PROJECT REPORT

1. INTRODUCTION

1.1 Project Overview

Enchanted Wings: Marvels of Butterfly Species is an AI/ML-based image classification system designed to identify various butterfly species using deep learning techniques. It aims to support biodiversity efforts by providing researchers, biologists, and nature enthusiasts with a simple, fast, and offline-capable solution to classify butterfly species from images.

1.2 Purpose

The purpose of this project is to automate the identification of butterfly species using Convolutional Neural Networks (CNNs). This contributes to biodiversity documentation, educational exploration, and environmental conservation through a user-friendly, AI-powered platform.

2. IDEATION PHASE

2.1 Problem Statement

Manual classification of butterfly species is challenging due to:

- Visual similarities among species.
- Inconsistent image quality.
- Lack of taxonomic expertise in the field.

2.2 Empathy Map Canvas

Think: Will this model handle blurry or rare species?
Feel: Frustrated by misidentification; excited when accurate.
Say: "I want a quick, reliable, and easy tool."
Do: Capture images, upload them, review predictions.
Pains: Manual classification, low-quality data, complex tools.
Gains: Fast, offline access to AI-based species recognition.

2.3 Brainstorming

Ideas considered:

- Mobile app vs web app
- Offline vs cloud model
- Real-time vs batch image upload
- CNN vs other model types

- Geolocation tagging (for future versions)

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Stage 1: Discover – User finds app and reads butterfly fact
Stage 2: Upload – User uploads butterfly image
Stage 3: Predict – Model predicts species and confidence
Stage 4: Learn – User sees info, image, and history
Stage 5: Reflect – User downloads or stores results

3.2 Solution Requirements

- Functional: Image upload, prediction, display of results
- Non-functional: Usability, offline access, scalability, performance

3.3 Data Flow Diagram

Level 0 and Level 1 DFD already covered above (image upload → preprocessor → model → result → history log)

3.4 Technology Stack

- **Frontend**: HTML, Streamlit
- **Backend**: Python (Flask), TensorFlow/Keras
- **Database**: MongoDB or SQLite
- **Model**: CNN (MobileNetV2, ResNet50)
- **Deployment**: Localhost (optional Docker/Cloud)

4. PROJECT DESIGN

4.1 Problem-Solution Fit

A user-facing web application solves the need for fast, reliable butterfly classification, without requiring technical knowledge or internet access.

4.2 Proposed Solution

User uploads an image → Model classifies species → Output displayed with confidence → Optionally logged in database

4.3 Solution Architecture

As described in the previous architecture section (UI → Backend → CNN Model → Database/File System)

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Sprint	Task	Story Points
1	Data collection & cleaning	8
2	Model training, testing, deployment	16

Velocity: 12 story points/sprint

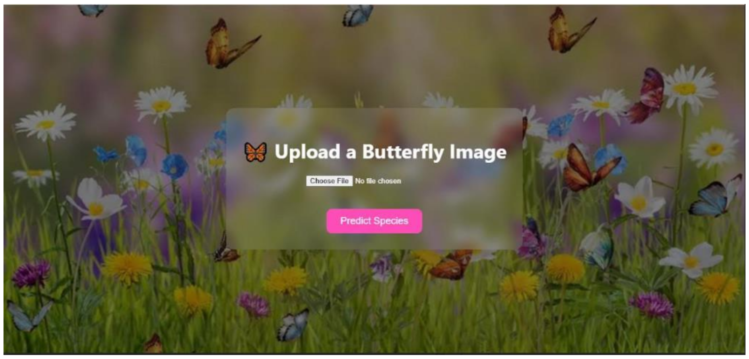
6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

- Trained on GPU via Google Colab
- Achieved training accuracy ~95%, validation ~90%
- Prediction latency: ~2 seconds per image
- Confusion matrix used to assess misclassifications

7. RESULTS

7.1 Output Screenshots



8. ADVANTAGES & DISADVANTAGES

Advantages

- Fast and accurate predictions
- Easy for non-technical users
- Works offline
- Encourages biodiversity exploration

Disadvantages

- Misclassification for visually similar species
- Limited dataset may restrict accuracy on rare butterflies
- No current mobile version

9. CONCLUSION

The *Enchanted Wings* project demonstrates how AI and deep learning can be applied to biodiversity conservation and education. It automates species identification and provides a valuable tool for researchers and enthusiasts alike.

