Artificial Intelligence And Machine Learning

Project Documentation

Introduction

Project Title: Poultry Disease Classification System

The Poultry Disease Classification System is an AI-powered solution designed to support poultry farmers and veterinarians in identifying common poultry diseases through machine learning and image analysis. By leveraging deep learning techniques on poultry image data, the system provides quick, reliable disease predictions to enhance poultry health management, minimize losses, and improve farm productivity.

The system focuses on classifying poultry into four categories based on disease conditions: Salmonella, New Castle Disease, Coccidiosis, and Healthy birds. This web-based platform enables users to upload poultry images and receive immediate diagnostic feedback, thereby assisting in timely intervention and treatment.

Team Members

- **Karri Ratna Amulya [Member 1] – Project Manager**
(Requirement gathering, workflow planning, coordination, and documentation)

- **Kusunuri Navya [Member 2] – Machine Learning Engineer**
(Data preprocessing, model development, training, and optimization)

- **Mithun Kumar [Member 3] – Full Stack Developer**
(Web interface design, API integration, and user experience flow)

- **Priyanshu Raj [Member 4] – Cloud & DevOps Engineer**
(Deployment on cloud platform, CI/CD pipelines, storage solutions, and scaling infrastructure)

Project Overview

The Poultry Disease Classification System is an innovative AI-based tool designed to help detect and classify poultry diseases at an early stage through automated image recognition. The system utilizes transfer learning and convolutional neural networks (CNNs) to process and analyze images of poultry, determining the presence of specific diseases with high accuracy.

By integrating modern machine learning models with a user-friendly web interface, the solution allows farmers, veterinary professionals, and farm managers to:
Upload poultry images via a web application
Receive instant predictions of disease class
Use the insights to take preventive or corrective measures promptly

The system supports the broader goal of improving poultry health management by enabling data-driven decisions, reducing mortality rates, and supporting sustainable poultry farming practices.

By combining deep learning with accessible technology, the Poultry Disease Classification System transforms raw image data into actionable intelligence, promoting better animal health and farm productivity.

ARCHITECTURE

Frontend Architecture – HTML

This is an HTML file for the user interface of the Poultry Disease Classification System, allowing users to upload poultry images (and optionally provide metadata) for disease prediction. The form submits the data to a Flask backend, which processes the image, performs the classification, and returns the result to be displayed on the page.

Structure Breakdown

<!DOCTYPE html> & <html>
• Defines the document type and starts the HTML structure.

<head>
• Contains metadata:

- Character encoding is set to UTF-8.
- The page title is "Poultry Disease Classifier".

<body>
• May include a background image (e.g., farm or poultry-related scene) via a URL or CSS styling.
• Contains the main content wrapped inside a (e.g., with a class like container or form-wrapper for future CSS styling).

Form Purpose

The <form> is used to collect user input and send it to the Flask backend:

• **action="{{ url_for('predict') }}"** → Sends form data to the Flask function named predict.
• **method="post"** → Sends data securely using the POST method.
• **enctype="multipart/form-data"** → Enables image file uploads.

Input Fields

The form collects the following:

Image Upload
• A file input field:
<input type="file" name="file" accept="image/*" required>
• Ensures the uploaded file is an image and is mandatory.

Optional Metadata (if used)
• Dropdown: Bird Age Group
<select name="age_group">
<option value="chick">Chick</option>
<option value="adult">Adult</option>
</select>

Submit Button

• A button styled using Bootstrap or CSS:
<button type="submit" class="btn btn-primary">Predict</button>
• Triggers form submission for disease classification.

Prediction Output Display

• Below the form, a <h2> element displays the prediction:
<h2 style="background-color:rgba(0,0,0,0.6); color:white; padding:10px;">{{ prediction_text }}</h2>
• Uses Jinja2 templating ({{ prediction_text }}) to dynamically render the result from Flask.

Styling Notes

• The form is centered using tags or CSS.
• Minimal styling except for:

- Background image for a professional appearance.

- Inline CSS (or external style.css) for the prediction output box (semi-transparent black background, white text).
- Bootstrap classes for buttons and optional inputs for consistency and better UI.

Functionality Summary

Feature	Description
Input collection	Collects poultry image (and optional metadata) from the user.
Data submission	Sends the image to Flask for disease classification.
Result display	Displays the predicted disease class dynamically below the form.
Template integration	Uses Jinja2 ({{ prediction_text }}) for dynamic content with Flask.

