Project Title: Enchanted Wings: Marvels of Butterfly Species - A CNN-Based Image Classification Model

Background:

Butterflies, as important pollinators and indicators of biodiversity, exhibit a remarkable diversity in their wing patterns, colors, and morphology. Manually identifying butterfly species is a labor-intensive process that requires expert taxonomic knowledge. With advancements in artificial intelligence and deep learning, computer vision systems can assist in automating this classification task. This project leverages Convolutional Neural Networks (CNNs) to identify butterfly species from images, contributing to efficient biodiversity monitoring and aiding ecological research and education.

Abstract:

This project focuses on classifying butterfly species using a Convolutional Neural Network (CNN) trained on a Kaggle dataset. With the increasing role of machine learning in biodiversity, this project aims to automate the recognition of butterfly species based on image data. The deep learning model enhances accuracy by using data augmentation and normalization techniques. The solution has real-world applications in ecological research, conservation, and education.

Objective:

To develop a CNN model that classifies images of butterflies into their respective species using supervised learning and image processing techniques.

Algorithm Explanation:

The CNN algorithm used involves several convolutional layers to extract spatial features, pooling layers to reduce dimensionality, batch normalization for faster convergence, and dense layers to perform final classification. The model uses the softmax function for multi-class classification and is trained using the Adam optimizer and categorical cross-entropy loss.

Project Explanation:

1. The butterfly image dataset is downloaded from Kaggle using kagglehub.
2. Images are loaded and labeled using a CSV file.
3. The dataset is split into training and validation sets.
4. A CNN is built using TensorFlow/Keras with multiple layers.
5. The model is trained over 30 epochs.
6. Performance is evaluated using accuracy and loss curves.

7. Predictions are visualized to compare actual and predicted labels.

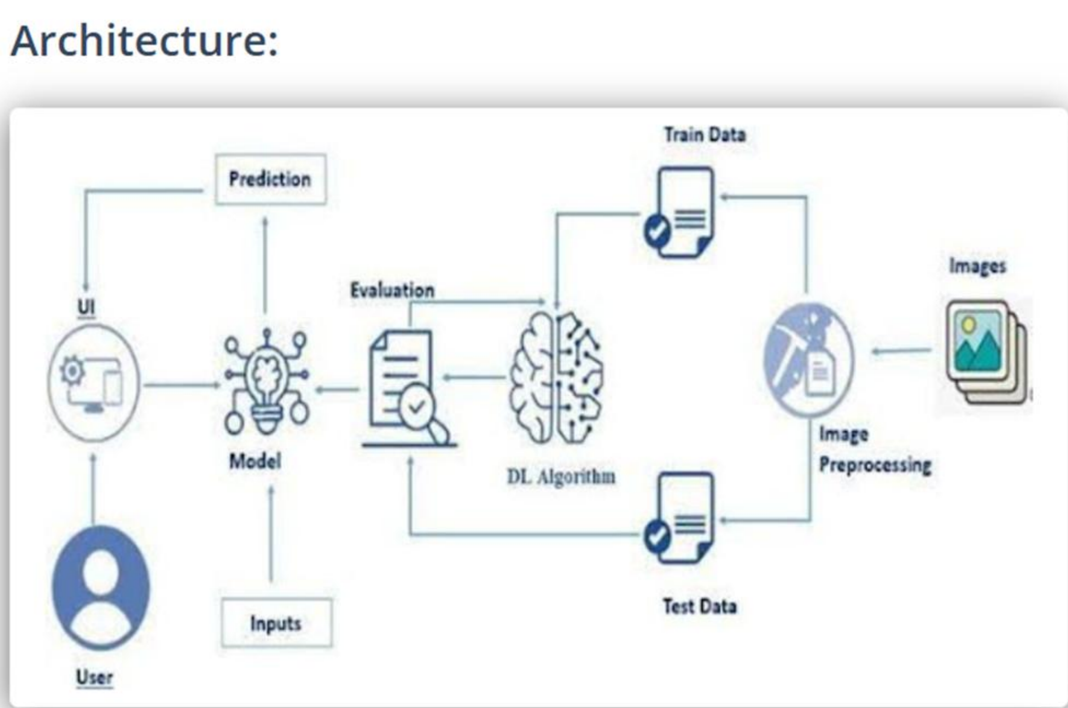
Project Flow:

1. Dataset Collection
2. Data Preprocessing
3. Image Augmentation
4. CNN Model Building
5. Training the Model
6. Model Evaluation
7. Predictions

Prior Knowledge Required:

- Basics of Python programming
- Understanding of neural networks and CNNs
- Familiarity with TensorFlow/Keras libraries
- Knowledge of image data processing
- Basic data handling with pandas and NumPy

Architecture:



The architecture of the Convolutional Neural Network (CNN) used in this project is specifically designed to handle image data and learn hierarchical patterns from raw pixels to high-level features such as species characteristics. Below is a visual representation and detailed explanation:

Input Image (128x128x3)
↓
[Conv2D + ReLU] → [BatchNormalization] → [MaxPooling]
↓
[Conv2D + ReLU] → [BatchNormalization] → [MaxPooling]
↓
[Conv2D + ReLU] → [BatchNormalization] → [MaxPooling]
↓
[Flatten] → [Dense 128] → [Dense 512] → [Dense n_classes + Softmax]

1. **Input Layer (128x128x3):**
    - Accepts RGB butterfly images resided to 128x128 pixels.
    - The '3' denotes the three color channels: Red, Green, and Blue.
2. **Convolutional Layer (Conv2D)**:
    - Applies multiple filters to extract low-level features like edges, curves, and textures.
    - Each filter learns to activate on different patterns during training.
3. **Activation Function (ReLU)**:
    - Introduces non-linearity by converting all negative values to zero, helping the network learn complex functions.
4. **BatchNormalization Layer**:
    - Normalizes the output of the convolution layer to speed up training and improve stability.
    - Helps in reducing the internal covariant shift.
5. **MaxPooling Layer**:
    - Down samples the feature map size by taking the maximum value from a group of pixels.
    - Reduces computational load and provides translation invariant.
6. **Flatten Layer**:
    - Converts the 2D feature maps into a 1D vector so it can be fed into dense layers.
7. **Dense (Fully Connected) Layers**:
    - First dense layer (128 neurons): Learns more abstract combinations of the earlier features.
    - Second dense layer (512 neurons): Further refines high-level representations.
8. **Output Layer (Dense with Softmax)**:
    - Number of neurons equals the number of butterfly species (classes).
    - Softmax activation converts raw outputs into probabilities that sum to 1.

Project Structure:

```
├── butterfly_classification_project
│   ├── dataset
│   │   ├── train
│   │   ├── test
│   │   └── Training_set.csv
│   ├── main.py (or .ipynb)
│   ├── model.h5
│   ├── requirements.txt
│   └── README.md
```

1.dataset/

This folder contains all the image data needed for the model.

- **train/** – This folder includes images used to train the model. Each image is labeled with a filename that corresponds to a butterfly species.
- **test/** – Used for model evaluation or prediction after training.
- **Training_set.csv** – A CSV file that contains two main columns:
  - filename: Name of the image file

label: Butterfly species for that image

2. main.py or notebook.ipynb

This is the **core file** of the project.

- Contains all the Python code for:
  - Loading the dataset
  - Preprocessing images
  - Defining the CNN model
  - Training and validating the model
  - Saving the model
  - Making predictions
- If you're using **Google Colab or Jupyter**, this would be a .ipynb (notebook) file.

3. model.h5

- This is the **trained model file**.
- After training, the model is saved in .h5 format using:

python
CopyEdit
model.save("model.h5")

- You can later load it with:

python
CopyEdit
model = tf.keras.models.load_model("model.h5")

- This avoids retraining every time.

