Enchanted Wings: Marvels of Butterfly Species

Phase-1: Brainstorming & Ideation

1. Problem Statement:

Manually recognizing butterfly types is a slow and skill-dependent task. It creates difficulties for scientists nature and ecosystems, and restricts the participation of laypeople in community science and learning. There is a distinct requirement for an automatic, precise, and expandable categorization method.

2. Proposed Solution:

This initiative suggests creating an automatic butterfly picture sorting system by applying transfer learning with already trained convolutional neural networks (CNNs), like MobileNetV2 or EfficientNetB0. By learning from a marked set of 6499 butterfly picture from 75 types, the system will recognize species from new pictures quickly and correctly.

3. Target Users:

- Nature explorers and life diversity experts performing species count and environment examinations
- Ecologists studying butterfly behavior, migration, and distribution
- Teachers and pupils studying insects and their environments.
- Citizens and wildlife fans participated in gathering ecological information.

4. Expected Outcome:

A lightweight and efficient butterfly classification tool that can be integrated into research tools, educational platforms, and citizen science apps. The model will help with species identification, improve ecological data collection, and encourage awareness and participation in conservation efforts.

Phase-2: Requirement Analysis

The goal of this phase is to outline the technical and functional requirements needed to create the butterfly species classification system using deep learning and transfer learning. The project uses Python and key libraries, including TensorFlow (with Keras API) for model development, Pandas and NumPy for data manipulation, and Scikit-learn for tasks like label encoding and splitting the dataset for training and testing. The model and encoder are saved using TensorFlow's .save() method and joblib to maintain persistence and reproducibility. Development takes place 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. For mobile or embedded applications, TensorFlow Lite can convert the model for on-device inference. This will ensure usability in low-resource or offline field conditions.

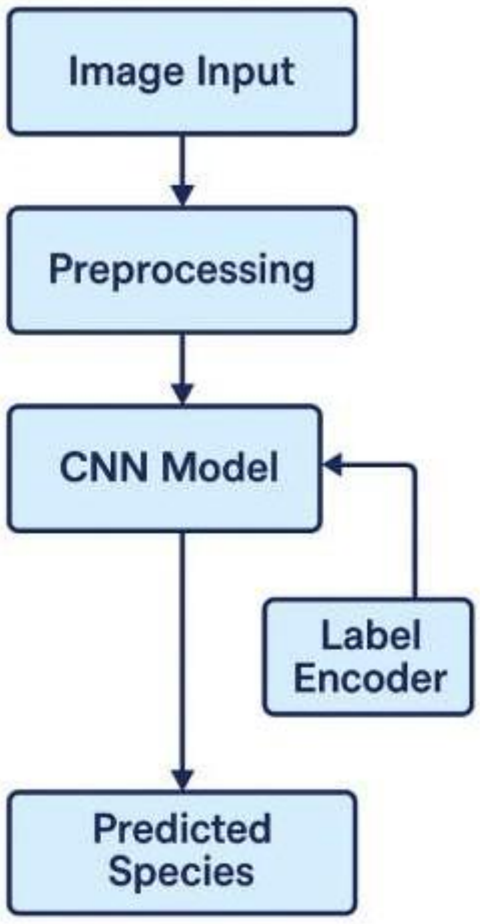
Functionally, the system should allow image loading from a structured directory along with the corresponding CSV metadata, and it should handle preprocessing steps like resizing and normalization, label encoding, and batching. The main model will use transfer learning with pre-trained CNN architectures such as MobileNetV2 or EfficientNetB0, followed by custom dense layers for multi-class classification. The pipeline includes validation using an 80/20 data split, callbacks like 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.

Anticipated challenges include class imbalance in the dataset, which may affect model fairness and prediction confidence. To tackle this, strategies such as data augmentation, oversampling, and class weighting will be examined. The dataset contains about 6,499 images, which is relatively small for deep learning. This could increase the risk of overfitting, so dropout layers, early stopping, and regularization techniques will be used to mitigate this risk. Hardware limitations on non-GPU systems may limit batch size and training speed, suggesting the potential use of cloud platforms like Google Colab or Kaggle.