Backend Architecture – Flask

This Python-based backend handles **poultry disease classification** by integrating a machine learning image classification model with a web interface built using **Flask**. It acts as the bridge between the user's uploaded image and the pre-trained deep learning model, delivering predictions in real time.

Purpose

The backend:
• Receives the poultry image (and optional metadata) uploaded via the HTML form.
• Preprocesses and formats the image to match the input shape and requirements of the trained model.
• Loads and uses a pre-trained **convolutional neural network (CNN)** model for classification.
• Returns the predicted disease class to the user interface for display.

Key Functionalities

1. **Homepage Display**
• When users visit the root URL (/), Flask serves an HTML page (index.html).
• This page contains the form for uploading poultry images (and optionally selecting metadata).

2.**Form Submission and Input Handling**
• The form is submitted to the /predict route using the POST method.
• The backend receives the image file and optional form inputs.
• The image is validated (checked for file type, presence) and preprocessed (e.g., resized, normalized) as expected by the model.

3.Model Loading and Prediction

• The trained model is loaded using pickle, TensorFlow, or Keras (e.g., model.h5 or model.pkl).
• The preprocessed image is passed to the model to generate a prediction.
• The model outputs a class index or label (e.g., 0=Salmonella, 1=New Castle, 2=Coccidiosis, 3=Healthy), which is mapped to a readable disease name.

4. Displaying the Result

• The prediction result is passed back to the frontend via render_template() and displayed dynamically on the same page.
• If an error occurs (e.g., missing file, invalid input, model issue), a meaningful error message is displayed instead.

Files Involved

• **model.h5 / model.pkl** → The trained CNN model file used for predictions.
• **index.html** → The form-based frontend template that collects the image and displays results.
• **Flask app (e.g., app.py)** → The Python script handling routing, image processing, prediction logic, and rendering output.
• *(Optional)* **encoder.pkl / label_map.json** → Used if your model requires mapping numeric outputs to class names.

How It Works in the Project

• The user uploads an image and submits the form via the frontend.
• Flask receives the image, preprocesses it, and feeds it to the ML model.
• The model generates a disease class prediction.
• Flask sends the result back to the frontend for display.
• The entire process happens in real-time within the browser, offering immediate feedback to the user.

Role in Poultry Disease Classification

This backend is essential for:
• Connecting the user interface with the deep learning model logic.
• Enabling real-time disease detection based on uploaded images.
• Managing the entire data flow from image submission to prediction display.

Machine Learning Model

• A pre-trained deep learning model (e.g., model.h5 or model.pkl) is used for poultry disease classification.
• The model is built using Convolutional Neural Network (CNN) architecture or a transfer learning approach (such as MobileNetV2, ResNet50, or VGG16), trained on labeled poultry

images.
• The model is executed via a Python script (Flask backend) that receives image input, preprocesses it, and returns the predicted disease class.
• Input features are derived directly from the uploaded image after preprocessing:

- Resized image data (e.g., 224×224 RGB pixels)
- Normalized pixel values (e.g., scaled between 0 and 1)
- *(Optional)* Metadata features (e.g., age group) if included in the model design

Database

In the Poultry Disease Classification System, the database is used to store and manage details of each disease prediction made through the web interface. This supports logging, auditing, and potential future analytics on disease trends.

Type of Database

• The system uses MongoDB, a NoSQL document-oriented database ideal for storing structured and semi-structured records, including image metadata and prediction logs.

How the Database is Used

When a user uploads a poultry image and the Flask backend generates a disease prediction:

• The input data (e.g., filename, optional metadata like age group)
• The predicted disease class (e.g., Salmonella, New Castle, Coccidiosis, Healthy)
• The timestamp (date and time of prediction)

are saved into MongoDB as a single document.

Structure of Each Record

Each document stored in the database contains:

```
{
  "inputData": {
    "filename": "uploaded_image.jpg",
    "ageGroup": "Chick" // If collected
  },
  "predictedDisease": "Coccidiosis",
  "timestamp": "2025-06-28T12:34:56"
}
```

This structure allows easy tracking of all predictions, supports reviewing historical cases, and enables future features like data visualization and reporting.

Database Integration

• The Flask backend uses PyMongo or a similar library to connect to MongoDB.
• After each prediction, the system automatically saves the record to the database.
• The database can be queried to:

- Retrieve all past predictions
- Build dashboards to analyze disease patterns over time
- Export logs for further study

Role in Poultry Disease Classification

The machine learning model provides the intelligence for automated disease detection, while the database ensures that all interactions are logged for transparency, learning, and potential integration into farm management systems or health tracking dashboards.

Setup Instructions

This section outlines the tools and steps required to install and run the Poultry Disease Classification System on a local machine.

