Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management
- **Team Members:** Pala Yogendra-Team Leader
  Reddy Manoj Kumar-Team member
  Parisineti Venkata Nisyal-Team member

2. Project Overview

Purpose:

The primary purpose of the project "Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management" is to provide an automated, fast, and accessible tool for detecting poultry diseases using deep learning techniques. It aims to support farmers and poultry caretakers in identifying infections early, minimizing bird mortality, reducing economic losses, and improving overall poultry farm health.

Goals of the Project:

- Develop an AI-powered system using transfer learning to classify major poultry diseases from images.
- Enable early and accurate disease diagnosis to improve poultry health and reduce spread.
- Provide a simple and intuitive web interface for easy image upload and result viewing by farmers.
- Reduce dependency on manual inspections and accelerate the diagnosis process.
- Ensure affordability and scalability so the system can be used in both small-scale and commercial poultry farms.
- Support data-driven poultry management by collecting and analyzing image-based disease patterns over time.

Features:

- **AI-Powered Disease Detection**

Uses transfer learning and pre-trained deep learning models (e.g., ResNet) to detect poultry diseases from images.

- **Web-Based Interface**

User-friendly and accessible through web browsers, allowing farmers to interact with the system easily.

- **Fast Prediction Results**

Provides instant disease classification with high accuracy, helping users take timely action.

- **Image Upload System**

Supports uploading poultry images in formats like JPG or PNG for analysis.

- **Visual Result Display**

Displays the prediction along with a representative image of the diagnosed disease for clarity.

- **Feedback Collection Module**

Allows users to submit feedback for continuous system improvement.

Core Functionalities:

- **User Registration & Confirmation**

Secure user signup and verification system.

- **Disease Classification**

Analyzes uploaded images and returns disease type (e.g., Coccidiosis, Salmonella, etc.).

- **Admin Portal**

Enables administrators to manage disease info, update the model, and monitor usage.

- **Disaster & Disease Info Display**

Shares updated resources on poultry disease management and outbreaks.

3. Architecture

Frontend Architecture (HTML, CSS, JavaScript)

- The frontend is designed using **HTML** for layout, **CSS** for styling, and **JavaScript** for interactivity.
- Key Pages:
  - **Home Page:** Introduces the platform and allows users to navigate.
  - **Prediction Page:** Enables users to upload poultry images for analysis.
- **JavaScript Functions**:
  - Handles file selection and validation
  - Sends image data to the backend
  - Dynamically displays prediction results

Backend Architecture (Flask - Python)

- The backend is built using **Flask**, a lightweight Python framework that handles HTTP requests and routes.
- **Core Endpoints:**
  - /predict – Accepts image uploads and sends them to the AI model
  - /upload – Manages temporary image storage and preprocessing
  - /result – Returns classification results (e.g., Coccidiosis, Healthy, etc.)
- **AI Model Integration**:
  - A pretrained CNN model (e.g., ResNet, EfficientNet using transfer learning) processes the uploaded image
  - Prediction output is returned to the frontend as a string and displayed with an optional image
- All data is processed **in-memory** during runtime, without persistent storage.

.

4. Setup Instructions

- **Prerequisites:**

Frontend (HTML, CSS, JavaScript)

- **HTML5** – Markup language for building structured web pages
- **CSS3** – Styling of pages and layout responsiveness
- **JavaScript (ES6+)** – Handles interactivity, file upload, and dynamic content
- **Bootstrap (optional)** – For responsive design and styling (if used)

Backend (Flask - Python)

- **Python 3.x** – Core programming language for backend logic and AI integration
- **Flask** – Lightweight web framework to handle routes and serve API endpoints
- **Werkzeug** – Used internally by Flask for request/response handling
- **Flask-CORS** – Enables cross-origin requests from frontend to backend
- **Flask-Uploads / Flask-WTF** – For handling image file uploads (optional)

AI/Model Dependencies

- **TensorFlow / Keras** – Deep learning framework for building and loading transfer learning models
- **OpenCV / PIL (Pillow)** – Image processing libraries for resizing, normalization, and display
- **NumPy** – Used for numerical operations during image processing
- **Matplotlib (optional)** – For visualization or debugging model outputs

Other Utilities

- **Gunicorn / Waitress** – For deploying the Flask app (production server)
- **Jupyter Notebook (optional)** – For model training and experimentation during development
- **scikit-learn** – For performance metrics like confusion matrix, accuracy, etc.

