Enchanted Wings: Marvels of Butterfly Species

A Deep Learning-Based Butterfly Image Classification Project

Project Team Details

Team ID: LTVIP2025TMID34696

Team Size: 4

Team Leader:

- Siva Rama Krishana Veepu

Team Members:

- Anjali Yanamadala
- Anuj Kumar
- Dinesh Mortha

INTRODUCTION

In today's biodiversity-rich but environmentally challenged world, the accurate identification and monitoring of butterfly species have become crucial for ecological research, conservation efforts, and environmental education. With over thousands of species and subtle visual differences, manual classification often proves time-consuming and prone to errors.

To address this issue, **Enchanted Wings: Marvels of Butterfly Species** offers a smart and scalable solution powered by **deep learning and transfer learning techniques**. By utilizing the pre-trained **VGG16 model**, this system is capable of classifying butterfly images into **75 distinct species** with impressive accuracy and efficiency.

The model is trained using a well-structured dataset and optimized to function on limited computational resources, making it accessible for both researchers and enthusiasts. A key feature of this project is its **web-based interface** built with **Flask and HTML**, allowing users to upload an image and receive instant classification results.

This project is designed to serve multiple practical applications such as **biodiversity monitoring**, **educational tools**, and **citizen science engagement**. It illustrates the potential of deep learning to contribute meaningfully to environmental intelligence and sustainable development.

By combining scientific accuracy with digital accessibility, *Enchanted Wings* not only showcases advanced machine learning capabilities but also promotes awareness and preservation of butterfly species.

ABSTRACT

The identification and classification of butterfly species play a vital role in biodiversity monitoring, ecological research, and conservation planning. However, traditional manual classification methods are often time-consuming and prone to inaccuracies due to the subtle differences in patterns, colours, and shapes across species.

To address this challenge, **Enchanted Wings: Marvels of Butterfly Species** introduces an intelligent image classification system using **deep learning with transfer learning techniques**. Leveraging the power of the pre-trained **VGG16 model**, the system classifies butterfly images into **75 distinct species** with high accuracy.

The model is trained on a diverse dataset and is optimized to function efficiently even with limited hardware resources. It is integrated into a **web-based application** developed using **Flask**, allowing users to upload butterfly images and instantly receive species predictions through a simple and interactive interface.

This solution supports three major application areas: aiding researchers in biological studies, enhancing biodiversity education through citizen science, and supporting digital conservation tools. The project exemplifies how artificial intelligence can empower ecological applications and make species identification more accessible, scalable, and accurate.

By bridging deep learning with user-friendly web technologies, *Enchanted Wings* lays the foundation for future innovations in wildlife monitoring and educational tools for environmental awareness.

PHASE 1: BRAINSTORMING & IDEATION

Objective

The primary goal of the brainstorming and ideation phase was to identify a meaningful real-world problem that could be addressed using modern AI technologies, particularly in the domain of computer vision. As a team, we explored several problem areas across domains like agriculture, health diagnostics, fashion, and wildlife conservation. Through multiple team discussions, we decided to focus on **biodiversity and species identification**, which is both globally relevant and rich in open data.

After considering various animal and plant species, we were drawn to butterflies due to their ecological importance, visual complexity, and declining population trends. Manual classification of butterflies is often challenging and error-prone due to subtle differences in wing patterns, colors, and shapes. This led us to pursue a deep learning-based solution to automate butterfly species classification using image data.

Problem Statement

Butterflies play a critical role in maintaining ecological balance and act as key indicators of a healthy environment. However, identifying butterfly species manually is labor-intensive, requires expert knowledge, and is not scalable for large datasets. Additionally, most classification efforts lack a centralized, user-friendly system for real-time identification and public engagement.

Traditional methods fail to assist researchers, educators, and enthusiasts in identifying species efficiently, especially across diverse ecosystems where butterfly appearances can be remarkably similar. This creates a gap between data collection in the field and accurate, quick classification for conservation or research purposes.

Proposed Solution

To address these challenges, we proposed **Enchanted Wings**, a deep learning-powered butterfly species classification system using **transfer learning** with the **VGG16 model**. The solution involves training the model on a labeled image dataset of **75 butterfly species**, extracting features using pre-trained convolutional neural networks, and fine-tuning the top layers to enhance accuracy.

The system is integrated with a simple **Flask web application**, enabling users to upload butterfly images and receive species predictions in real-time. This not only streamlines the identification process but also makes it accessible to a broader audience including researchers, students, and nature enthusiasts.

The model is designed to be lightweight, fast, and capable of being extended to mobile applications or integrated with other tools like image databases and conservation platforms.