Prerequisites

Before setting up the project, ensure that the following software dependencies are installed:

System Requirements

• **Operating System:** Windows 10/11, macOS, or Linux
• **Internet connection** (for downloading packages and optionally connecting to MongoDB Atlas if used)

Software Dependencies

1. Python (3.x)
• Required to run the Flask backend and execute the deep learning model (model.h5 or model.pkl).
• Handles data processing using libraries like TensorFlow/Keras, NumPy, and OpenCV.

2. Flask
• A lightweight Python web framework used to build the backend server that receives images and returns the predicted disease class.

3. TensorFlow / Keras (or PyTorch, if applicable)
• Used to load and run the trained deep learning model for poultry disease classification.

4. **NumPy and OpenCV**
• NumPy: For numerical array handling and image data processing.
• OpenCV: For image reading, resizing, and preprocessing.

5. **MongoDB (optional)**
• A NoSQL database to store uploaded image details, predictions, and timestamps for logging and analytics.
• Can be set up locally or accessed via MongoDB Atlas.

6. HTML/CSS (Frontend)
• The HTML form acts as the user interface for uploading poultry images and displaying predictions.

7. Git (optional)
• Useful for version control and collaboration if hosting the project on GitHub.

8. Code Editor
• Any text editor or IDE such as VS Code, PyCharm, or Jupyter Notebook is recommended for coding, debugging, and managing files.

Installation

This step-by-step guide helps you set up the Poultry Disease Classification System on your local machine.

Step 1: Organize Project Files

Create a main project folder (e.g., PoultryDiseaseClassifier) and place these essential files inside it:

• app.py → Flask backend application
• model.h5 or model.pkl → Trained deep learning model
• templates/index.html → HTML frontend form
• static/ → (Optional) CSS, background images, or JavaScript files
• predict.py → (Optional) If model logic is separated

This structure ensures Flask can locate your frontend and model files properly.

Step 2: Set Up Python and Dependencies

Ensure Python 3.x is installed. Then, in your terminal:

pip install flask tensorflow numpy opencv-python pymongo

(If using PyTorch instead of TensorFlow, replace tensorflow with torch and torchvision.)

These tools allow the system to receive images, process them, and make predictions.

Step 3: Configure Flask Application

In app.py, ensure routes and logic are set up to:

• Load the HTML form at the home route (/).
• Handle image uploads at the /predict route.
• Preprocess the image, load the model, and return the predicted disease class.
• Render results dynamically on the frontend.

Verify your templates/ folder is in the right place so Flask can render index.html.

Step 4: Set Up MongoDB (Optional)

If your project logs each prediction:

• Create a MongoDB database (e.g., poultry_logs).
• Each document will store:

- The input image filename (or metadata)
- The predicted disease
- The timestamp of the prediction

This setup supports building history logs, reports, or analytics dashboards later.

Step 5: Run Poultry Disease Classifier Locally

1. Open terminal or command prompt.
2. Navigate to your project directory.
3. Run the Flask app:

```
python app.py
```

4. Open a browser and go to:

http://localhost:5000

You'll see the Poultry Disease Classification Form. Upload an image, submit, and view the predicted disease class instantly.

Folder Structure

The Poultry Disease Classification System is organized into two main parts:
1. The **client (frontend)** — for user interaction (image upload + result display)
2. The **server (backend)** — for handling logic, prediction, and communication with the machine learning model

Client – Frontend Structure (HTML-based UI)

The client side provides the user interface where users upload poultry images and view disease predictions.

1. **templates/**

• This folder is used by Flask to serve HTML pages.
• Contains the main user interface file:

- **index.html** → HTML form where users upload poultry images and optionally select metadata (e.g., age group). It also displays the predicted disease returned from the backend.

2.**static/** *(optional)*

• Used for custom CSS, images, or JavaScript files to style and enhance the frontend.
• Example structure:

- **static/css/style.css** → For custom styling of the form and output display.
- **static/images/** → To store background images or icons (e.g., poultry farm background).

Server – Backend Structure (Flask Application)

The server side processes uploaded images, runs the machine learning model, and sends results back to the frontend.

Root files:
• **app.py** → The main Flask application script.

- Loads the trained model (e.g., model.h5 or model.pkl).
- Defines routes:
    - / → Renders the form page (index.html).
    - /predict → Handles image submission, processes the image, and returns the predicted disease.

