```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os
from tqdm import tqdm
import kagglehub

# Deep Learning libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, optimizers, callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import imagenet_utils
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import joblib

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Check GPU availability
print("GPU Available: ", tf.config.list_physical_devices('GPU'))
print("TensorFlow version:", tf.__version__)

# Download dataset
print("Downloading dataset...")
data_path = kagglehub.dataset_download('phucthaiv02/butterfly-image-classification')
print(f"Dataset downloaded to: {data_path}")

# Explore dataset structure
print("\nExploring dataset structure...")
for root, dirs, files in os.walk(data_path):
    level = root.replace(data_path, '').count(os.sep)
    indent = ' ' * 2 * level
    print(f"{indent}{os.path.basename(root)}/")
    subindent = ' ' * 2 * (level + 1)
    for file in files[:5]:  # Show only first 5 files
        print(f"{subindent}{file}")
    if len(files) > 5:
        print(f"{subindent}... and {len(files)-5} more files")
```

```
# Load CSV files
csv_loaded = False
try:
    csv_files = ['Training_set.csv', 'Testing_set.csv', 'train.csv', 'test.csv']
    found_csvs = []

    for root, dirs, files in os.walk(data_path):
        for file in files:
            if file.endswith('.csv'):
                found_csvs.append(os.path.join(root, file))

    print(f"\nFound CSV files: {found_csvs}")

    if found_csvs:
        for csv_file in found_csvs:
            try:
                df = pd.read_csv(csv_file)
                print(f"\nLoaded {csv_file}:")
                print(f"Shape: {df.shape}")
                print(f"Columns: {df.columns.tolist()}")
                print(df.head())

                if 'training' in csv_file.lower() or 'train' in csv_file.lower():
                    train_df = df
                elif 'testing' in csv_file.lower() or 'test' in csv_file.lower():
                    test_df = df

                csv_loaded = True
            except Exception as e:
                print(f"Error loading {csv_file}: {e}")

except Exception as e:
    print(f"Error exploring CSV files: {e}")

if not csv_loaded:
    print("No CSV files found or loaded. Will work with directory structure.")
    train_df = None
    test_df = None

# Reduced image size for faster training
IMG_SIZE = (64, 64)  # Reduced from 224x224 for faster training
BATCH_SIZE = 64
EPOCHS = 30

print(f"Image size set to: {IMG_SIZE}")
print(f"Batch size: {BATCH_SIZE}")
```

```
print(f" Batch size: {BATCH_SIZE}")

# Function to load and preprocess images for CNN
def load_and_preprocess_images_cnn(base_path, df=None, img_size=(224, 224)):
    """
    Load and preprocess images for CNN (keeping spatial structure)
    """
    images = []
    labels = []

    print(f"Loading images from: {base_path}")

    if df is not None and not df.empty:
        print("Using CSV file for image paths and labels")
        for idx, row in tqdm(df.iterrows(), total=len(df), desc="Loading images"):
            try:
                filename_col = None
                for col in ['filename', 'image', 'path', 'file']:
                    if col in row.index:
                        filename_col = col
                        break

                label_col = None
                for col in ['label', 'class', 'category', 'target']:
                    if col in row.index:
                        label_col = col
                        break

                if filename_col is None or label_col is None:
                    print(f"Could not find filename or label columns. Available columns: {row.index.tolist()}")
                    continue

                img_path = os.path.join(base_path, str(row[filename_col]))

                possible_paths = [
                    img_path,
                    os.path.join(base_path, 'train', str(row[filename_col])),
                    os.path.join(base_path, 'test', str(row[filename_col])),
                    os.path.join(base_path, str(row[label_col]), str(row[filename_col]))
                ]

                img_found = False
                for path in possible_paths:
                    if os.path.exists(path):
                        img_path = path
                        img_found = True
                        break
```

```
            if not img_found:
                continue

            # Load image
            img = cv2.imread(img_path)
            if img is None:
                continue

            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # Resize image
            img = cv2.resize(img, img_size)

            # Normalize pixel values to [0, 1]
            img = img.astype(np.float32) / 255.0

            images.append(img)
            labels.append(str(row[label_col]))

        except Exception as e:
            print(f"Error processing image {idx}: {e}")
            continue
else:
    print("Using directory structure for image loading")
    subdirs = []
    for item in os.listdir(base_path):
        item_path = os.path.join(base_path, item)
        if os.path.isdir(item_path):
            subdirs.append(item)

    print(f"Found subdirectories: {subdirs}")

    if not subdirs:
        print("No subdirectories found, looking for images directly")
        for img_file in os.listdir(base_path):
            if img_file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
                try:
                    img_path = os.path.join(base_path, img_file)
                    img = cv2.imread(img_path)
                    if img is None:
                        continue

                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    img = cv2.resize(img, img_size)
                    img = img.astype(np.float32) / 255.0
```

```
                            images.append(img)
                            labels.append('unknown')
                        except Exception as e:
                            print(f"Error processing {img_file}: {e}")
                            continue
            else:
                for class_folder in subdirs:
                    class_path = os.path.join(base_path, class_folder)
                    if os.path.isdir(class_path):
                        print(f"Processing class: {class_folder}")
                        img_files = [f for f in os.listdir(class_path)
                                   if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff'))]

                        print(f"Found {len(img_files)} images in {class_folder}")

                        for img_file in tqdm(img_files, desc=f"Loading {class_folder}"):
                            try:
                                img_path = os.path.join(class_path, img_file)
                                img = cv2.imread(img_path)
                                if img is None:
                                    continue

                                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                                img = cv2.resize(img, img_size)
                                img = img.astype(np.float32) / 255.0

                                images.append(img)
                                labels.append(class_folder)
                            except Exception as e:
                                print(f"Error processing {img_file}: {e}")
                                continue

        print(f"Loaded {len(images)} images with {len(set(labels))} unique labels")
        if len(set(labels)) > 0:
            print(f"Label distribution: {dict(pd.Series(labels).value_counts())}")

        return np.array(images), np.array(labels)

    # Load training data
    print("\nLoading training images...")