4. requirements.txt

- A list of all the **Python libraries** needed to run the project.
- Example:

  nginx
  CopyEdit
  tensorflow

pandas
numpy
matplotlib

- You can install them all at once using:
  nginx
  CopyEdit

pip install -r requirements.txt

5. README.md

- This is a markdown file that explains the **project overview**:
  - What the project does
  - Dataset used
  - Steps to run the code
  - Expected outputs

It's like a **mini guide** for users or developers.

Dataset Collection:

- Source: Kaggle Dataset - phucthaiv02/butterfly-image-classification
- Format: Images (.jpg/.png) with a CSV file containing filename-label pairs.
- Categories: Multiple butterfly species

	A	B
1	filename	label
2	Image_1.jpg	SOUTHERN DOGFACE
3	Image_2.jpg	ADONIS
4	Image_3.jpg	BROWN SIPROETA
5	Image_4.jpg	MONARCH
6	Image_5.jpg	GREEN CELLED CATTLEHEART
7	Image_6.jpg	CAIRNS BIRDWING
8	Image_7.jpg	GREEN CELLED CATTLEHEART
9	Image_8.jpg	EASTERN DAPPLE WHITE
10	Image_9.jpg	BROWN SIPROETA
11	Image_10.jpg	RED POSTMAN
12	Image_11.jpg	MANGROVE SKIPPER
13	Image_12.jpg	BLACK HAIRSTREAK
14	Image_13.jpg	CABBAGE WHITE
15	Image_14.jpg	RED ADMIRAL
16	Image_15.jpg	PAINTED LADY
17	Image_16.jpg	MANGROVE SKIPPER
18	Image_17.jpg	PAPER KITE
19	Image_18.jpg	SOOTYWING
20	Image_19.jpg	PINE WHITE
21	Image_20.jpg	PEACOCK
22	Image_21.jpg	CHECQUERED SKIPPER
23	Image_22.jpg	JULIA
24	Image_23.jpg	COMMON WOOD-NYMPH
25	Image_24.jpg	BLUE MORPHO
26	Image_25.jpg	CLOUDED SULPHUR
27	Image_26.jpg	STRAITED QUEEN
28	Image_27.jpg	ORANGE OAKLEAF
29	Image_28.jpg	PURPLISH COPPER