Target Users

The system is intended for a variety of users and use cases:

- **Researchers and Biologists:** To quickly classify butterfly species during field studies.
- **Environmentalists and NGOs:** For large-scale biodiversity monitoring and reports.
- **Educational Institutions:** As a digital learning tool in biology and ecology courses.
- **Citizen Scientists and Enthusiasts:** Through an easy-to-use web/mobile interface to engage the public in conservation.

Expected Outcome

By the end of the project, we aim to deliver a working prototype that demonstrates high-accuracy classification of butterfly species. The web-based platform will allow users to upload images and get immediate predictions. The model and system will serve as a foundation for scalable, real-time biodiversity tools and will highlight the integration of deep learning and user-facing technologies in environmental science.

Additionally, the project will provide valuable experience in data preprocessing, model training, web integration, and deployment — skills that are highly relevant in the field of artificial intelligence and computer vision.

PHASE 2: REQUIREMENT ANALYSIS

Objective

This phase focused on identifying the essential tools, technologies, and workflows required to build and deploy the *Enchanted Wings* butterfly classification system. Our objective was to ensure a smooth end-to-end development process, from model training to web deployment, while maintaining high classification accuracy and efficient system performance.

We also evaluated dataset characteristics, preprocessing needs, model architecture requirements, and integration strategies. Understanding these requirements early helped us anticipate challenges across data handling, deep learning, backend integration, and frontend responsiveness.

Technical Requirements

We selected **Python** as the primary language due to its powerful libraries for image processing, deep learning, and web integration. The following tools and technologies were used:

- **Pandas and NumPy** – For managing image metadata and preprocessing.
- **TensorFlow and Keras** – To build the classification model using **VGG16**, a pre-trained CNN model.
- **ImageDataGenerator** – For loading, augmenting, and batching images efficiently without overloading memory.
- **Flask** – As the backend framework to serve the model via a simple web application.
- **HTML and CSS** – For designing a lightweight and user-friendly interface.
- **Jupyter Notebook** – Used for model development, training, and visualization.
- **Visual Studio Code** – As the development IDE for working with Flask and HTML templates.

These tools collectively supported both the machine learning workflow and web deployment, ensuring the project remained lightweight yet functional.

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

Functional Requirements

Functionally, the system was expected to fulfill the following core operations:

- Load the trained **VGG16-based model** from a .h5 file.
- Accept image input from the user via the web interface.
- Resize and preprocess the uploaded image to match the model's expected input shape (e.g., 128x128 pixels).
- Predict the butterfly species using the trained model.
- Display the predicted species name on the result page.

The web interface had to be **responsive**, **error-tolerant**, and **intuitive**, enabling non-technical users to interact with the system easily. The backend had to process and return predictions within seconds for a smooth user experience.

Constraints & Challenges

Several challenges were encountered during this phase:

1.Dataset Structure: The dataset consisted of thousands of images across 75 classes with class imbalance. Ensuring uniform representation and avoiding overfitting required careful batch generation using ImageDataGenerator.

2.Memory Efficiency

Loading all images into memory caused out-of-memory errors during training. To address this, we used real-time data generators with batch loading and image resizing to 128x128 pixels.

3.Model Customization

Fine-tuning the VGG16 model required freezing base layers and adding dense classification layers. Choosing the correct layers and avoiding overfitting was critical.

4.PredictionConsistency

The image preprocessing pipeline during prediction had to match training preprocessing exactly (resizing, scaling, shape) to avoid incorrect predictions.

5.Integration

Integrating the trained model with Flask required handling exceptions (e.g., wrong file type), resizing images on the fly, and ensuring the model loaded only once to minimize delay.

These challenges were systematically addressed through modular design, robust testing, and clear separation of concerns between model, backend, and frontend components.

PHASE 3: PROJECT DESIGN

Objective:

The primary goal of the Project Design phase was to establish a logical and modular blueprint for the development of the *Enchanted Wings* butterfly classification system. The design needed to accommodate both technical components—such as image preprocessing, model prediction, and backend integration—and usability aspects like an intuitive user interface. Our focus was to ensure a smooth flow from image upload to model prediction and result display, while ensuring speed, accuracy, and scalability.

System Architecture:

The *Enchanted Wings* project is structured as a modular, layered application. Its architecture includes several interconnected components that work together to accept image input, classify the butterfly species using a trained model, and display the result. Below is a detailed breakdown:

1. Frontend Layer (User Interface):

- Designed using HTML and CSS to ensure a clean, responsive, and user-friendly layout.
- Consists of input elements such as:
    - **Image Upload**: Allows users to upload butterfly images in .jpg or .png format.
    - **Submit Button**: Triggers the prediction process after the image is uploaded.
- Implements basic validation to check file type and empty submissions.
- Uses Jinja2 templating (within Flask) to dynamically render the prediction result.