    X_train = np.array([])
    y_train = np.array([])

    possible_train_paths = [
        os.path.join(data_path, 'train'),
```

```
        os.path.join(data_path, 'training'),
        os.path.join(data_path, 'Training'),
        data_path
    ]

    train_loaded = False
    for train_path in possible_train_paths:
        if os.path.exists(train_path):
            print(f"Trying to load from: {train_path}")
            try:
                if 'train_df' in locals() and train_df is not None:
                    X_train, y_train = load_and_preprocess_images_cnn(train_path, train_df, IMG_SIZE)
                else:
                    X_train, y_train = load_and_preprocess_images_cnn(train_path, None, IMG_SIZE)

                if len(X_train) > 0:
                    train_loaded = True
                    print(f"Successfully loaded training data from: {train_path}")
                    break
            except Exception as e:
                print(f"Error loading from {train_path}: {e}")
                continue

    if not train_loaded or len(X_train) == 0:
        print("No training images found. Checking entire dataset structure...")
        X_train, y_train = load_and_preprocess_images_cnn(data_path, None, IMG_SIZE)

    print(f"Training data loaded: {X_train.shape if len(X_train) > 0 else 'No data'}")
    print(f"Unique labels: {len(np.unique(y_train)) if len(y_train) > 0 else 0}")

    if len(X_train) == 0:
        print("\n" + "="*60)
        print("ERROR: No training images could be loaded!")
        print("Please check dataset structure")
        print("="*60)
        exit()

    # Load testing data
    print("\nLoading testing images...")
    X_test = np.array([])
    y_test = np.array([])

    test_has_labels = False
    if 'test_df' in locals() and test_df is not None:
        test_has_labels = 'label' in test_df.columns
```

```
    print(f"Test CSV has labels: {test_has_labels}")

    if test_has_labels:
        test_path = os.path.join(data_path, 'test')
        print(f"Loading test data with labels from: {test_path}")
        X_test, y_test = load_and_preprocess_images_cnn(test_path, test_df, IMG_SIZE)
        print(f"Test data loaded: {X_test.shape}")
    else:
        print("Test data doesn't have labels. Splitting training data for evaluation...")

        if len(np.unique(y_train)) > 1:
            X_train, X_test, y_train, y_test = train_test_split(
                X_train, y_train, test_size=0.2, random_state=42, stratify=y_train
            )
        else:
            X_train, X_test, y_train, y_test = train_test_split(
                X_train, y_train, test_size=0.2, random_state=42
            )
        print("Created test split from training data")

    print(f"Final training data: {X_train.shape}")
    print(f"Final testing data: {X_test.shape}")

    if len(X_train) == 0 or len(X_test) == 0:
        print("\n" + "="*60)
        print("ERROR: No valid training or testing data!")
        print("="*60)
        exit()

    # Encode labels
    print("\nEncoding labels...")
    label_encoder = LabelEncoder()
    y_train_encoded = label_encoder.fit_transform(y_train)
    y_test_encoded = label_encoder.transform(y_test)

    # One-hot encode for multi-class classification
    num_classes = len(label_encoder.classes_)
    y_train_onehot = keras.utils.to_categorical(y_train_encoded, num_classes)
    y_test_onehot = keras.utils.to_categorical(y_test_encoded, num_classes)

    print(f"Classes: {label_encoder.classes_}")
    print(f"Number of classes: {num_classes}")

    # Data validation split
    X_train, X_val, y_train_onehot, y_val_onehot = train_test_split(
        X_train, y_train_onehot, test_size=0.2, random_state=42, stratify=y_train_encoded
```

```
)

print(f"Training data: {X_train.shape}")
print(f"Validation data: {X_val.shape}")
print(f"Test data: {X_test.shape}")

# Define AlexNet architecture
def create_alexnet(input_shape, num_classes):
    """
    Create modified AlexNet model for smaller images
    """
    model = models.Sequential([
        # First Convolutional Block - adjusted for smaller input
        layers.Conv2D(64, (7, 7), strides=2, activation='relu',
                     input_shape=input_shape, padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2), strides=2),

        # Second Convolutional Block
        layers.Conv2D(128, (5, 5), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2), strides=2),

        # Third Convolutional Block
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),

        # Fourth Convolutional Block
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),

        # Fifth Convolutional Block
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2), strides=2),

        # Fully Connected Layers - reduced size
        layers.Flatten(),
        layers.Dense(1024, activation='relu'),  # Reduced from 4096
        layers.Dropout(0.5),
        layers.Dense(512, activation='relu'),   # Reduced from 4096
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])

    return model
```

```
    # Create AlexNet model
    print("\nCreating AlexNet model...")