- **Installation:**

Clone the Repository
git clone https://github.com/your-username/project-name.git

Navigate into the Project Directory
cd project-name

Create and Activate Virtual Environment
python -m venv venv
source venv/bin/activate *(Windows: venv\Scripts\activate)*

Install Dependencies
pip install -r requirements.txt

Run the Flask App
flask run

5. Folder Structure

- **Client:**

  templates/

- Contains a **single HTML file: index.html**
- This file serves as the **main page** for the entire application and includes:
- The **home section** with a welcome message and introduction
- The **image upload section** for disease prediction
- A **feedback section** for users to submit comments or suggestions
- All the UI interactions happen within this single HTML file, likely managed using JavaScript or separate page sections.

  static/

- Contains **images** used in the frontend interface, such as:
- coccidiosis.jpg, salmonella.jpg, healthy.jpg, etc.
- These images are used for visual representation of poultry diseases and results.
- No separate CSS or JS files are used if you're keeping everything inline or minimal.

Server:

Backend Organization (Flask-Based)
The backend is developed using Flask and is kept clean and minimal, handling image uploads and predictions directly from a single script.

1. **app.py**

- The main backend file that runs the Flask application.
- It contains:
- Route definitions like / (for loading the main page) and /predict (for processing uploaded images).
- Logic to receive the uploaded poultry image from the frontend.
- Loads the trained transfer learning model (e.g., .h5 file) directly inside this file.
- Preprocesses the image and performs prediction in the same script.
- Returns the predicted disease label to the frontend**.**

2. model

- May contain the saved deep learning model file (model.h5) used by app.py for prediction.

3. static/uploads

- A temporary folder where uploaded images are saved during prediction, if needed.

6. Running the Application

Frontend Server (Static HTML/JS/CSS)

If you're using plain HTML/CSS/JS (no npm, no React), you don't need to start a frontend server — the HTML page runs directly in a browser.

Option 1: Open manually
Just double-click `index.html` or open it in a browser.

Option 2 (Recommended): Serve locally using Python

Backend Server(Flask)

source venv/bin/activate

export FLASK_APP=app.py

FLASK_ENV=development

flask run

7. API Documentation

/ Home Route

- **Method:** GET
- This route serves the main index.html page from the templates folder.
- It acts as the entry point to the application**.**

2. /predict – Disease Prediction

- **Method:** POST
- Accepts an image file uploaded by the user via the frontend.
- Processes the image using the trained AI model and returns the predicted disease name.
- This is the core functionality of the backend.