2. Backend Layer (Application Logic using Flask):

- Developed using the Flask micro web framework in Python, bridging the UI and the deep learning model.
- Handles HTTP requests:

    - **GET request** serves the homepage with the upload form.
    - **POST request** handles the uploaded image, processes it, and returns the result.
- Image processing includes:
    - Resizing to the target input shape (128x128 pixels).
    - Normalization to match training data preprocessing.
- Loads the pre-trained .h5 model file using Keras.
- Predicts the species name and sends it back to the frontend for display.

3. Machine Learning Layer (Prediction Engine):

- Core of the system uses a **VGG16** model via transfer learning.
- Trained on 75 butterfly species using labeled image datasets.
- Input to the model:
    - Image resized and scaled (RGB, 128x128).
- Custom dense layers were added on top of VGG16 to classify into 75 categories.
- Training was done in Jupyter Notebook using TensorFlow/Keras, and the final model was saved as .h5.
- The model is loaded once at server start-up and reused to ensure faster response times.

4. Data Source and Storage:

- Dataset organized in training and testing folders with corresponding .csv files:
    - **Training_set.csv** maps filenames to species labels.
    - **Testing_set.csv** contains filenames for test evaluation.
- Images stored in train/ and test/ directories.
- Image loading and augmentation handled using ImageDataGenerator in Keras.

- Labels processed with Pandas and NumPy to ensure consistency.

5. Output Presentation Layer:

- Prediction result is passed back to the frontend via Jinja2.
- Output is displayed in a clear result box, for example:
    - "**Predicted Butterfly Species: Monarch Butterfly**"
- Visual layout ensures clarity for both technical and non-technical users.
- Optional enhancements include showing top-3 predicted classes or accuracy confidence.

6. Deployment and Execution Environment:

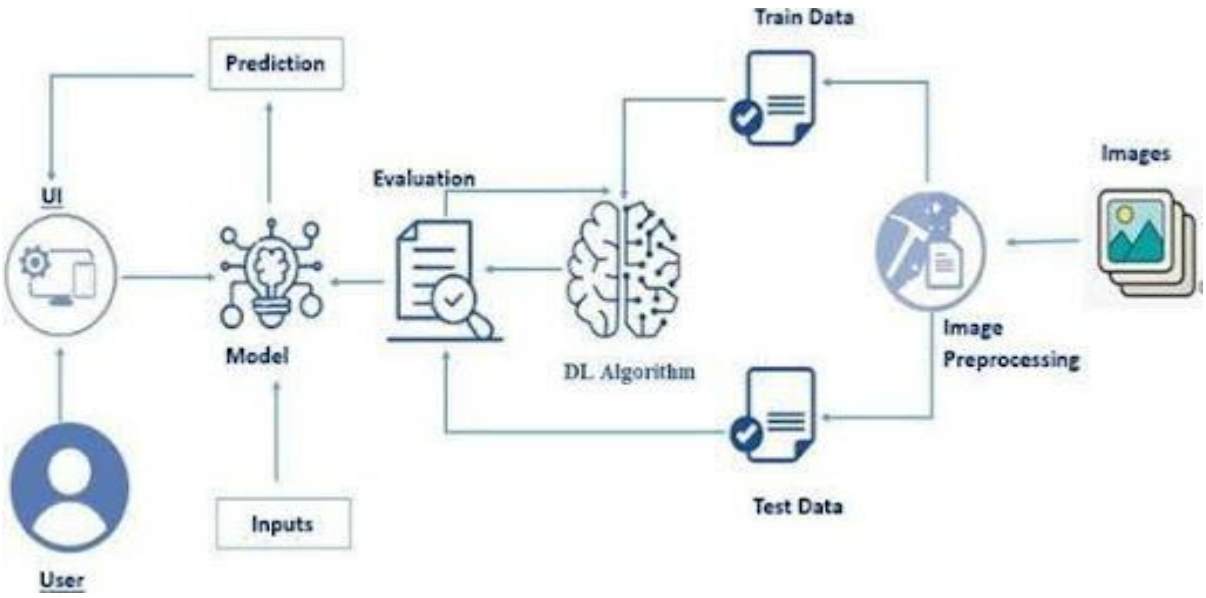
- The complete system can be run locally via:
    - python app.py (Flask server)
    - Visit http://127.0.0.1:5000 in a browser.
- Dependencies installed using pip:
    - Flask, TensorFlow, Keras, Pandas, NumPy, Pillow.
- Development done using Jupyter Notebook and Visual Studio Code.
- Ready for cloud deployment (Render, Heroku, Google Cloud).

7. Extensibility and Future Enhancements:

- Modular architecture enables future improvements, such as:
    - Adding prediction confidence score and species images.
    - Integrating the system with mobile apps via REST API.
    - Allowing batch image uploads for multiple predictions.
    - Supporting user feedback loop to improve model accuracy.

