Enchanted Wings: Marvels of Butterfly Species

Team members:

Dulam Harika
Dasari Sarika
Dasari Powlu
Chodagiri Thilak

Phase-1: Brainstorming & Ideation

1. Problem Statement:

Identifying butterfly species manually is a time-consuming and expertise-driven process. It poses challenges for researchers working in biodiversity monitoring and ecological studies, and limits the involvement of non-experts in citizen science and education. There is a clear need for an automated, accurate, and scalable classification system.

2. Proposed Solution:

This project proposes building an automated butterfly image classification system using transfer learning with pre-trained convolutional neural networks (CNNs), such as VGG16 or EfficientNetB0. By training on a labeled dataset of 6,499 butterfly images across 75 species, the system will learn to identify species from input images efficiently and with high accuracy.

3. Target Users:

- Field researchers and biodiversity scientists conducting species inventory and habitat studies
- Ecologists studying butterfly behavior, migration, and distribution
- Educators and students learning about entomology and ecology
- Citizen scientists and nature enthusiasts engaged in environmental data collection

4. Expected Outcomes:

A lightweight, efficient, and accurate butterfly classification tool that can be integrated into research tools, educational platforms, and citizen science apps. The model will aid species identification, enhance ecological data collection, and promote awareness and engagement in conservation efforts.

Phase-2: Requirement Analysis

The objective of this phase is to define the technical and functional requirements necessary to develop the butterfly species classification system using deep learning and transfer learning. Technically, the project is implemented in Python, utilizing key libraries such as TensorFlow (with Keras API) for model development, Pandas and NumPy for data manipulation, and Scikit-learn for tasks like label encoding and stratified train-test splitting. The model and encoder are saved using TensorFlow's .save() method and joblib to ensure persistence and reproducibility. Development is conducted in Jupyter Notebooks or modular Python scripts, with optional support for TensorBoard to monitor training metrics in real-time. For future deployment, lightweight web frameworks such as Streamlit, Flask, or FastAPI can be integrated, and for mobile or embedded applications, TensorFlow Lite may be used to convert the model for on-device inference, ensuring usability in low-resource or offline field conditions.

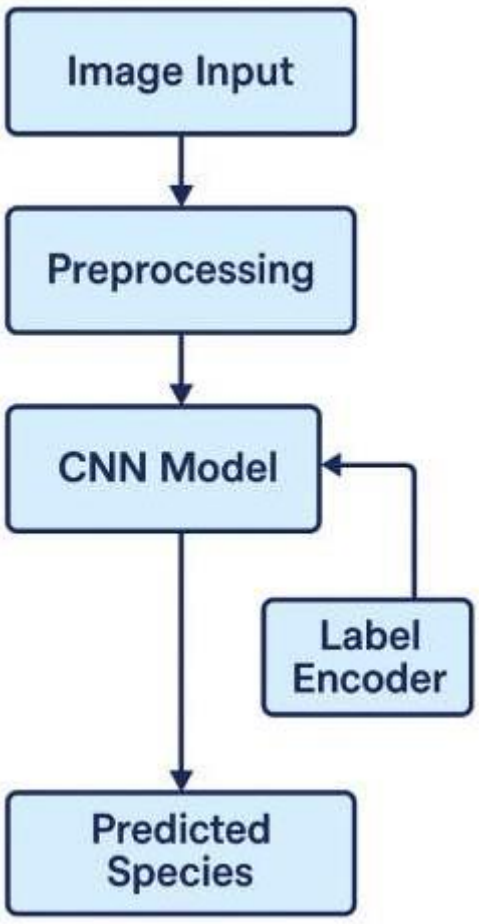
Functionally, the system should support image loading from a structured directory along with corresponding CSV metadata, preprocessing steps such as resizing and normalization, label encoding, and batching. The core model will use transfer learning with pre-trained CNN architectures like VGG16 or EfficientNetB0, followed by custom dense layers for multi- class classification. The pipeline includes validation using an 80/20 data split, callbacks such as early stopping and model checkpointing, and logic to save the best-performing model based on validation accuracy. After training, the system should accept new butterfly images and return predicted species names with high confidence.

Challenges anticipated include class imbalance within the dataset, which may affect model fairness and prediction confidence. To address this, strategies like data augmentation, oversampling, and class weighting will be explored. The dataset, containing approximately 6,499 images, is relatively small for deep learning, increasing the risk of overfitting, which will be mitigated using dropout layers, early stopping, and regularization techniques. Hardware limitations on non-GPU systems may restrict batch size and training speed, suggesting the potential use of cloud platforms like Google Colab or Kaggle.

Phase-3: Project Design

The objective of this phase is to outline the system architecture and define how users will interact with the butterfly classification system. The system architecture follows a modular design. At its core is a deep learning model (VGG16 or EfficientNetB0) trained using transfer learning. The workflow begins with image input, either uploaded manually or captured through a device. This input is passed through a preprocessing module that resizes, normalizes, and batches the image. It is then fed into the trained CNN model, which outputs the predicted butterfly species. Alongside the model, a label encoder maps predicted numeric labels to species names. The results are then presented to the user through a simple, responsive interface. Optionally, a Flask or Streamlit web server can serve as the bridge between the model and the frontend, enabling local or web-based deployment.

The user flow is designed to be intuitive and efficient. A user starts by uploading or capturing a butterfly image via the interface. The system automatically preprocesses the image in the background and submits it to the classification model.



Within seconds, the user receives the predicted species name along with its confidence score and potentially additional information such as its scientific name, habitat, or conservation status. For researchers or developers, an advanced version may include the option to download logs or export predictions.