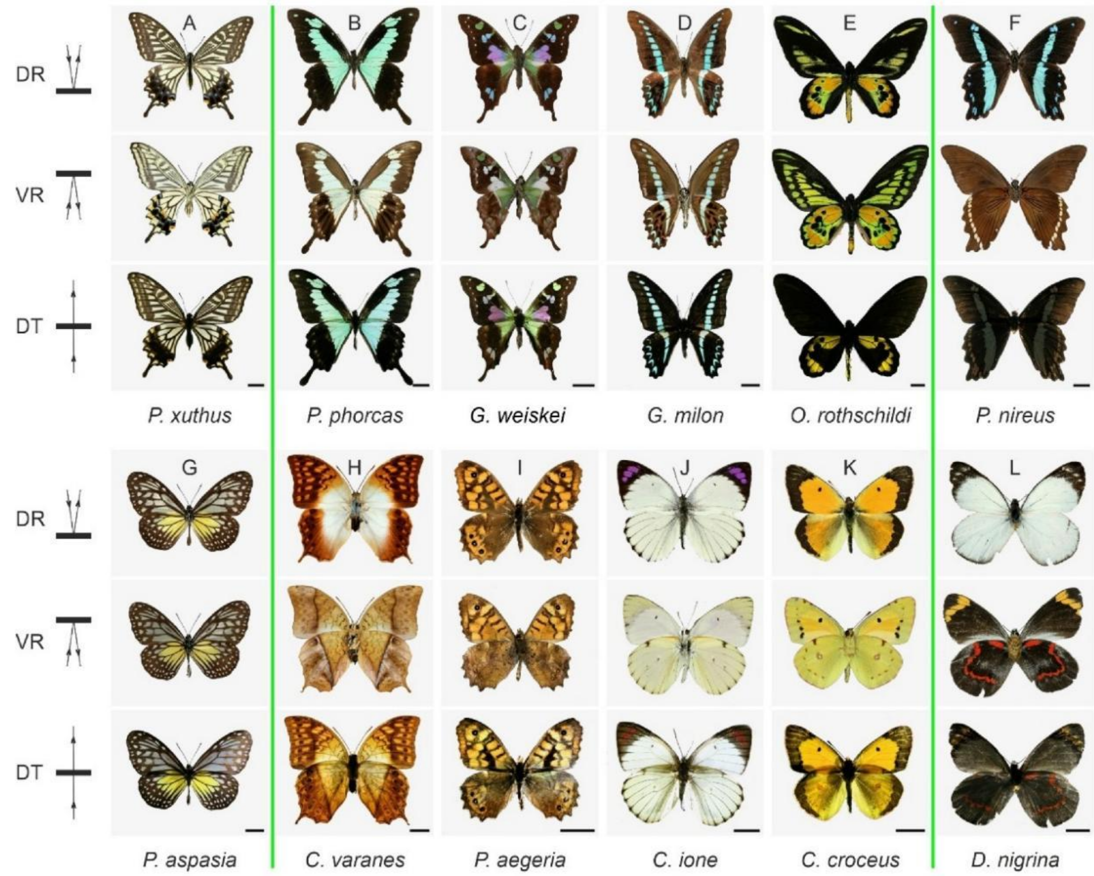
28	Image_27.jpg	ORANGE OAKLEAF
29	Image_28.jpg	PURPLISH COPPER
30	Image_29.jpg	CLOUDED SULPHUR
31	Image_30.jpg	ATALA
32	Image_31.jpg	IPHICLUS SISTER
33	Image_32.jpg	CAIRNS BIRDWING
34	Image_33.jpg	CAIRNS BIRDWING
35	Image_34.jpg	BLACK HAIRSTREAK
36	Image_35.jpg	DANAID EGGFLY
37	Image_36.jpg	PAINTED LADY
38	Image_37.jpg	LARGE MARBLE
39	Image_38.jpg	DANAID EGGFLY
40	Image_39.jpg	PIPEVINE SWALLOW
41	Image_40.jpg	BLUE SPOTTED CROW
42	Image_41.jpg	LARGE MARBLE
43	Image_42.jpg	EASTERN DAPPLE WHITE
44	Image_43.jpg	LARGE MARBLE
45	Image_44.jpg	PAINTED LADY
46	Image_45.jpg	RED CRACKER
47	Image_46.jpg	QUESTION MARK
48	Image_47.jpg	CRIMSON PATCH
49	Image_48.jpg	BANDED PEACOCK
50	Image_49.jpg	CHECQUERED SKIPPER
51	Image_50.jpg	DANAID EGGFLY
52	Image_51.jpg	RED POSTMAN
53	Image_52.jpg	COMMON WOOD-NYMPH
54	Image_53.jpg	SCARCE SWALLOW
55	Image_54.jpg	DANAID EGGFLY
56	Image_55.jpg	ORANGE OAKLEAF
57	Image_56.jpg	COPPER TAIL
58	Image_57.jpg	GREAT JAY
59	Image_58.jpg	ORANGE OAKLEAF
60	Image_59.jpg	SCARCE SWALLOW