8. Authentication

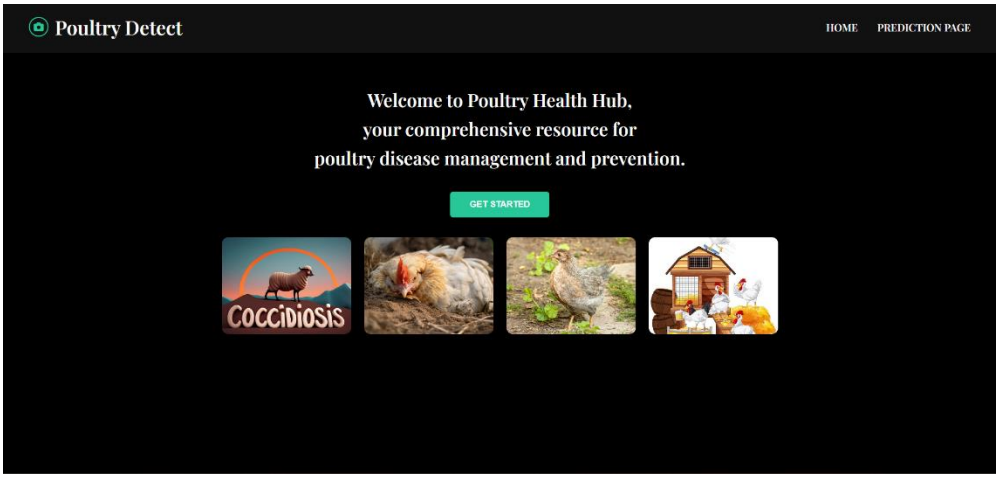
1. User Authentication

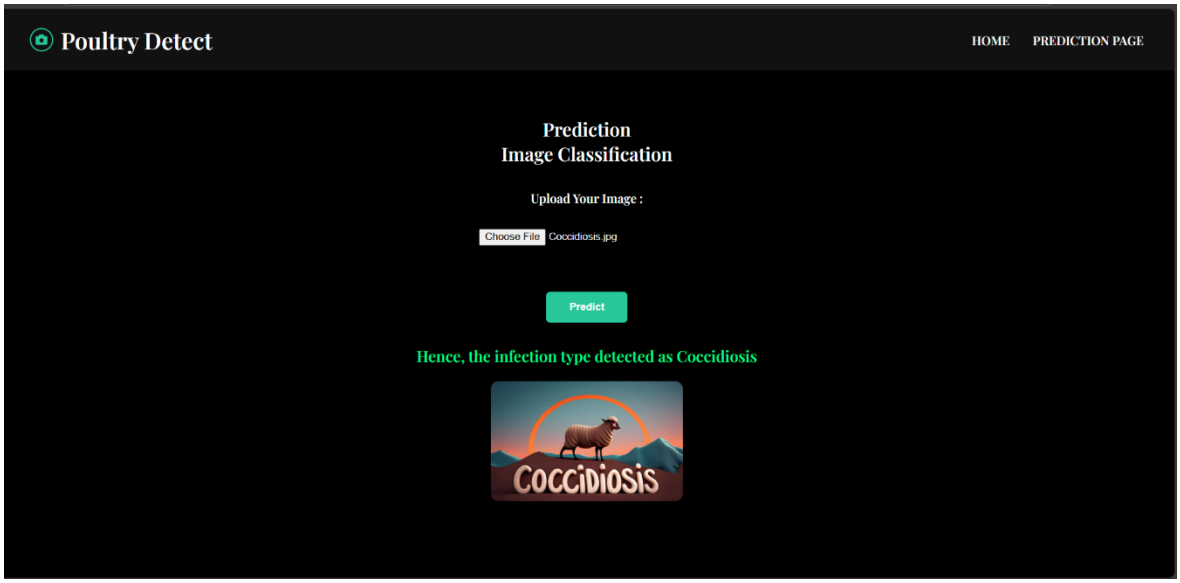
- Users are authenticated through a simple login form built into index.html.
- Upon submitting credentials, the data is sent to the backend via a POST request (e.g., /login route if implemented).
- The backend verifies the username and password against stored values (in memory or file-based if no database is used).

2. Sessions or Tokens

- **If sessions are used:**
  - Flask can store user sessions using the session object.
  - Upon successful login, the user's session is marked as authenticated (session['logged_in'] = True).
  - This session persists until the browser is closed or the user logs out.
- **If tokens (like JWT) are used:**
  - On login, a token is generated and returned to the client.
  - The token is included in subsequent requests (in headers) to validate access.
  - Flask verifies the token on each protected route.

9. User Interface





10. Testing

Strategy and Tools Used

1. Manual Testing

- The project primarily uses **manual testing** to validate its functionality, since it involves image uploads and UI interaction.
- Each feature (image upload, prediction display, navigation) was tested by uploading various poultry images and checking output consistency.
- Edge cases like submitting without a file or uploading an unsupported format were also verified.

2. Test Case-Based Verification

- A structured **test case table** was created to verify system responses.
- Test inputs like coccidiosis.jpg, salmonella.jpg, etc., were used to confirm that the model returns correct predictions.
- Status of each test case (e.g., Passed/Failed) was recorded to ensure quality.

3. Frontend Testing

- Basic frontend testing was done using:
    - Browser developer tools to inspect form validation
    - Manual interaction to verify responsiveness, navigation, and prediction flow
    - Image rendering and layout correctness across screen sizes

4. Model Evaluation (During Development)

- During model training, tools like:
    - **Keras/TensorFlow evaluation metrics** (accuracy, loss)
    - **Confusion matrix and classification report** from scikit-learn
    - **Jupyter Notebook** was used to visualize model performance

11. Screenshots or Demo

DEMO VIDEO LINK:
https://drive.google.com/file/d/1qhNFvb2nCqvwxsYI1GfoYViOE8Hq_5Kb/view?usp=drivesdk

12. Known Issues

1. Image Quality Sensitivity

Low-resolution or unclear images may result in incorrect disease predictions. The model performs best when given clear, focused images.

2. No Feedback Storage

The feedback form accepts user input but does not save it anywhere, as there is no database or file storage mechanism currently implemented.

3. Single HTML File Limitation

All frontend logic and layout are handled in a single `index.html` file. This makes future updates or scaling more difficult to manage.

4. Lack of User Authentication

There is no user login or authentication system in place. All users can access the full functionality without restrictions.

5. Limited Mobile Responsiveness

The current layout may not display correctly on all mobile or smaller screen devices. Some content may overflow or become misaligned.

13. Future Enhancements

1. Multi-Disease Detection

Allow the model to detect and classify multiple infections in a single image if symptoms overlap.

2. Mobile App Integration

Develop a dedicated Android or iOS app to make the system more accessible, especially for rural users.

3. Multi-Language Support

Add support for regional languages in the interface to improve usability for non-English-speaking users.

4. Feedback Storage and Analytics

Enable saving of feedback and use it to generate insights, usage trends, or model improvement suggestions.

5. Real-Time Chat with Experts

Integrate a live chat or message board for users to consult veterinarians or domain experts after prediction.

6. Disease History Tracking

Implement user-based history logs to let farmers track past submissions, predictions, and treatment advice.

7. Improved UI Design

Add animations, tooltips, and modern layout enhancements for better user experience and engagement.