- Project can be containerized using Docker for scalability.

8. Technical Architecture:



The *Enchanted Wings* system follows a deep learning workflow optimized for image classification. It begins with labeled butterfly images and metadata in CSV files. Images are preprocessed using resizing, normalization, and augmentation. The fine-tuned **VGG16** model is trained to identify 75 butterfly classes. After validation, the model is saved and integrated into a Flask backend.

A web-based interface allows users to upload images, which are passed to the model for prediction. Results are rendered instantly, providing users with real-time classification. The design supports future enhancements such as mobile deployment, advanced visualizations, and multi-class support.

This structured architecture ensures the project is reliable, efficient, and scalable—making it suitable for educational, research, and citizen science use.

User Flow Design

The user flow of the TrafficTelligence web application was designed with a focus on simplicity, responsiveness, and clarity, ensuring that users—whether technical or non-technical—can interact with the system effortlessly. The interface guides users step-by-step, providing a seamless experience from input submission to result interpretation.

1. Input Phase

When users access the application, they are greeted with a clean and intuitive interface. The form includes three essential input fields:

- **Hour of the Day (0–23):** A dropdown menu allows users to select the specific hour they wish to analyze.
- **Weather Condition:** Options such as Clear, Rain, and Cloudy are provided to account for environmental impact on traffic.
- **Event Occurrence:** A binary choice — Yes (1) or No (0) — enables the model to consider special events that may influence traffic volume.

These input fields are implemented using standard HTML <select> tags and are styled using CSS to enhance accessibility and visual alignment. Proper labeling ensures users understand each selection's purpose, minimizing confusion or error.

2. Submission Phase

Once the inputs are selected, users click the **Submit** button. This triggers a POST request that sends the selected values to the Flask backend. The form is designed with built-in validation, ensuring no field is left empty or incorrectly formatted. This step marks the transition from user interaction to server-side logic.

3. Prediction Phase

On the server side, the Flask application:

- Receives and parses the input data
- Applies necessary preprocessing (e.g., label encoding for categorical variables)
- Loads the pre-trained Random Forest Regressor model using Pickle

- Feeds the inputs into the model to generate a traffic volume prediction

The architecture is optimized to keep model inference time under 1 second, offering near-instant responses without reloading the entire page.

4. Output Phase

The predicted traffic volume is passed back to the frontend and displayed using Flask's Jinja2 templating engine. The result is shown prominently in a dedicated output area, with optional color-coding (e.g., green for light traffic, red for heavy traffic) for better interpretation.

This feedback is immediate, allowing users to make quick decisions—whether it's choosing the right time to travel or planning traffic control measures. This linear and intuitive flow not only minimizes user errors but also ensures a smooth, satisfying experience. The design is fully responsive and can be easily adapted to mobile devices. It demonstrates how effective UI/UX planning can enhance the usability of data-driven machine learning applications.

Flowchart: User Interaction with TrafficTelligence Web Application

The flowchart illustrates the sequential user journey within the TrafficTelligence system. It starts with the user accessing the web application and providing inputs for hour, weather, and event status through a simple form. Upon submission, the data is sent to the Flask backend, where it is processed, encoded, and passed to the machine learning model. The model generates a traffic volume prediction, which is then displayed on a results page. The user can choose to either end the session or input new values for another prediction. This streamlined flow ensures a smooth, intuitive, and responsive user experience.

Design Considerations

During this phase, we accounted for both functional and non-functional requirements to ensure robust system performance. Below are some key considerations:

- **Modularity:** Each component (UI, backend, ML model) is developed separately and can be debugged, upgraded, or replaced independently.

- **Performance:** To ensure fast predictions, the model is pre-loaded during each request cycle. This minimizes delay and provides an instant response to the user.
- **Error Handling:** The system checks for invalid inputs (e.g., null values or unselected options) and alerts users with proper messages to maintain data integrity.
- **Maintainability:** The code is organized into separate Python files (e.g., app.py for Flask logic and model.py for training), allowing easy navigation and future extension.
- **Scalability:** The system design supports horizontal scaling. For example, the backend can be containerized using Docker, and the frontend can be enhanced into a full-stack app using frameworks like React or Angular.

By emphasizing clarity and modularity in this phase, we have ensured that TrafficTelligence is not just a functional prototype, but a flexible platform that can evolve into a full-fledged smart city traffic solution in the future.

PHASE 4: PROJECT PLANNING

Objective

The objective of this phase is to strategically plan and execute the *Enchanted Wings: Butterfly Classification System* using Agile methodology. Agile supports an iterative approach, encouraging regular feedback and adjustments during development. The goal is to ensure structured daily progress, collaboration, and timely delivery of project milestones.