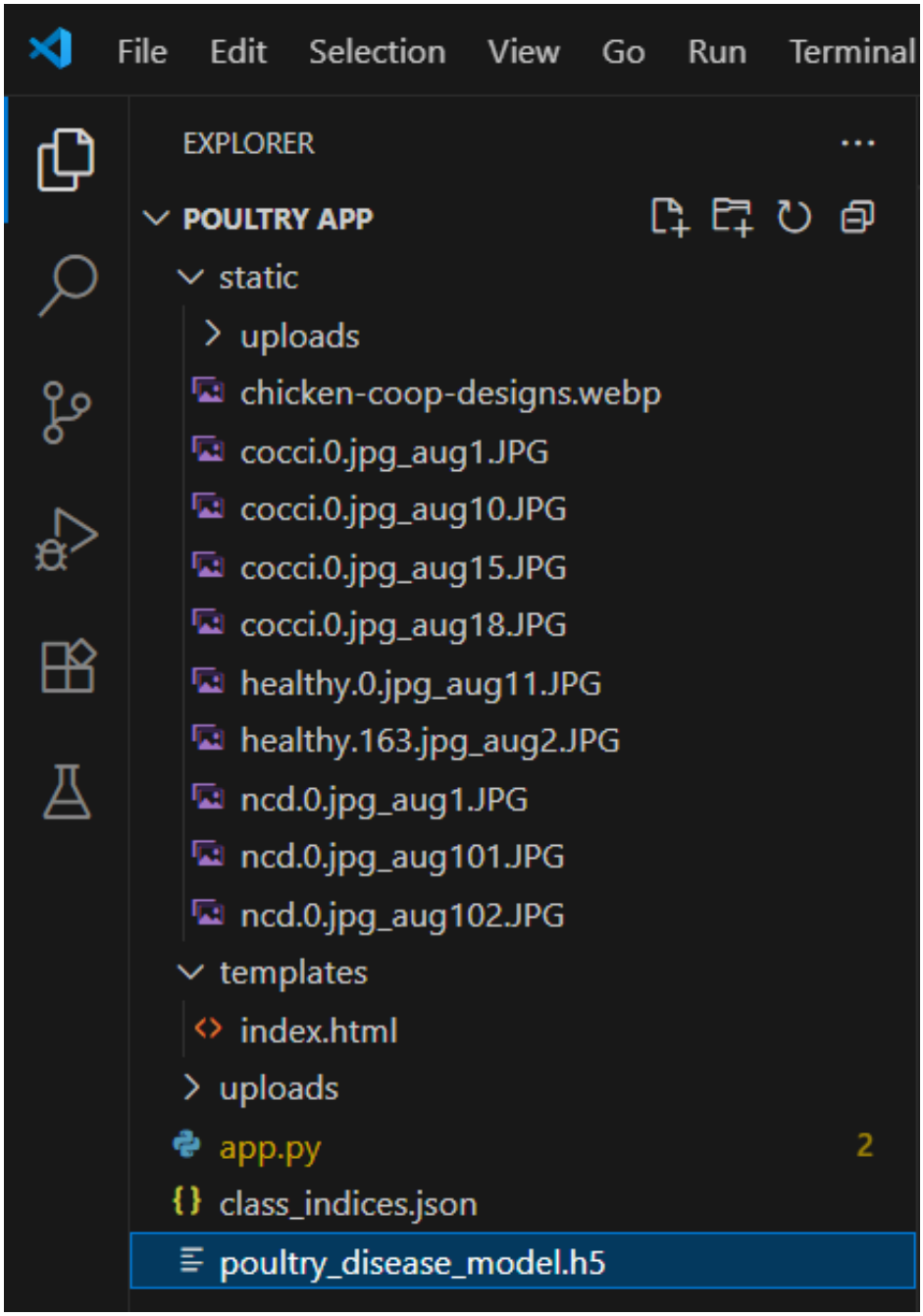
• **model.h5** or **model.pkl** → The pre-trained deep learning model for classifying poultry diseases.

Optional files:
• **predict.py** → Separate module for model preprocessing and prediction logic (for cleaner code structure).
• **.env** → Stores environment variables such as the MongoDB connection URI if database integration is used.
• **requirements.txt** → Contains the list of Python packages required to run the app (Flask, TensorFlow/Keras, NumPy, OpenCV, PyMongo, etc.).