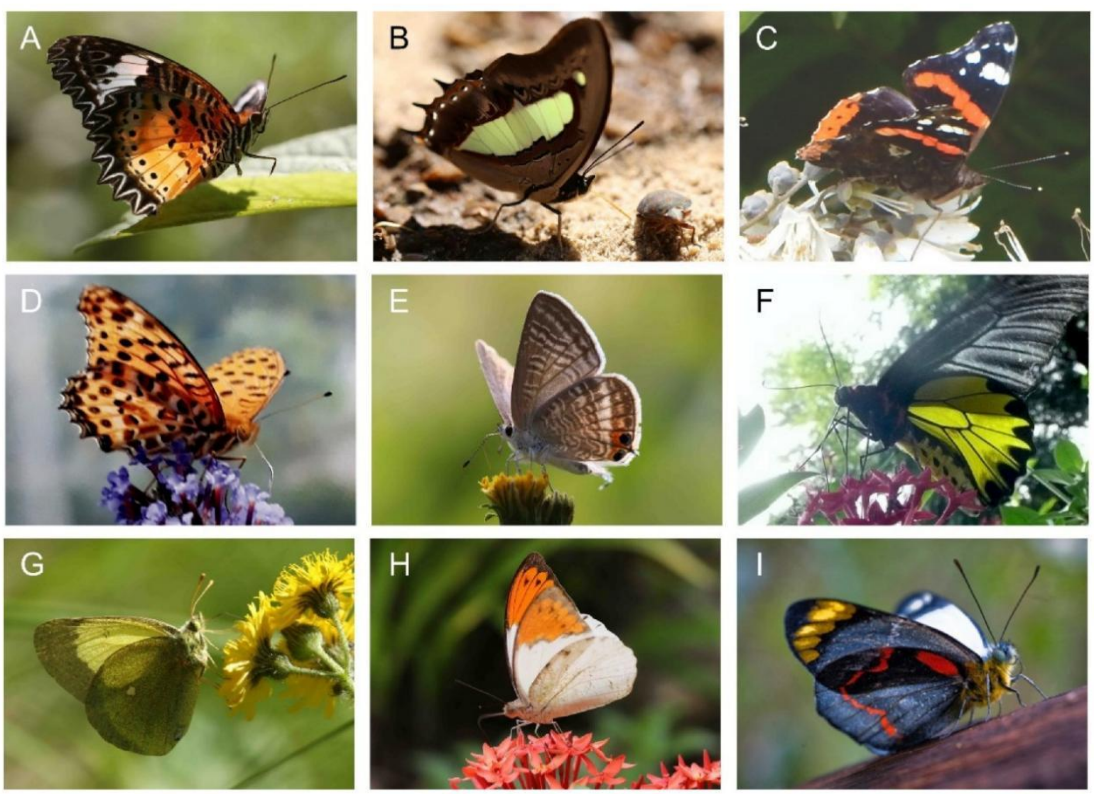
Training_set +

Data Visualization:

- Sample image grid from the dataset
- Bar chart showing the distribution of classes
- Accuracy vs Epochs line plot
- Loss vs Epochs line plot

Example Reference Images Used:





Model Building:

- Uses Keras Sequential API

- Three Conv2D + MaxPooling2D layers
- Dense layers for classification
- Optimizer: Adam
- Loss: categorical_crossentropy
- Metrics: Accuracy

Model Building with Code:

```
# Step 1: Importing required libraries
import pandas as pd
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormaliz
ation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

```
# Step 2: Load dataset

import kagglehub

phucthaiv02_butterfly_image_classification_path = kagglehub.dataset_download('phucthaiv02/butterfly-
image-classification')

print('Data source import complete.')
from google.colab import drive
import kagglehub
path = kagglehub.dataset_download("phucthaiv02/butterfly-image-classification")
```

output:

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/phucthaiv02/butterfly-image-classification?dataset_version_number=2...
100%|█████████| 226M/226M [00:02<00:00, 83.6MB/s]Extracting files...

Data source import complete.
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/phucthaiv02/butterfly-image-classification/versions/2
```

```
# Step 3: Load label file
labels_df = pd.read_csv('/content/Training_set.csv')

#printing the csv files

df
```