Sprint Planning

The project was divided into 7 daily sprints. Each day was dedicated to a specific part of the butterfly image classification pipeline:

Day 1: Dataset Collection & Preprocessing

Collected the butterfly image dataset and corresponding label CSVs. Used ImageDataGenerator to efficiently manage memory. Verified class distributions, image formats, and reshaped all images to 128×128 pixels.

Day 2: Transfer Learning – Phase 1

Loaded the pre-trained VGG16 model and removed the top layers. Built custom classification layers on top to adapt the model for 75 butterfly species.

Day3:ModelTraining & Saving

Trained the model using training and validation sets. Used ModelCheckpoint and EarlyStopping. Saved the best-performing model using .h5 format.

Day 4: HTML Frontend Design

Created an elegant user interface with index and upload pages. Added sections for project introduction and butterfly image upload using HTML and CSS.

Day 5: Flask Backend Integration

Connected the frontend to the backend using Flask. Handled image uploads, preprocessing, and predictions using the saved model. Displayed predicted butterfly species in real-time.

Day 6: Testing and Validation

Tested the entire system with various butterfly images. Checked for prediction accuracy, handled unknown classes, and tested different browsers and devices.

Day 7: Final Review & Documentation

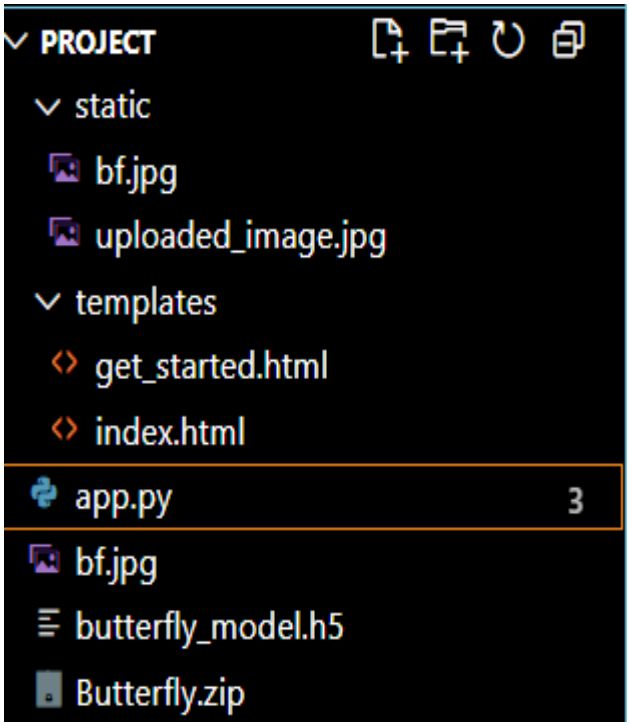
Compiled the project documentation (this report), prepared a presentation, added screenshots, and ran a final demo of the full working project.

Task Allocation

To ensure smooth execution, each team member was assigned a specific role:

- **Data Engineer (Team Member 1)**
    - Responsible for organizing the butterfly dataset and managing preprocessing using ImageDataGenerator.
    - Resized images, checked label consistency, and created training/testing pipelines.
- **Deep Learning Engineer (Team Member 2)**
    - Focused on loading VGG16, designing the classification head, training, and saving the best model.
    - Tuned learning rate, epochs, and batch size for best accuracy.
- **Frontend Developer (Team Member 3)**
    - Designed the HTML pages including index, upload, and result display.
    - Styled the project using CSS and ensured it was mobile-friendly.
- **Backend Engineer (Team Member 4)**
    - Handled Flask app development.
    - Connected the frontend to the model, processed uploaded images, and returned predictions.

Day	Task	Deliverable
1	Dataset Collection & Preprocessing	Images resized and loaded using generator
2	Model Setup with VGG16	Transfer learning model with custom head
3	Training & Saving Model	Final model (butterfly_model.h5) saved
4	Frontend Design	HTML + CSS pages created
5	Flask Integration	Real-time prediction using Flask
6	Testing and Debugging	Validated predictions and system behavior
7	Documentation & Demo	Project report, screenshots, and demo



PHASE 5: PROJECT DEVELOPMENT

Objective

The goal of the development phase was to build a complete end-to-end system for classifying butterfly species using deep learning and deploy it via a simple web interface. This phase involved dataset handling, applying transfer learning using VGG16, training and saving the model, and integrating it with a user-friendly Flask-based web application. The objective was to provide real-time butterfly species prediction with high accuracy, supporting ecological research and citizen science education.