Example Structure

```
PoultryDiseaseClassifier/
|
 ├── app.py
 ├── model.h5              # or model.pkl
 ├── predict.py            # (optional)
 ├── requirements.txt
 ├── .env                  # (optional, for environment variables)
|
 ├── templates/
|   └── index.html
|
 ├── static/
|    ├── css/
|   |   └── style.css      # (optional custom styles)
|   └── images/
|       └── background.jpg  # (optional background image)
```

Structure:


File Edit Selection View Go Run Terminal
EXPLORER
POULTRY APP
static
uploads
chicken-coop-designs.webp
cocci.0.jpg_aug1.JPG
cocci.0.jpg_aug10.JPG
cocci.0.jpg_aug15.JPG
cocci.0.jpg_aug18.JPG
healthy.0.jpg_aug11.JPG
healthy.163.jpg_aug2.JPG
ncd.0.jpg_aug1.JPG
ncd.0.jpg_aug101.JPG
ncd.0.jpg_aug102.JPG
templates
index.html
uploads
app.py
2
class_indices.json
poultry_disease_model.h5

Running the Application

To run the **Poultry Disease Classification** application locally, both the frontend and backend must be started. This enables seamless interaction between the user interface and the machine learning model for real-time disease prediction.

Frontend (User Interface)

The frontend of the Poultry Disease Classification system is built using static HTML pages served through Flask (not React). Therefore:

- There is no need to run npm start.
- The HTML file (e.g., index.html) is located inside the templates**/** folder.
- Flask automatically renders the frontend when the backend is started and you navigate to http://localhost:5000.

Frontend is served directly by Flask — no separate startup required.

Backend (Flask Server with ML Model)

The backend is the core of the Poultry Disease Classification system. It handles user input (e.g., images or form data), applies the machine learning model, and returns the predicted disease class.

To run the backend:

1. Open a terminal or command prompt.

2. Navigate to the project folder where app.py is located.

3. Run the Flask server using:

   python app.py

4. Once the server is running, open your browser and go to:

   http://localhost:5000

5. You will see the Poultry Disease Classification form.

6. After uploading an image or entering relevant details, submit the form — the model will generate and display the prediction.

Summary

Component	**Startup Required**	**How to Run**
Frontend	No (served by Flask)	Automatically available at localhost:5000
Backend	Yes	Run python app.py to start the Flask server

API Documentation

The Poultry Disease Classification backend exposes two primary endpoints through the Flask server. These endpoints handle web page rendering and disease prediction based on user input.

1. Home Endpoint

- **Endpoint:** /
- **Method:** GET
- **Function:**
    - Renders and displays the HTML form where users can upload poultry images (or provide related data if applicable).
    - No parameters are required from the client for this endpoint.
    - Acts as the landing page of the application.

2. Prediction Endpoint

- **Endpoint:** /predict
- **Method:** POST
- **Function:**
    - Receives data submitted from the HTML form (e.g., image file).
    - Processes the input and passes it to the pre-trained machine learning model.
    - Returns the predicted disease class (e.g., *Salmonella*, *New Castle Disease*, *Coccidiosis*, *Healthy*) back to the same HTML page for display.

Request Parameters

The input form sends the following parameters to the backend:

Parameter	Type	Description
file	file	The poultry image uploaded by the user (usually .jpg, .png, etc.)
(Optional) additional_data	string / numeric	Any extra form data if included (e.g., age, environment details)

Responses

- **On success:**
  The predicted disease category is displayed on the same page, typically below the form.
- **On failure:**
  An error message is shown (e.g., if the image is missing, invalid, or processing fails).

Summary

- The API design is simple and form-based, perfect for a single-page web application.
- It enables real-time interaction between the user and the machine learning model using just two endpoints.
- The structure can be easily extended in the future for features like:
  - Upload history
  - API integration for mobile apps
  - Logging and analytics

Authentication

Current State

In its current form, the Poultry Disease Classification project does not include user authentication or authorization mechanisms. The application is designed as a simple utility that allows any user to upload poultry images (or related data) and receive a disease prediction from the machine learning model without needing to log in or create an account.

Reason for No Authentication

- The primary goal of the project is to demonstrate poultry disease classification using machine learning.
- It is intended as a prototype or academic project, where open access makes testing and demonstration easier.
- Since there is no user account system, no sessions, tokens, or login processes are implemented.

Future Scope for Authentication

If the project evolves into a production-level tool or a platform for farmers, veterinarians, or agricultural agencies, the following authentication methods can be considered:

1. User Login System

- Implement user accounts so farmers or vets can save diagnosis history.
- Use email/password combinations securely stored in a database like MongoDB or MySQL.

2.Session Management

- Use Flask sessions or JWT (JSON Web Tokens) to manage login states.
- Allow personalized dashboards or history tracking.

3. Role-Based Access Control

- Define roles such as farmer, vet, admin, or researcher.
- Restrict access to features like detailed analytics, disease trends, or user management based on roles.