    input_shape = (IMG_SIZE[0], IMG_SIZE[1], 3)
    model = create_alexnet(input_shape, num_classes)

    # Compile the model
    model.compile(
        optimizer=optimizers.Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy', 'top_k_categorical_accuracy']
    )

    # Model summary
    print("\nModel Architecture:")
    model.summary()

    # Data augmentation
    datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        zoom_range=0.2,
        shear_range=0.2,
        fill_mode='nearest'
    )

    # Callbacks
    callbacks_list = [
        callbacks.EarlyStopping(
            monitor='val_accuracy',
            patience=10,
            restore_best_weights=True,
            verbose=1
        ),
        callbacks.ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=5,
            min_lr=1e-7,
            verbose=1
        ),
        callbacks.ModelCheckpoint(
            'best_alexnet_butterfly.h5',
            monitor='val_accuracy',
```

```
            save_best_only=True,
            verbose=1
        )
    ]

    # Train the model
    print("\nTraining AlexNet model...")
    print("This may take a while...")

    history = model.fit(
        datagen.flow(X_train, y_train_onehot, batch_size=BATCH_SIZE),
        steps_per_epoch=len(X_train) // BATCH_SIZE,
        epochs=EPOCHS,
        validation_data=(X_val, y_val_onehot),
        callbacks=callbacks_list,
        verbose=1
    )

    print("Training completed!")

    # Evaluate the model
    print("\nEvaluating model on test data...")
    test_loss, test_accuracy, test_top_k = model.evaluate(X_test, y_test_onehot, verbose=0)
    print(f"Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")
    print(f"Test Top-K Accuracy: {test_top_k:.4f} ({test_top_k*100:.2f}%)")

    # Make predictions
    print("\nMaking predictions...")
    y_pred_proba = model.predict(X_test)
    y_pred = np.argmax(y_pred_proba, axis=1)
    y_true = np.argmax(y_test_onehot, axis=1)

    # Classification report
    print("\nClassification Report:")
    class_names = label_encoder.classes_
    print(classification_report(y_true, y_pred, target_names=class_names))

    # Confusion Matrix
    print("\nConfusion Matrix:")
    cm = confusion_matrix(y_true, y_pred)
    print(cm)

    # Plot training history
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # Accuracy plot
```

```
    axes[0, 0].plot(history.history['accuracy'], label='Training Accuracy')
    axes[0, 0].plot(history.history['val_accuracy'], label='Validation Accuracy')
    axes[0, 0].set_title('Model Accuracy')
    axes[0, 0].set_xlabel('Epoch')
    axes[0, 0].set_ylabel('Accuracy')
    axes[0, 0].legend()
    axes[0, 0].grid(True)

    # Loss plot
    axes[0, 1].plot(history.history['loss'], label='Training Loss')
    axes[0, 1].plot(history.history['val_loss'], label='Validation Loss')
    axes[0, 1].set_title('Model Loss')
    axes[0, 1].set_xlabel('Epoch')
    axes[0, 1].set_ylabel('Loss')
    axes[0, 1].legend()
    axes[0, 1].grid(True)

    # Confusion matrix heatmap
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names, ax=axes[1, 0])
    axes[1, 0].set_title('Confusion Matrix')
    axes[1, 0].set_xlabel('Predicted Labels')
    axes[1, 0].set_ylabel('True Labels')

    # Top-K accuracy plot
    if 'top_k_categorical_accuracy' in history.history:
        axes[1, 1].plot(history.history['top_k_categorical_accuracy'], label='Training Top-K Accuracy')
        axes[1, 1].plot(history.history['val_top_k_categorical_accuracy'], label='Validation Top-K Accuracy')
        axes[1, 1].set_title('Model Top-K Accuracy')
        axes[1, 1].set_xlabel('Epoch')
        axes[1, 1].set_ylabel('Top-K Accuracy')
        axes[1, 1].legend()
        axes[1, 1].grid(True)

    plt.tight_layout()
    plt.show()

    # Sample predictions with confidence
    def predict_and_visualize_samples(model, X_test, y_test_onehot, label_encoder, n_samples=6):
        """Visualize sample predictions"""
        indices = np.random.choice(len(X_test), n_samples, replace=False)

        fig, axes = plt.subplots(2, 3, figsize=(15, 10))
        axes = axes.ravel()

        for i, idx in enumerate(indices):
```

```
        # Get image and prediction
        img = X_test[idx]
        true_label_idx = np.argmax(y_test_onehot[idx])
        true_label = label_encoder.inverse_transform([true_label_idx])[0]

        # Make prediction
        pred_proba = model.predict(np.expand_dims(img, axis=0))[0]
        pred_idx = np.argmax(pred_proba)
        pred_label = label_encoder.inverse_transform([pred_idx])[0]
        confidence = pred_proba[pred_idx]

        # Plot
        axes[i].imshow(img)
        axes[i].set_title(f'True: {true_label}\nPred: {pred_label}\nConf: {confidence:.3f}')
        axes[i].axis('off')

        # Color border based on correctness
        if true_label == pred_label:
            axes[i].patch.set_edgecolor('green')
        else:
            axes[i].patch.set_edgecolor('red')
        axes[i].patch.set_linewidth(3)

    plt.tight_layout()
    plt.show()

# Visualize sample predictions
predict_and_visualize_samples(model, X_test, y_test_onehot, label_encoder)

# Model analysis
def analyze_model_performance(y_true, y_pred, y_pred_proba, class_names):
    """Analyze model performance in detail"""

    print("\n" + "="*60)
    print("DETAILED PERFORMANCE ANALYSIS")
    print("="*60)

    # Overall metrics
    accuracy = accuracy_score(y_true, y_pred)
    print(f"Overall Accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")

    # Per-class analysis
    print("\nPer-class Performance:")
    print("-" * 40)

    for i, class_name in enumerate(class_names):
```

```
            class_mask = (y_true == i)
            if np.sum(class_mask) > 0:
                class_accuracy = accuracy_score(y_true[class_mask], y_pred[class_mask])
                class_samples = np.sum(class_mask)
                avg_confidence = np.mean(np.max(y_pred_proba[class_mask], axis=1))

                print(f"{class_name:15} | Acc: {class_accuracy:.3f} | "
                      f"Samples: {class_samples:3d} | Avg Conf: {avg_confidence:.3f}")

        # Confidence distribution
        confidences = np.max(y_pred_proba, axis=1)
        correct_mask = (y_true == y_pred)

        print(f"\nConfidence Statistics:")
        print(f"Correct predictions - Mean: {np.mean(confidences[correct_mask]):.3f}, "
              f"Std: {np.std(confidences[correct_mask]):.3f}")
        print(f"Wrong predictions   - Mean: {np.mean(confidences[~correct_mask]):.3f}, "
              f"Std: {np.std(confidences[~correct_mask]):.3f}")

    # Perform detailed analysis
    analyze_model_performance(y_true, y_pred, y_pred_proba, class_names)

    # Save model and components
    print("\nSaving model...")