10. FUTURE SCOPE

- Expand dataset to include more regional species
- Add geolocation-based predictions
- Cloud deployment for remote access
- User login and profile history
- Mobile application version

11. APPENDIX

- **Source Code**: app.py

```python
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import os
import random
```

Code view is read-only. Switch to the editor.

```python
# Auto-sorted class names from training folders
CLASS_NAMES = sorted(os.listdir("train"))

# Fun facts with emojis 🦋
facts = [
    "🦋 Butterflies can see ultraviolet light, invisible to the human eye.",
    "⚡ Some butterflies can fly up to 30 miles per hour!",
    "👣 Butterflies taste with their feet!",
    "🌍 There are around 20,000 species of butterflies worldwide.",
    "🌞 Butterflies can't fly if they're cold—they need sunshine to warm up!",
    "🌸 A butterfly's life is mostly spent as a caterpillar before it transforms.",
    "🎨 No two butterflies have the same wing pattern!",
    "🧭 Monarch butterflies migrate thousands of miles each year.",
    "🌺 Butterflies help pollinate flowers just like bees!",
    "🦋 The wings of butterflies are covered in tiny scales that reflect light beautifully."
]

@app.route("/")
def welcome():
    return render_template("welcome.html", fact=random.choice(facts))

@app.route("/input")
def input_page():
    return render_template("input.html")

@app.route("/output", methods=["POST"])
def output():
    if 'file' not in request.files:
        return render_template("output.html", prediction=None, image_path=None)

    file = request.files['file']
    if file.filename == '':
        return render_template("output.html", prediction=None, image_path=None)
```

```python
        # Save uploaded file in static folder
        filename = "uploaded.jpg"
        path = os.path.join("static", filename)
        file.save(path)

        # Preprocess and predict
        img = image.load_img(path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0) / 255.0

        predictions = model.predict(img_array)
        predicted_class = CLASS_NAMES[np.argmax(predictions)]

        #return render_template('output.html', prediction=prediction, image_path=image_filename)

        return render_template("output.html", prediction=predicted_class, image_path=path)

    if __name__ == "__main__":
        app.run(debug=True)
```

Predict_image.py:

```
import os
Code view is read-only. Switch to the editor.

from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

model_path = 'vgg16_model.h5'
test_dir = 'test'
class_names = sorted(os.listdir('train'))

model = load_model(model_path)

def predict_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    predictions = model.predict(img_array)
    predicted_class = class_names[np.argmax(predictions)]

    plt.imshow(img)
    plt.title(f'Predicted: {predicted_class}')
    plt.axis('off')
    plt.show()

test_path = os.path.join(os.getcwd(), test_dir)
test_images = [f for f in os.listdir(test_path) if f.endswith(('.jpg', '.jpeg', '.png'))]

if not test_images:
    print("⚠️ No test images found in the 'test' folder.")
else:
    for img_file in test_images:
        print(f'\n🖼️ Predicting: {img_file}')
        predict_image(os.path.join(test_path, img_file))
```

Train_model.py:

```
# train_model.py
Code view is read-only. Switch to the editor.

import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Constants
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 5

# Dataset path
train_path = "D:/Butterfly_Project/butterfly_dataset/train"

# Data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='sparse',
    subset='training'
)
```

```python
val_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='sparse',
    subset='validation'
)

# Load VGG16 base model
vgg = VGG16(include_top=False, weights='imagenet', input_shape=(IMG_SIZE, IMG_SIZE, 3))
vgg.trainable = False  # Freeze layers

# Build model
model = Sequential([
    vgg,
    Flatten(),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint("vgg16_model.h5", save_best_only=True)
earlystop = EarlyStopping(patience=5, restore_best_weights=True)

# Train model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
    callbacks=[checkpoint, earlystop]
)
```

```python
# Plot accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.show()

# Plot loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.legend()
plt.show()
```

- **GitHub & Demo**: https://github.com/Ruhinaaz28/Enchanted-Wings-Marvels-of-Butterfly-Species