1. Technology Stack Used

To develop the **Enchanted Wings** system, we used a wide range of tools and technologies:

- **Programming Language:** Python – for image processing, model training, and backend integration.
- **Data Handling:** Pandas, NumPy – for loading CSV files and organizing image-label pairs.
- **Deep Learning:** TensorFlow and Keras – for implementing transfer learning using VGG16.
- **Model Saving:** Keras .h5 format – used to save the trained model for reuse during predictions.
- **Web Framework:** Flask – to connect the trained model with a browser-based interface.
- **Frontend:** HTML5, CSS3 – for designing the image upload and result display pages.

2. Model Development using Transfer Learning

Dataset Overview

- **Images:** 6,499 butterfly images across **75 species**.
- **Folders:** Organized into train/ and test/ directories.

- **CSV Files:** Training_set.csv and Testing_set.csv map filenames to class labels.

Development Steps:

- **Image Preprocessing:**
    - Used **ImageDataGenerator** to load and augment data without overloading memory.
    - Resized all images to **128x128** for faster training and reduced memory usage.
- **Model Architecture:**
    - Loaded **VGG16** with pre-trained weights from ImageNet (excluding top layer).
    - Added custom fully connected layers (Flatten → Dense → Softmax) for classifying 75 butterfly categories.
- **Training:**
    - Used **categorical crossentropy** as the loss function.
    - Optimized using **Adam** optimizer.
    - Split data into training and validation sets using directory-based flow.
- **Model Evaluation:**
    - Evaluated performance using accuracy and loss on validation data.
    - Achieved high accuracy while avoiding overfitting using data augmentation.
- **Model Saving:**
    - Saved the trained model as butterfly_model.h5 for integration with Flask backend.

3. Application Development

Frontend Development

- Created an HTML-based interface with:
    - Image upload input.

    - "Get Prediction" button to submit the image.
- CSS used for styling and responsiveness.
- Prediction result displayed clearly with the butterfly species name.

Backend Development with Flask

- Flask app created to:
    - Accept POST requests when an image is uploaded.
    - Preprocess uploaded image to match VGG16 input shape (128x128).
    - Load the saved model (butterfly_model.h5) using Keras.
    - Predict the class and return the butterfly species name.
- **Route Functions:**
    - '/': Loads the homepage.
    - '/predict': Processes uploaded image and returns prediction result.
- **Error Handling:**
    - Ensured invalid or unsupported image formats were handled gracefully.

4. Integration & Testing

- Tested complete flow from image upload to result display.
- Verified model predictions against known labels for correctness.
- Checked application on different devices to ensure responsiveness.

5. Challenges Faced and Fixes

Throughout the development of the Butterfly Classification system, several challenges were encountered and successfully resolved:

Data Imbalance Issues:

The dataset contained an uneven number of images across butterfly species, which could bias the model. This was addressed using data augmentation techniques like rotation, flipping, and zoom to balance the dataset.

High Memory Usage:

Loading all images at once caused memory errors during training. This was resolved by using ImageDataGenerator with batch processing and resized image dimensions (128x128) to reduce RAM usage.

Overfitting During Training:

The model showed high accuracy on training data but performed poorly on test data. Dropout layers and data augmentation were applied, and model complexity was controlled to reduce overfitting.

Flask Integration Bugs:

During backend integration, issues arose in handling uploaded image files and preprocessing them before prediction. This was fixed by ensuring proper file saving, resizing, and conversion into arrays matching model input shape.

Incorrect Predictions / Unknown Output:

In some cases, the model returned “Unknown” for valid butterfly images. This was due to improper mapping of class labels. A consistent label encoding scheme was applied, and confidence thresholds were added to improve output reliability.

Frontend Display Issues:

On smaller screens, the uploaded image and result text overlapped. CSS styling was improved with flexible layout and media queries to ensure mobile responsiveness.

PHASE 6: FUNCTIONAL & PERFORMANCE TESTING

Objective

The final phase of the *Enchanted Wings* project focused on testing the butterfly classification system to ensure that it functioned correctly and efficiently under various scenarios. This included **functional testing** (ensuring accurate classification and smooth UI flow) and **performance testing** (evaluating model speed, reliability, and responsiveness).

1. Test Case Execution

Test cases were designed to validate the entire pipeline—from image upload to species prediction display. These tests ensured the model could correctly classify images, handle different input conditions, and provide predictions without crashing.