    model.save('alexnet_butterfly_classifier.h5')

    # Save preprocessing components
    joblib.dump({
        'label_encoder': label_encoder,
        'img_size': IMG_SIZE,
        'num_classes': num_classes,
        'class_names': class_names
    }, 'alexnet_preprocessing.pkl')

    print("Model and preprocessing components saved!")

    # Final summary
    print("\n" + "="*60)
    print("ALEXNET MODEL SUMMARY")
    print("="*60)
    print(f"Dataset: Butterfly Image Classification")
    print(f"Architecture: AlexNet CNN")
    print(f"Training samples: {len(X_train)}")
    print(f"Validation samples: {len(X_val)}")
    print(f"Testing samples: {len(X_test)}")
    print(f"Number of classes: {num_classes}")
```

```
    print(f"Image size: {IMG_SIZE}")
    print(f"Batch size: {BATCH_SIZE}")
    print(f"Epochs trained: {len(history.history['loss'])}")
    print(f"Final test accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")
    print(f"Final test top-k accuracy: {test_top_k:.4f} ({test_top_k*100:.2f}%)")
    print(f"Total parameters: {model.count_params():,}")
    print("="*60)

    print(f"\nModel files saved:")
    print("- alexnet_butterfly_classifier.h5")
    print("- alexnet_preprocessing.pkl")
    print("- best_alexnet_butterfly.h5 (best checkpoint)")
```

```
GPU Available:  []
TensorFlow version: 2.18.0
Downloading dataset...
Dataset downloaded to: /kaggle/input/butterfly-image-classification

Exploring dataset structure...
butterfly-image-classification/
  Training_set.csv
  Testing_set.csv
  test/
    Image_747.jpg
    Image_561.jpg
    Image_345.jpg
    Image_2566.jpg
    Image_1593.jpg
    ... and 2781 more files
  train/
    Image_4378.jpg
    Image_5576.jpg
    Image_6267.jpg
    Image_747.jpg
    Image_5775.jpg
    ... and 6494 more files

Found CSV files: ['/kaggle/input/butterfly-image-classification/Training_set.csv', '/kaggle/input/butterfly-image-classification/Testing_set.c

Loaded /kaggle/input/butterfly-image-classification/Training_set.csv:
Shape: (6499, 2)
Columns: ['filename', 'label']
      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

Loaded /kaggle/input/butterfly-image-classification/Testing_set.csv:
Shape: (2786, 1)
Columns: ['filename']
      filename
0  Image_1.jpg
1  Image_2.jpg
2  Image_3.jpg
3  Image_4.jpg
4  Image_5.jpg
Image size set to: (64, 64)
Batch size: 64

Loading training images...
Trying to load from: /kaggle/input/butterfly-image-classification/train
```

```
Loading images from: /kaggle/input/butterfly-image-classification/train
Using CSV file for image paths and labels
Loading images: 100%|██████████| 6499/6499 [00:14<00:00, 459.71it/s]
Loaded 6499 images with 75 unique labels
Label distribution: {'MOURNING CLOAK': np.int64(131), 'SLEEPY ORANGE': np.int64(107), 'ATALA': np.int64(100), 'BROWN SIPROETA': np.int64(99),
Successfully loaded training data from: /kaggle/input/butterfly-image-classification/train
Training data loaded: (6499, 64, 64, 3)
Unique labels: 75

Loading testing images...
Test CSV has labels: False
Test data doesn't have labels. Splitting training data for evaluation...
Created test split from training data
Final training data: (5199, 64, 64, 3)
Final testing data: (1300, 64, 64, 3)

Encoding labels...