4. Token-Based Authentication (Advanced)

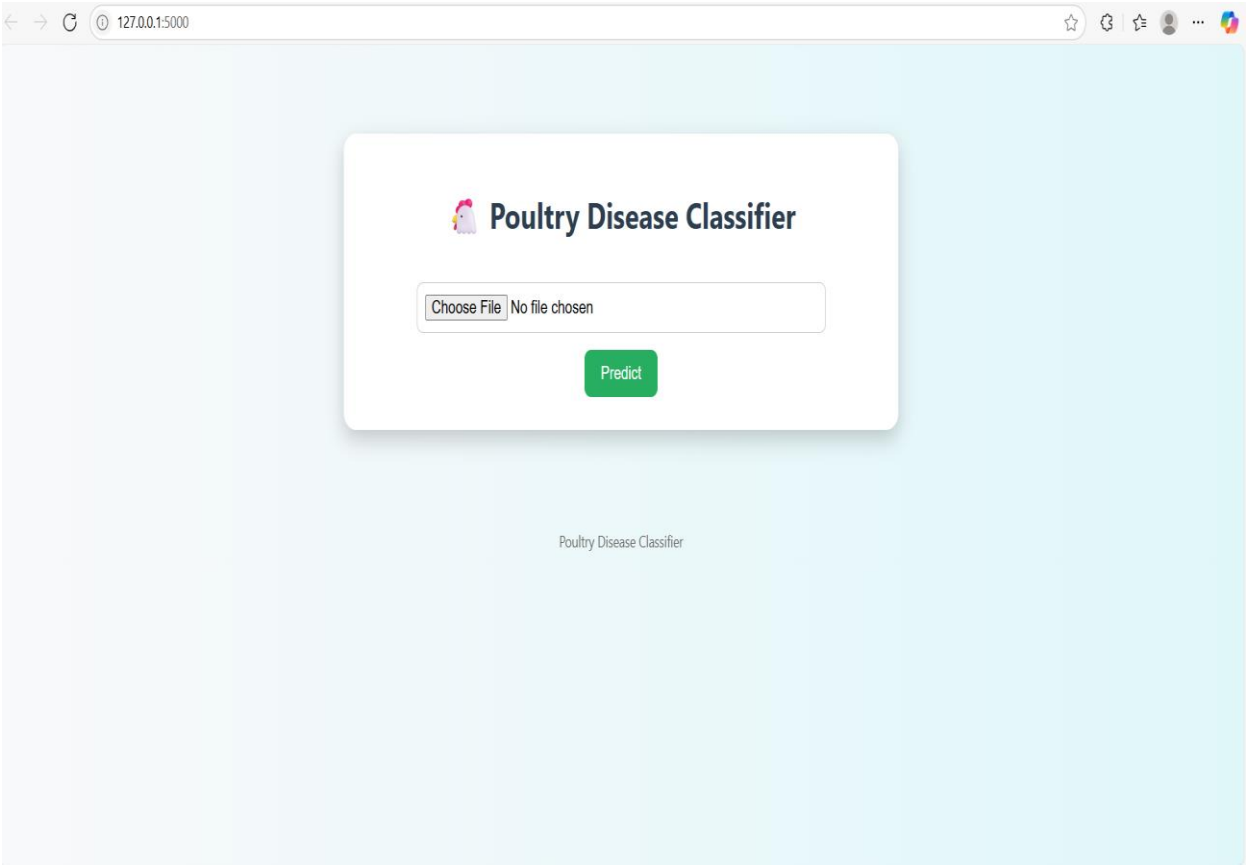
- Use JWT tokens for secure, stateless authentication — especially if you integrate a mobile app or a frontend framework like React in the future.

Summary

Feature	Current Status
User Login	Not Implemented
Session Handling	Not Implemented
Token Auth (JWT)	Not Implemented
Open Access	Enabled

- The project is currently open for all users but is scalable for secure authentication features if required in future development.

User Interface



Testing

In the Poultry Disease Classification project, testing is focused on ensuring the accuracy of the machine learning model and the correct functioning of the Flask backend. The testing strategy includes both manual testing and basic programmatic checks to verify that the application performs as expected under different scenarios.

1. Model Testing

- The machine learning model was tested using standard classification evaluation metrics such as:
    - **Accuracy** – to measure the proportion of correctly classified instances.
    - **Precision, Recall, F1-Score** – to evaluate performance across individual disease classes.
    - **Confusion Matrix** – to visualize the model's classification behavior on test data.
- Multiple pre-trained models (e.g., MobileNetV2, ResNet50, VGG16) were compared to select the best-performing model for poultry disease classification.
- The final model was saved as model.pkl (or similar format) and tested on unseen validation/test images before being integrated into the application.