Output:

	filename	label
0	Image_1.jpg	SOUTHERN DOGFACE
1	Image_2.jpg	ADONIS
2	Image_3.jpg	BROWN SIPROETA
3	Image_4.jpg	MONARCH
4	Image_5.jpg	GREEN CELLED CATTLEHEART
...	...	...
6494	Image_6495.jpg	MANGROVE SKIPPER
6495	Image_6496.jpg	MOURNING CLOAK
6496	Image_6497.jpg	APPOLLO
6497	Image_6498.jpg	ELBOWED PIERROT
6498	Image_6499.jpg	ATALA

6499 rows × 2 columns

```
# Step 4: Image augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = train_datagen.flow_from_dataframe(
    dataframe=labels_df,
    directory=path + '/train',
    x_col='filename',
    y_col='label',
    subset='training',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

val_generator = train_datagen.flow_from_dataframe(
    dataframe=labels_df,
    directory=path + '/train',
    x_col='filename',
```

```
    y_col='label',
    subset='validation',
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)
```

Output:

```
Found 5200 validated image filenames belonging to 75 classes.
Found 1299 validated image filenames belonging to 75 classes.
```

```
# Step 5: Model building
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    BatchNormalization(),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(512, activation='relu'),
    Dense(train_generator.num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
batch_normalization (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dense_1 (Dense)	(None, 512)	66,048
dense_2 (Dense)	(None, 0)	0

Total params: 3,371,584 (12.86 MB)
Trainable params: 3,371,136 (12.86 MB)
Non-trainable params: 448 (1.75 KB)

```
# Step 6: Training
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=30
)
```

Output:

```
Epoch 1/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 291s 2s/step - accuracy: 0.0939 - loss: 4.1223 - val_accuracy: 0.0148 - val_loss: 8.3309
Epoch 2/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 103ms/step - accuracy: 0.1562 - loss: 3.8571 - val_accuracy: 0.0125 - val_loss: 8.6293
Epoch 3/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 278s 2s/step - accuracy: 0.2245 - loss: 3.0616 - val_accuracy: 0.0547 - val_loss: 5.5589
Epoch 4/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 103ms/step - accuracy: 0.3438 - loss: 2.2359 - val_accuracy: 0.0469 - val_loss: 5.7130
Epoch 5/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.3235 - loss: 2.5513 - val_accuracy: 0.1727 - val_loss: 3.3235
Epoch 6/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 104ms/step - accuracy: 0.3125 - loss: 2.5558 - val_accuracy: 0.1828 - val_loss: 3.2672
Epoch 7/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 266s 2s/step - accuracy: 0.3983 - loss: 2.1874 - val_accuracy: 0.3664 - val_loss: 2.5152
Epoch 8/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 20s 113ms/step - accuracy: 0.4062 - loss: 2.0131 - val_accuracy: 0.3367 - val_loss: 2.5858
Epoch 9/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 264s 2s/step - accuracy: 0.4683 - loss: 1.9313 - val_accuracy: 0.4133 - val_loss: 2.1932
Epoch 10/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 104ms/step - accuracy: 0.4062 - loss: 1.8996 - val_accuracy: 0.4039 - val_loss: 2.2223
Epoch 11/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.5125 - loss: 1.6680 - val_accuracy: 0.4516 - val_loss: 2.0771
Epoch 12/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 113ms/step - accuracy: 0.5000 - loss: 2.0038 - val_accuracy: 0.4375 - val_loss: 2.0457
Epoch 13/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 268s 2s/step - accuracy: 0.5520 - loss: 1.5618 - val_accuracy: 0.4156 - val_loss: 2.3531
Epoch 14/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 104ms/step - accuracy: 0.4688 - loss: 2.0211 - val_accuracy: 0.4203 - val_loss: 2.3159
```

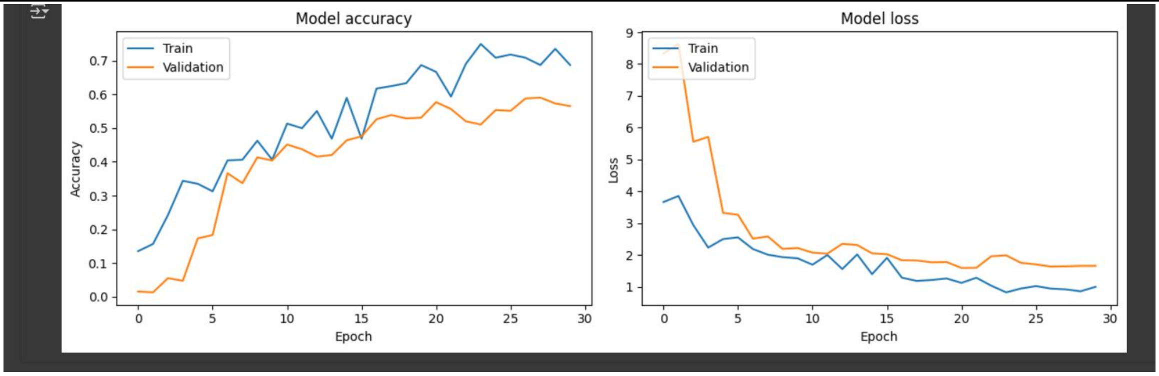
```
Epoch 15/30
 48/162 ━━━━━━━━━━━━━━━━━━━━ 2:54 2s/step - accuracy: 0.5745 - loss: 1.5192/usr/local/lib/python3.11/dist-packages/keras/src/legacy/preprocessing/image.py:1880: Depreca
  scipy.ndimage.interpolation.affine_transform(
162/162 ━━━━━━━━━━━━━━━━━━━━ 301s 2s/step - accuracy: 0.5797 - loss: 1.4566 - val_accuracy: 0.4641 - val_loss: 2.0561
Epoch 16/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 111ms/step - accuracy: 0.4688 - loss: 1.9141 - val_accuracy: 0.4758 - val_loss: 2.0277
Epoch 17/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 301s 2s/step - accuracy: 0.6121 - loss: 1.2837 - val_accuracy: 0.5266 - val_loss: 1.8384
Epoch 18/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 20s 113ms/step - accuracy: 0.6250 - loss: 1.1885 - val_accuracy: 0.5391 - val_loss: 1.8310
Epoch 19/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.6380 - loss: 1.1769 - val_accuracy: 0.5289 - val_loss: 1.7745
Epoch 20/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 113ms/step - accuracy: 0.6875 - loss: 1.2660 - val_accuracy: 0.5312 - val_loss: 1.7825
Epoch 21/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.6759 - loss: 1.0812 - val_accuracy: 0.5773 - val_loss: 1.5958
Epoch 22/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 107ms/step - accuracy: 0.5938 - loss: 1.2888 - val_accuracy: 0.5570 - val_loss: 1.6019
Epoch 23/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.6963 - loss: 1.0281 - val_accuracy: 0.5203 - val_loss: 1.9618
Epoch 24/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 103ms/step - accuracy: 0.7500 - loss: 0.8295 - val_accuracy: 0.5109 - val_loss: 1.9918
```

```
Epoch 23/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.6963 - loss: 1.0281 - val_accuracy: 0.5203 - val_loss: 1.9618
Epoch 24/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 103ms/step - accuracy: 0.7500 - loss: 0.8295 - val_accuracy: 0.5109 - val_loss: 1.9918
Epoch 25/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 323s 2s/step - accuracy: 0.7020 - loss: 0.9887 - val_accuracy: 0.5539 - val_loss: 1.7550
Epoch 26/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 18s 102ms/step - accuracy: 0.7188 - loss: 1.0257 - val_accuracy: 0.5516 - val_loss: 1.7070
Epoch 27/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 262s 2s/step - accuracy: 0.7238 - loss: 0.8892 - val_accuracy: 0.5883 - val_loss: 1.6402
Epoch 28/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 19s 104ms/step - accuracy: 0.6875 - loss: 0.9230 - val_accuracy: 0.5906 - val_loss: 1.6457
Epoch 29/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 322s 2s/step - accuracy: 0.7471 - loss: 0.8053 - val_accuracy: 0.5734 - val_loss: 1.6608
Epoch 30/30
162/162 ━━━━━━━━━━━━━━━━━━━━ 21s 118ms/step - accuracy: 0.6875 - loss: 1.0026 - val_accuracy: 0.5656 - val_loss: 1.6610
```

Step 7: Evaluation