Classes: ['ADONIS' 'AFRICAN GIANT SWALLOWTAIL' 'AMERICAN SNOOT' 'AN 88' 'APPOLLO'
 'ATALA' 'BANDED ORANGE HELICONIAN' 'BANDED PEACOCK' 'BECKERS WHITE'
 'BLACK HAIRSTREAK' 'BLUE MORPHO' 'BLUE SPOTTED CROW' 'BROWN SIPROETA'
 'CABBAGE WHITE' 'CAIRNS BIRDWING' 'CHECQUERED SKIPPER' 'CHESTNUT'
 'CLEOPATRA' 'CLODIUS PARNASSIAN' 'CLOUDED SULPHUR' 'COMMON BANDED AWL'
 'COMMON WOOD-NYMPH' 'COPPER TAIL' 'CRECENT' 'CRIMSON PATCH'
 'DANAID EGGFLY' 'EASTERN COMA' 'EASTERN DAPPLE WHITE'
 'EASTERN PINE ELFIN' 'ELBOWED PIERROT' 'GOLD BANDED' 'GREAT EGGFLY'
 'GREAT JAY' 'GREEN CELLED CATTLEHEART' 'GREY HAIRSTREAK' 'INDRA SWALLOW'
 'IPHICLUS SISTER' 'JULIA' 'LARGE MARBLE' 'MALACHITE' 'MANGROVE SKIPPER'
 'MESTRA' 'METALMARK' 'MILBERTS TORTOISESHELL' 'MONARCH' 'MOURNING CLOAK'
 'ORANGE OAKLEAF' 'ORANGE TIP' 'ORCHARD SWALLOW' 'PAINTED LADY'
 'PAPER KITE' 'PEACOCK' 'PINE WHITE' 'PIPEVINE SWALLOW' 'POPINJAY'
 'PURPLE HAIRSTREAK' 'PURPLISH COPPER' 'QUESTION MARK' 'RED ADMIRAL'
 'RED CRACKER' 'RED POSTMAN' 'RED SPOTTED PURPLE' 'SCARCE SWALLOW'
 'SILVER SPOT SKIPPER' 'SLEEPY ORANGE' 'SOOTYWING' 'SOUTHERN DOGFACE'
 'STRAITED QUEEN' 'TROPICAL LEAFWING' 'TWO BARRED FLASHER' 'ULYSES'
 'VICEROY' 'WOOD SATYR' 'YELLOW SWALLOW TAIL' 'ZEBRA LONG WING']
Number of classes: 75
Training data: (4159, 64, 64, 3)
Validation data: (1040, 64, 64, 3)
Test data: (1300, 64, 64, 3)

Creating AlexNet model...

Model Architecture:
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 32, 32, 64)	9,472

batch_normalization_5 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_6 (Conv2D)	(None, 16, 16, 128)	204,928
batch_normalization_6 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_7 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_7 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_8 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_8 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_9 (Conv2D)	(None, 8, 8, 128)	295,040
batch_normalization_9 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 1024)	2,098,176
dropout_2 (Dropout)	(None, 1024)	0
dense_4 (Dense)	(None, 512)	524,800
dropout_3 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 75)	38,475

Total params: 4,059,467 (15.49 MB)
Trainable params: 4,057,803 (15.48 MB)
Non-trainable params: 1,664 (6.50 KB)

```
Training AlexNet model...
This may take a while...
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should
  self._warn_if_super_not_called()
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.0266 - loss: 5.2140 - top_k_categorical_accuracy: 0.1014
Epoch 1: val_accuracy improved from -inf to 0.01923, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 110s 2s/step - accuracy: 0.0267 - loss: 5.2053 - top_k_categorical_accuracy: 0.1018 - val_accuracy: 0.0192 - val_lo
```

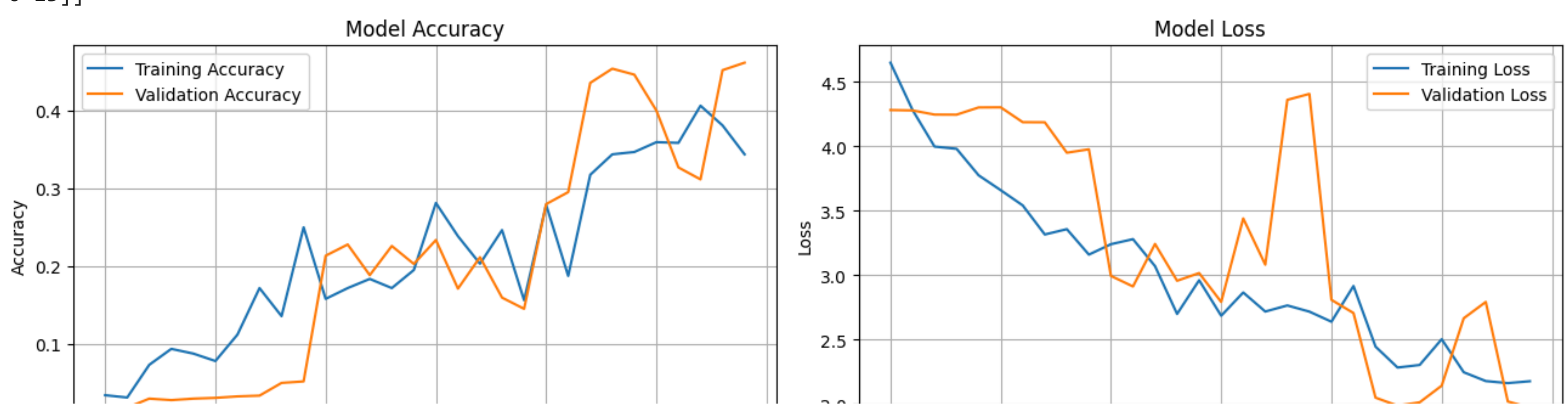
```
Epoch 2/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:20 1s/step - accuracy: 0.0312 - loss: 4.2813 - top_k_categorical_accuracy: 0.1719/usr/local/lib/python3.11/dist-p
  self._interrupted_warning()

Epoch 2: val_accuracy did not improve from 0.01923
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 163ms/step - accuracy: 0.0312 - loss: 4.2813 - top_k_categorical_accuracy: 0.1719 - val_accuracy: 0.0183 - val_
Epoch 3/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.0683 - loss: 4.0697 - top_k_categorical_accuracy: 0.2221
Epoch 3: val_accuracy improved from 0.01923 to 0.02981, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 99s 2s/step - accuracy: 0.0683 - loss: 4.0686 - top_k_categorical_accuracy: 0.2225 - val_accuracy: 0.0298 - val_los
Epoch 4/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:22 1s/step - accuracy: 0.0938 - loss: 3.9792 - top_k_categorical_accuracy: 0.2656
Epoch 4: val_accuracy did not improve from 0.02981
64/64 ━━━━━━━━━━━━━━━━━━━━ 8s 107ms/step - accuracy: 0.0938 - loss: 3.9792 - top_k_categorical_accuracy: 0.2656 - val_accuracy: 0.0279 - val_l
Epoch 5/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.0851 - loss: 3.8191 - top_k_categorical_accuracy: 0.2970
Epoch 5: val_accuracy did not improve from 0.02981
64/64 ━━━━━━━━━━━━━━━━━━━━ 105s 2s/step - accuracy: 0.0851 - loss: 3.8184 - top_k_categorical_accuracy: 0.2972 - val_accuracy: 0.0298 - val_lo
Epoch 6/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:17 1s/step - accuracy: 0.0781 - loss: 3.6581 - top_k_categorical_accuracy: 0.2812