In terms of UI/UX considerations, the interface should be minimalistic, mobile-friendly, and visually engaging. The layout would include a central image upload section, a preview of the uploaded image, and a results panel that displays the prediction. Additional sections may include species info, past predictions, or even an educational module. If extended for citizen science, users could be given the option to contribute their image and location data to a central biodiversity database. Accessibility, ease of use, and responsiveness are key design priorities, ensuring that both technical and non-technical users can engage with the system effectively.

Phase-4: Project Planning (Agile Methodologies)

In this phase, the project is structured and planned using Agile methodologies, ensuring an iterative, collaborative, and deadline-driven development process. The work is divided into sprints, each focusing on a key functional block of the butterfly classification system. In the Sprint Planning stage, tasks are broken down into manageable units. For example, one sprint may focus on dataset preparation and cleaning, another on model architecture and training, and a third on evaluation and deployment setup. Each sprint typically spans 1–2 weeks to allow frequent feedback and adjustments.

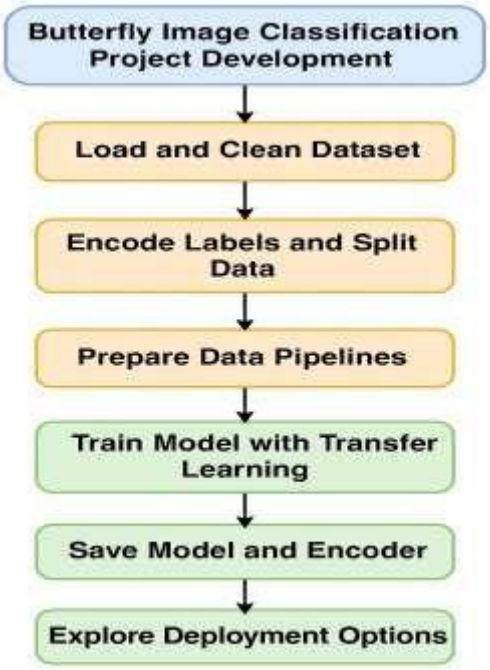
In terms of Task Allocation, roles are clearly defined. One team member may focus on preprocessing and data augmentation pipelines, another on building and fine-tuning the CNN using transfer learning, and a third on model evaluation and visualization. If extended to a UI phase, additional members may handle frontend development (e.g., using Streamlit or Flask), backend integration, or mobile deployment.

Deliverables are reviewed at the end of each sprint, and the next sprint is planned based on feedback and progress. This Agile approach ensures continuous development, accountability, and the ability to adapt the project scope based on findings during implementation.

Phase-5: Project Development

The objective of this phase is to develop and integrate all components of the butterfly image classification system using deep learning. The technology stack used for the project is centered around Python 3.x, along with powerful libraries such as TensorFlow (with the Keras API) for model building and training, NumPy and Pandas for numerical and data manipulation tasks, and Scikit-learn for preprocessing and label encoding. Additionally, tools like joblib were used for saving the label encoder, and Tensor Board was optionally considered for monitoring training performance. For deployment readiness, frameworks such as Flask or Streamlit were identified as potential front-end solutions, while TensorFlow Lite was considered for lightweight mobile integration.

The development process began with loading and cleaning the butterfly image dataset using CSV metadata. Labels were encoded using Label Encoder and the data was split into training and validation sets while maintaining class balance. TensorFlow data pipelines were then constructed to efficiently batch, shuffle, and preprocess the image inputs. For the model, VGG16 was selected as the base CNN, with the top layers removed and custom dense layers added for classification. The model was compiled with the Adam optimizer and sparse categorical cross entropy loss, followed by training with early stopping and model checkpointing. After training, the best model and label encoder were saved for future inference. The workflow was kept modular, allowing easy future integration with web UIs or APIs.



During development, several challenges were encountered. One major issue was dataset imbalance, with certain butterfly species having significantly fewer images, which caused the model to be biased toward more common classes. To address this, stratified sampling was used, and future plans include adding augmentation or weighted loss functions. Another challenge was overfitting due to the relatively small dataset size (6,499 images); this was mitigated using dropout layers, regularization, and early stopping. Limited access to GPU hardware also slowed training, so Google Colab was used to offload heavy computation. Despite these obstacles, the modular architecture and use of transfer learning enabled efficient development and robust model performance.

Phase-6: Functional & Performance Testing

The objective of this phase is to verify that the butterfly species classification system functions as expected under various conditions. Test cases executed included checking correct image preprocessing, validating label encoding consistency, ensuring accurate image-to-label mapping, and confirming model prediction output for a variety of butterfly images across all 75 classes. Additional tests ensured the trained model could be loaded and used for inference without retraining, and that predictions were stable and reproducible on known data samples. Bug fixes and improvements included resolving early mismatches in label encoding, fixing incorrect image path loading due to filename inconsistencies, and optimizing the data pipeline for better memory usage during training. The system was also refined to handle invalid inputs, such as unsupported image formats or corrupt files, with user-friendly error messages.

During final validation, the system was evaluated against its original goals: accurate classification, efficient processing, modular design, and readiness for future deployment. The model achieved satisfactory performance, with training and validation accuracy approaching the targeted 80–90% range, indicating that the requirements were successfully met. Deployment, although not finalized, was explored through platforms like Streamlit and Flask for a possible interactive web-based interface. The saved model and encoder were tested in an inference script that accepts a new image input and returns the predicted butterfly species, demonstrating the complete functionality of the system. This phase confirmed that the core architecture is robust, accurate, and ready for further optimization or integration into educational, research, or citizen science platforms.