basic concepts to advanced implementation details. Use it as a reference for understanding, modifying, or expanding the project.