Epoch 6: val_accuracy improved from 0.02981 to 0.03077, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 7s 88ms/step - accuracy: 0.0781 - loss: 3.6581 - top_k_categorical_accuracy: 0.2812 - val_accuracy: 0.0308 - val_lo
Epoch 7/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.1154 - loss: 3.5583 - top_k_categorical_accuracy: 0.3649
Epoch 7: val_accuracy improved from 0.03077 to 0.03269, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 132s 2s/step - accuracy: 0.1154 - loss: 3.5580 - top_k_categorical_accuracy: 0.3650 - val_accuracy: 0.0327 - val_lo
Epoch 8/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:33 1s/step - accuracy: 0.1719 - loss: 3.3141 - top_k_categorical_accuracy: 0.4688
Epoch 8: val_accuracy improved from 0.03269 to 0.03365, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 167ms/step - accuracy: 0.1719 - loss: 3.3141 - top_k_categorical_accuracy: 0.4688 - val_accuracy: 0.0337 - val_
Epoch 9/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.1281 - loss: 3.3866 - top_k_categorical_accuracy: 0.4167
Epoch 9: val_accuracy improved from 0.03365 to 0.05000, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 100s 2s/step - accuracy: 0.1282 - loss: 3.3861 - top_k_categorical_accuracy: 0.4168 - val_accuracy: 0.0500 - val_lo
Epoch 10/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:18 1s/step - accuracy: 0.2500 - loss: 3.1578 - top_k_categorical_accuracy: 0.4531
Epoch 10: val_accuracy improved from 0.05000 to 0.05192, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 166ms/step - accuracy: 0.2500 - loss: 3.1578 - top_k_categorical_accuracy: 0.4531 - val_accuracy: 0.0519 - val_
Epoch 11/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.1703 - loss: 3.2142 - top_k_categorical_accuracy: 0.4869
Epoch 11: val_accuracy improved from 0.05192 to 0.21346, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 136s 2s/step - accuracy: 0.1702 - loss: 3.2146 - top_k_categorical_accuracy: 0.4867 - val_accuracy: 0.2135 - val_lo
Epoch 12/30
```

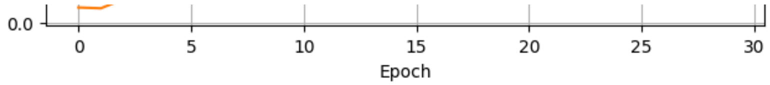
```
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:19 1s/step - accuracy: 0.1719 - loss: 3.2777 - top_k_categorical_accuracy: 0.4844
Epoch 12: val_accuracy improved from 0.21346 to 0.22788, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 166ms/step - accuracy: 0.1719 - loss: 3.2777 - top_k_categorical_accuracy: 0.4844 - val_accuracy: 0.2279 - val_
Epoch 13/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.1873 - loss: 3.0601 - top_k_categorical_accuracy: 0.5292
Epoch 13: val_accuracy did not improve from 0.22788
64/64 ━━━━━━━━━━━━━━━━━━━━ 103s 2s/step - accuracy: 0.1873 - loss: 3.0602 - top_k_categorical_accuracy: 0.5290 - val_accuracy: 0.1885 - val_lo
Epoch 14/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:20 1s/step - accuracy: 0.1719 - loss: 2.6971 - top_k_categorical_accuracy: 0.6562
Epoch 14: val_accuracy did not improve from 0.22788
64/64 ━━━━━━━━━━━━━━━━━━━━ 7s 85ms/step - accuracy: 0.1719 - loss: 2.6971 - top_k_categorical_accuracy: 0.6562 - val_accuracy: 0.2260 - val_lo
Epoch 15/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.2001 - loss: 2.9395 - top_k_categorical_accuracy: 0.5613
Epoch 15: val_accuracy did not improve from 0.22788
64/64 ━━━━━━━━━━━━━━━━━━━━ 138s 2s/step - accuracy: 0.2000 - loss: 2.9398 - top_k_categorical_accuracy: 0.5612 - val_accuracy: 0.2029 - val_lo
Epoch 16/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:18 1s/step - accuracy: 0.2812 - loss: 2.6830 - top_k_categorical_accuracy: 0.6719
Epoch 16: val_accuracy improved from 0.22788 to 0.23365, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 166ms/step - accuracy: 0.2812 - loss: 2.6830 - top_k_categorical_accuracy: 0.6719 - val_accuracy: 0.2337 - val_
Epoch 17/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.2343 - loss: 2.8715 - top_k_categorical_accuracy: 0.5786
Epoch 17: val_accuracy did not improve from 0.23365
64/64 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.2343 - loss: 2.8713 - top_k_categorical_accuracy: 0.5787 - val_accuracy: 0.1712 - val_lo
Epoch 18/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 2:02 2s/step - accuracy: 0.2031 - loss: 2.7158 - top_k_categorical_accuracy: 0.5938
Epoch 18: val_accuracy did not improve from 0.23365
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 163ms/step - accuracy: 0.2031 - loss: 2.7158 - top_k_categorical_accuracy: 0.5938 - val_accuracy: 0.2115 - val_
Epoch 19/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.2410 - loss: 2.7940 - top_k_categorical_accuracy: 0.6104