2. Backend Testing

- The Flask backend was tested manually by:
    - Uploading different poultry images (representing all disease categories and healthy cases).
    - Verifying that the predicted disease class is correctly displayed on the web page.
    - Checking the backend's response to invalid inputs (e.g., no file selected, unsupported file type).
- Error handling was tested to ensure the application shows meaningful error messages when invalid or missing inputs are submitted.

3. Integration Testing

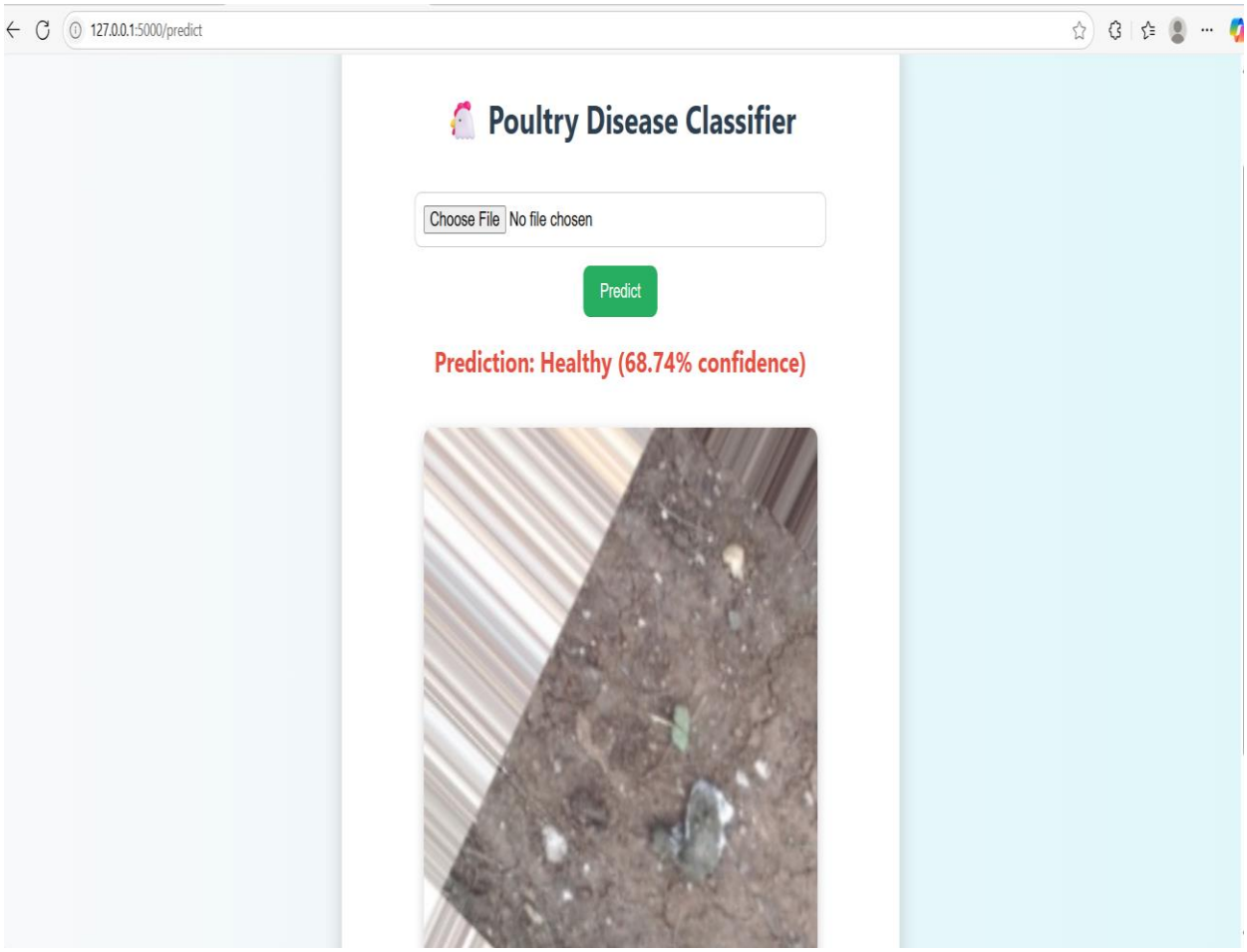
- The full workflow — from image upload to prediction display — was tested as an end-to-end flow.
- Tests included:
    - Ensuring the frontend sends image data correctly to the backend.
    - Verifying that the model returns the correct disease class.
    - Confirming that the prediction is displayed properly on the frontend.