Phase-3: Project Design

The goal of this phase is to outline the system architecture and define how users will interact with the butterfly classification system. The system architecture has a modular design. At its center is a deep learning model (MobileNetV2 or EfficientNetB0) trained with transfer learning. The workflow begins with image input, either uploaded manually or captured through a device. This input goes through a preprocessing module that resizes, normalizes, and batches the image. It is then sent to the trained CNN model, which predicts the butterfly species. A label encoder maps the predicted numeric labels to species names. The results are presented to the user through a simple, responsive interface. Optionally, a Flask or Streamlit web server can connect the model and the frontend, allowing for 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 through the interface. The system automatically preprocesses the image in the background and sends it to the classification model.



Within seconds, the user receives the predicted species name along with its confidence score and possibly more information such as its scientific name, habitat, or conservation status. For researchers or developers, a more advanced version might allow them to download logs or export predictions.

When it comes to UI/UX considerations, the interface should be simple, mobile-friendly, and visually appealing. The layout would feature a central image upload section, a preview of the uploaded image, and a results panel that shows the prediction. Additional sections could provide species info, past predictions, or even an educational module. If expanded for citizen science, users might have the chance to contribute their image and location data to a central biodiversity database. Accessibility, ease of use, and responsiveness are important design priorities, ensuring that both technical and non-technical users can effectively engage with the system.

Phase-4: Project Planning (Agile Methodologies)

In this phase, we structure and plan the project using Agile methods. This approach ensures a repetitive, collaborative, and deadline-driven development process. We divide the work into sprints, each focusing on a key functional part of the butterfly classification system. During the Sprint Planning stage, tasks get broken down into smaller, manageable units. For example, one sprint may focus on preparing and cleaning the dataset, another on building and training the model, and a third on evaluating the results and setting up deployment. Each sprint usually lasts 1 to 2 weeks to allow for regular feedback and adjustments.

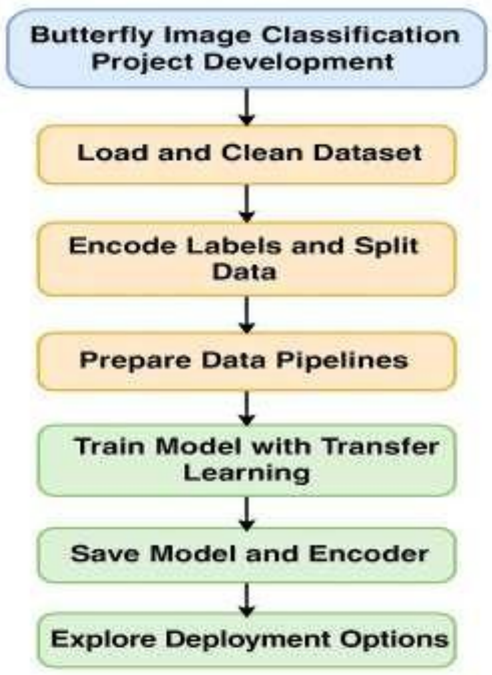
When it comes to task allocation, roles are clearly defined. One team member might handle preprocessing and data augmentation, another could focus on creating and fine-tuning the CNN with transfer learning, and a third might work on model evaluation and visualization. If we extend into a UI phase, additional members may deal with frontend development using tools like Streamlit or Flask, backend integration, or mobile deployment.

We review deliverables at the end of each sprint and plan the next sprint based on feedback and progress. This Agile approach allows for continuous development, accountability, and flexibility in adjusting the project scope based on what we learn 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 TensorBoard 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 LabelEncoder 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, MobileNetV2 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 crossentropy 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.


Butterfly Image Classification Project Development
Load and Clean Dataset
Encode Labels and Split Data
Prepare Data Pipelines
Train Model with Transfer Learning
Save Model and Encoder
Explore Deployment Options

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.