Epoch 19: val_accuracy did not improve from 0.23365
64/64 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.2411 - loss: 2.7935 - top_k_categorical_accuracy: 0.6105 - val_accuracy: 0.1596 - val_lo
Epoch 20/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:18 1s/step - accuracy: 0.1562 - loss: 2.7151 - top_k_categorical_accuracy: 0.6875
Epoch 20: val_accuracy did not improve from 0.23365
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 163ms/step - accuracy: 0.1562 - loss: 2.7151 - top_k_categorical_accuracy: 0.6875 - val_accuracy: 0.1452 - val_
Epoch 21/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.2633 - loss: 2.6477 - top_k_categorical_accuracy: 0.6407
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 21: val_accuracy improved from 0.23365 to 0.27981, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 105s 2s/step - accuracy: 0.2635 - loss: 2.6475 - top_k_categorical_accuracy: 0.6408 - val_accuracy: 0.2798 - val_lo
Epoch 22/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:17 1s/step - accuracy: 0.1875 - loss: 2.9141 - top_k_categorical_accuracy: 0.5625
Epoch 22: val_accuracy improved from 0.27981 to 0.29519, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 166ms/step - accuracy: 0.1875 - loss: 2.9141 - top_k_categorical_accuracy: 0.5625 - val_accuracy: 0.2952 - val_
Epoch 23/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.3064 - loss: 2.4748 - top_k_categorical_accuracy: 0.6956
```

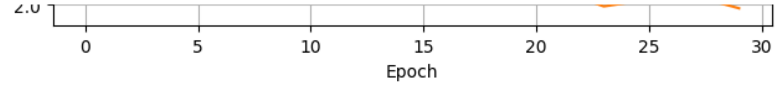
```
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.3064 - loss: 2.4748 - top_k_categorical_accuracy: 0.6956
Epoch 23: val_accuracy improved from 0.29519 to 0.43558, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 100s 2s/step - accuracy: 0.3066 - loss: 2.4743 - top_k_categorical_accuracy: 0.6957 - val_accuracy: 0.4356 - val_lo
Epoch 24/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:54 2s/step - accuracy: 0.3438 - loss: 2.2809 - top_k_categorical_accuracy: 0.7656
Epoch 24: val_accuracy improved from 0.43558 to 0.45385, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 167ms/step - accuracy: 0.3438 - loss: 2.2809 - top_k_categorical_accuracy: 0.7656 - val_accuracy: 0.4538 - val_
Epoch 25/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.3345 - loss: 2.2976 - top_k_categorical_accuracy: 0.7266
Epoch 25: val_accuracy did not improve from 0.45385
64/64 ━━━━━━━━━━━━━━━━━━━━ 106s 2s/step - accuracy: 0.3347 - loss: 2.2977 - top_k_categorical_accuracy: 0.7266 - val_accuracy: 0.4462 - val_lo
Epoch 26/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:31 1s/step - accuracy: 0.3594 - loss: 2.5008 - top_k_categorical_accuracy: 0.6562
Epoch 26: val_accuracy did not improve from 0.45385
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 163ms/step - accuracy: 0.3594 - loss: 2.5008 - top_k_categorical_accuracy: 0.6562 - val_accuracy: 0.4000 - val_
Epoch 27/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.3573 - loss: 2.2572 - top_k_categorical_accuracy: 0.7348
Epoch 27: val_accuracy did not improve from 0.45385
64/64 ━━━━━━━━━━━━━━━━━━━━ 104s 2s/step - accuracy: 0.3573 - loss: 2.2571 - top_k_categorical_accuracy: 0.7348 - val_accuracy: 0.3269 - val_lo
Epoch 28/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:21 1s/step - accuracy: 0.4062 - loss: 2.1759 - top_k_categorical_accuracy: 0.7656
Epoch 28: val_accuracy did not improve from 0.45385
64/64 ━━━━━━━━━━━━━━━━━━━━ 7s 86ms/step - accuracy: 0.4062 - loss: 2.1759 - top_k_categorical_accuracy: 0.7656 - val_accuracy: 0.3115 - val_lo
Epoch 29/30
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.3667 - loss: 2.1933 - top_k_categorical_accuracy: 0.7408
Epoch 29: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 29: val_accuracy did not improve from 0.45385
64/64 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.3670 - loss: 2.1928 - top_k_categorical_accuracy: 0.7410 - val_accuracy: 0.4519 - val_lo
Epoch 30/30
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1:21 1s/step - accuracy: 0.3438 - loss: 2.1746 - top_k_categorical_accuracy: 0.7500
Epoch 30: val_accuracy improved from 0.45385 to 0.46154, saving model to best_alexnet_butterfly.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered
64/64 ━━━━━━━━━━━━━━━━━━━━ 12s 167ms/step - accuracy: 0.3438 - loss: 2.1746 - top_k_categorical_accuracy: 0.7500 - val_accuracy: 0.4615 - val_
Restoring model weights from the end of the best epoch: 30.