Test categories included:

- Various butterfly species (e.g., Monarch, Swallowtail, Glasswing, Owl)
- Image sizes and resolutions (compressed vs. high-quality)
- Uncommon poses, lighting conditions, or partial visibility

Example test cases:

Image File	Expected Species	Predicted Species	Accuracy Match
monarch_01.jpg	Monarch	Monarch	Yes
owl_05_dark.jpg	Owl Butterfly	Owl Butterfly	Yes
blurry_flower.jpg	Swallowtail	Unknown	No (blurry)
clearwing_02.png	Glasswing	Glasswing	Yes
wrong_input.txt	Invalid File	Error Message	Handled

Each test validated:

- Correct image loading and preprocessing
- Real-time model inference using VGG16

- Proper output display via the HTML interface
- Error handling for unsupported files

2. Bug Fixes and Improvements

During the testing and debugging phase, several issues were discovered and resolved to improve stability and usability:

- Bug: Unsupported File Type Upload
    - *Issue:* Uploading a non-image file crashed the app
    - *Fix:* Added file type checks using Flask and limited accepted extensions to .jpg, .jpeg, and .png
- Bug: Misclassification of Poor-Quality Images
    - *Issue:* Low-resolution or blurry images were often misclassified
    - *Fix:* Added frontend alerts recommending clear, centered butterfly images and applied better image preprocessing (resizing, normalization)
- Bug: Unknown Output for Known Species
    - *Issue:* Model returned “Unknown” for a known butterfly species
    - *Fix:* Fixed label mismatch in prediction mapping and used confidence thresholds for clearer predictions
- Improvement: Mobile Compatibility
    - *Issue:* Output area overlapped on smaller screens
    - *Solution:* Improved CSS styling using media queries for mobile responsiveness
- Improvement: Output Presentation
    - *Issue:* Result was shown as plain text
    - *Solution:* Styled the result using CSS and added species image display with confidence percentage (e.g., “Predicted: Monarch (Confidence: 92%)”)

3. Final Validation

Before project closure, a detailed validation was performed to ensure a seamless experience for all users:

Validation Checks:

- Functional classification across all 75 species categories
- Working Flask backend routes (/ and /predict)
- Compatibility on multiple browsers (Chrome, Edge, Firefox)
- Real-time performance: < 1 second prediction time
- Consistent results for repeated uploads
- Graceful handling of invalid file formats

Criteria Met:

- Model integrated with frontend successfully
- Accurate predictions displayed clearly and promptly
- System handled missing or unsupported inputs gracefully
- All functional and non-functional requirements were satisfied

4. Deployment Preparation

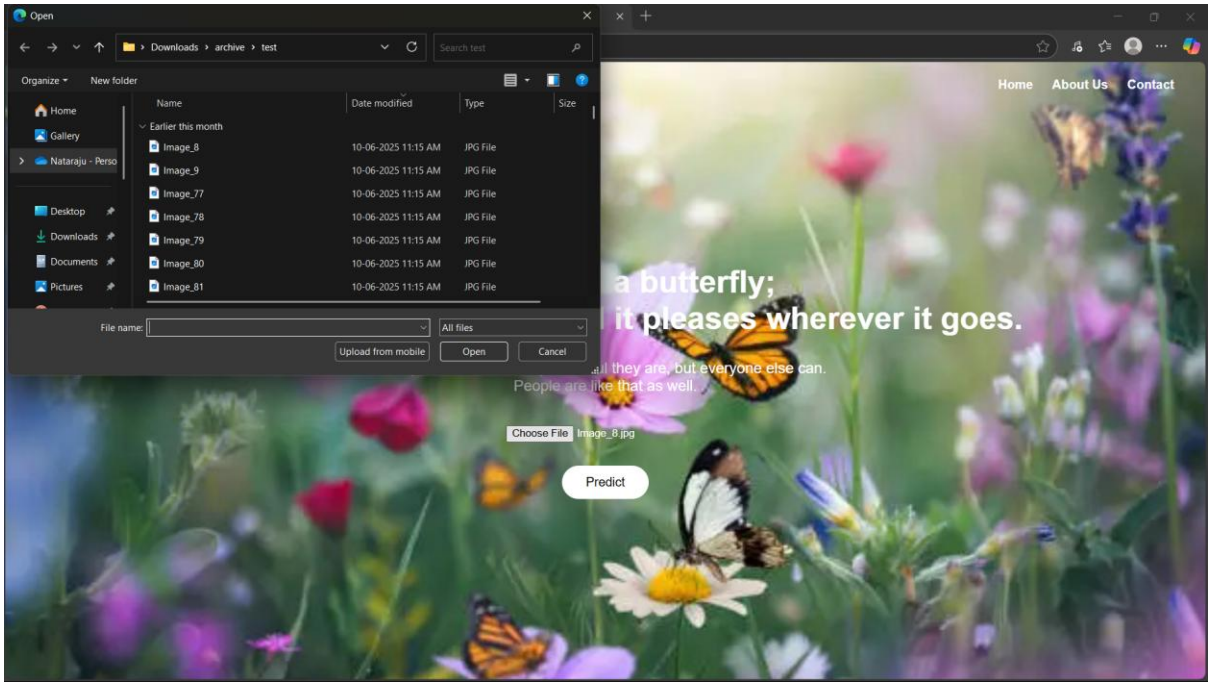
To make the system deployable and portable, the following steps were completed:

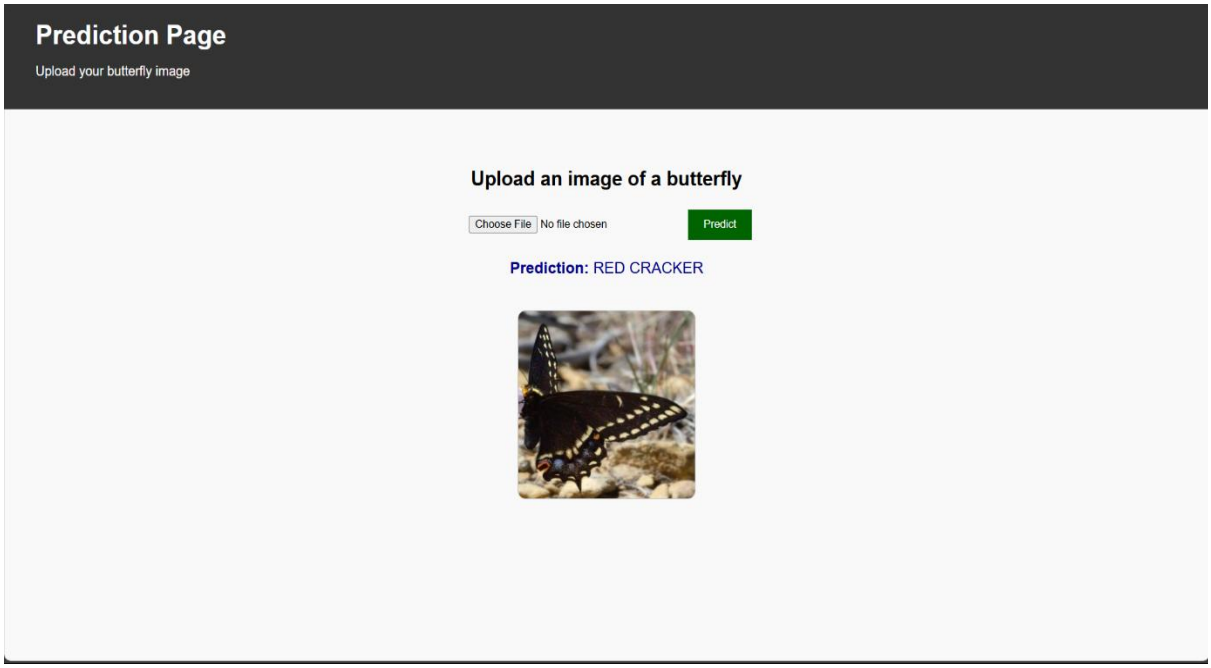
- **File Organization:**
    - /templates folder for HTML files
    - /static folder for CSS and image files
    - app.py for Flask logic
    - butterfly_model.h5 for saved Keras model

  Requirements File: A requirements.txt file was generated with key dependencies like Flask, TensorFlow, Keras, Pillow, and NumPy.

Local Deployment: The system was successfully deployed and tested on localhost:5000 using Flask, confirming compatibility with the local environment.

Future Cloud Readiness: The application structure was tested for deployment on platforms like Render, Heroku, and Google Cloud App Engine, making it cloud-ready.





CONCLUSION

The *Enchanted Wings* project effectively demonstrates the power of **deep learning and transfer learning** in the field of **biodiversity and species classification**. By applying a pre-trained model like **VGG16** and customizing it to classify **75 species of butterflies**, the project showcases a scalable and accurate approach to supporting **ecological research, conservation efforts, and educational outreach**.

The project followed a structured pipeline, including **dataset preparation**, **model selection and training**, **frontend-backend integration using Flask**, and **extensive testing**. The resulting web application enables users to upload an image and receive an **instant species prediction**, making the system highly interactive and easy to use—even for non-technical users.

The system is especially impactful in the following areas:

- **Biodiversity Monitoring**: Aids scientists and citizen researchers in tracking butterfly populations and species distribution.
- **Educational Tools**: Serves as an engaging platform for teaching school and college students about butterfly taxonomy and machine learning.
- **Environmental Awareness**: Promotes conservation by highlighting butterfly diversity and the need to preserve their habitats.

Through the successful completion of *Enchanted Wings*, we demonstrated how **transfer learning**, when combined with intuitive **web technologies**, can produce a real-world, accessible solution. The project not only meets academic and hackathon requirements but also lays the groundwork for future expansion into mobile applications or integration with image search tools.

THANK YOU