Testing Tools Used

- Jupyter Notebook / Python scripts — for training, evaluating, and comparing machine learning models.
- Manual browser testing — to check the frontend-backend integration and ensure smooth user experience.
- No automated unit tests (e.g., pytest, unittest) were implemented, as the current project is focused on prototype functionality.

➢ The testing approach ensures the core functionality of the system works reliably, and the design can be extended with automated testing frameworks in future iterations.

Screenshots or Demo



Known Issues

The Poultry Disease Classification application is functional and stable for basic use, but as with any prototype or academic project, there are a few known issues and limitations that users or developers should be aware of.

1. No User Authentication

- The application does not include login or user verification.
- Anyone accessing the web interface can upload images and use the prediction feature.
- This limits the ability to provide user-specific diagnosis history, logs, or access control.

2. Limited Input Validation

- While basic checks exist (e.g., requiring an image file), the app does not:
    - Prevent uploading non-image files (e.g., someone could try uploading a text file).
    - Validate file size or resolution.
    - Handle repetitive or duplicate uploads gracefully.
- More robust input validation could improve security and reliability.

3. No Automated Testing

- All testing has been done manually so far.
- The lack of automated unit tests or integration tests means future updates could introduce bugs without being immediately detected.

4. No Model Retraining Pipeline

- The machine learning model (model.pkl) is static.
- There is no automated pipeline to retrain or update the model as new poultry disease data becomes available, which may impact prediction accuracy over time.

5.Limited Error Handling

- The app shows basic error messages in the browser (e.g., no file selected).

- Internal errors (e.g., failure to load model, file I/O errors) may not be communicated clearly to the user, making debugging harder.

6.Basic UI Design

- The frontend is simple and functional, but:
    - It may not display well on small screens or mobile devices.
    - It lacks modern design elements, responsiveness, and accessibility features.

7. No Logging or History Dashboard

- The application does not track or display past predictions or uploads.
- There is no interface for users to view historical results or logs.

8. Deployment Limitation

- The app is currently designed to run locally on localhost.
- It has not been configured or tested for deployment on platforms like Heroku, Render, or AWS.

These issues are common in early-stage prototypes and can be addressed in future enhancements as the system evolves toward production readiness.

Future Enhancements

While the Poultry Disease Classification application successfully identifies poultry diseases based on user input using machine learning, there is great potential to expand and enhance the system with new features and capabilities. Below are some ideas for future improvements:

1. User Authentication System

- Implement login and registration functionality.
- Allow users (e.g., farmers, vets) to save diagnosis history.
- Introduce role-based access (e.g., admin, farmer, veterinarian).

2. Responsive and Modern UI

- Redesign the frontend using React.js, Bootstrap, or Tailwind CSS.
- Make the interface mobile-friendly for field use.
- Add image previews and interactive visualizations (e.g., charts for disease trends).

3. Prediction History and Analytics Dashboard

- Display past prediction logs from a database (e.g., MongoDB, SQLite).
- Show disease occurrence trends over time or by farm/location.
- Add filters (e.g., date, disease type, location).

4. Integration with Farm Data Systems

- Connect to farm management tools to fetch additional animal health data.
- Provide more context-aware disease predictions using environment/symptom records.

5. Model Improvement and Auto-Retraining

- Add a feature to retrain the model periodically using newly collected labeled images.
- Explore advanced models (e.g., EfficientNet, Vision Transformers) for better accuracy.
- Apply hyperparameter tuning and cross-validation for optimal performance.

6. Notification or Alert System

- Notify users (via email, SMS, or app alert) when high-risk diseases are detected.
- Help farmers take immediate preventive measures.

7. Deployment on Cloud Platforms

- Deploy the app on platforms like Render, Heroku, or AWS.
- Provide public access through a secure domain and HTTPS.

8. API Development

- Convert the prediction logic into a REST API.
- Allow external tools, mobile apps, or agricultural platforms to consume predictions as a service.

9. Location-Based Disease Risk Analysis

- Include location data (e.g., farm coordinates).
- Predict and map regional disease risk levels based on historical data.

10. **Multi-Language Support**

- Add translations for the UI text to support non-English-speaking users (e.g., local languages used by farmers).
- Increase accessibility for a wider audience.

➢ These enhancements can help transition your app from a prototype into a production-ready system for real-world farm use.

THANK YOU