```
lt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()

plt.show()
```

Output:



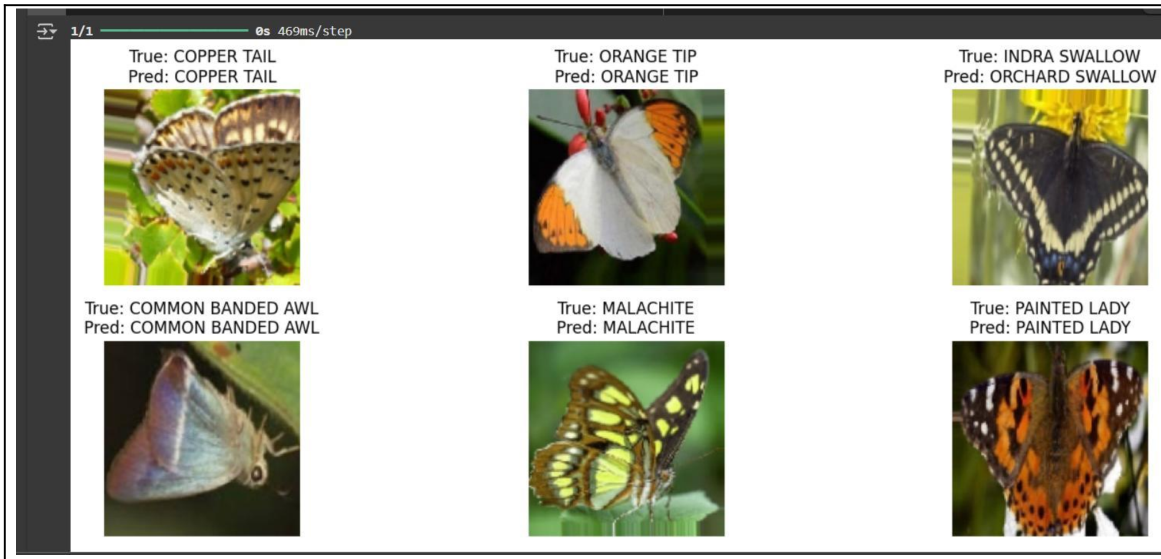
Testing Model & Data Prediction:

- Predictions on validation set
- np.argmax to convert probabilities to class labels
- Display true vs predicted labels using matplotlib
-

Testing Model & Data PredictionCode:

```
# Make predictions
val_images, val_labels = next(val_generator)
pred_labels = model.predict(val_images)
pred_labels = np.argmax(pred_labels, axis=1)
true_labels = np.argmax(val_labels, axis=1)

# Visualization
class_indices = val_generator.class_indices
class_names = {v: k for k, v in class_indices.items()}

def display_images(images, true_labels, pred_labels, class_names, num_images=9):
    plt.figure(figsize=(15, 8))
    for i in range(num_images):
        plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        true_label = class_names[int(true_labels[i])]
        pred_label = class_names[int(pred_labels[i])]
        plt.title(f"True: {true_label}\nPred: {pred_label}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

display_images(val_images, true_labels, pred_labels, class_names)
```

Output:



Applications:

- Biodiversity conservation
- Educational tools for species identification
- Automated classification in ecological surveys
- Assisting biologists in insect taxonomy

Conclusion:

This project successfully demonstrates the ability of CNNs to classify butterfly species based on image input. With good accuracy and visual validation, it sets a strong foundation for further enhancements using transfer learning or larger datasets.