Training completed!

Evaluating model on test data...
Test Accuracy: 0.4423 (44.23%)
Test Top-K Accuracy: 0.7692 (76.92%)

Making predictions...
41/41 ━━━━━━━━━━━━━━━━━━━━ 8s 180ms/step

Classification Report:
                           precision    recall  f1-score   support

                   ADONIS       0.80      0.67      0.73        18
AFRICAN GIANT SWALLOWTAIL       0.44      0.47      0.45        15
```

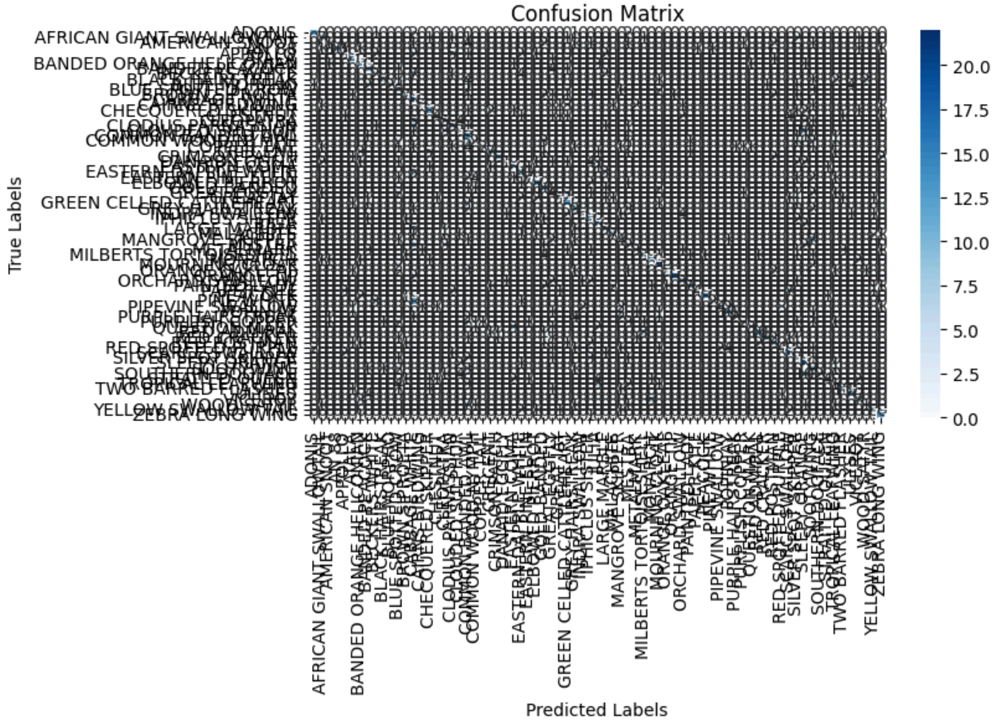
AMERICAN SNOOT	0.20	0.07	0.10	15
AN 88	1.00	0.59	0.74	17
APPOLLO	0.30	0.17	0.21	18
ATALA	0.65	0.75	0.70	20
BANDED ORANGE HELICONIAN	0.50	0.65	0.57	20
BANDED PEACOCK	0.44	0.65	0.52	17
BECKERS WHITE	0.22	0.12	0.16	16
BLACK HAIRSTREAK	0.75	0.35	0.48	17
BLUE MORPHO	1.00	0.13	0.24	15
BLUE SPOTTED CROW	0.18	0.35	0.24	17
BROWN SIPROETA	0.58	0.35	0.44	20
CABBAGE WHITE	0.22	1.00	0.36	18
CAIRNS BIRDWING	0.71	0.29	0.42	17
CHECQUERED SKIPPER	0.52	0.58	0.55	19
CHESTNUT	0.57	0.24	0.33	17
CLEOPATRA	0.36	0.26	0.30	19
CLODIUS PARNASSIAN	0.31	0.29	0.30	17
CLOUDED SULPHUR	0.21	0.17	0.19	18
COMMON BANDED AWL	0.19	0.76	0.30	17
COMMON WOOD-NYMPH	0.20	0.11	0.14	18
COPPER TAIL	0.67	0.11	0.18	19
CRECENT	0.58	0.35	0.44	20
CRIMSON PATCH	0.82	0.64	0.72	14
DANAID EGGFLY	0.33	0.05	0.09	19
EASTERN COMA	0.52	0.74	0.61	19
EASTERN DAPPLE WHITE	0.36	0.56	0.43	18
EASTERN PINE ELFIN	0.70	0.37	0.48	19
ELBOWED PIERROT	0.63	0.75	0.69	16
GOLD BANDED	0.33	0.67	0.44	15
GREAT EGGFLY	0.17	0.25	0.21	16
GREAT JAY	0.56	0.26	0.36	19
GREEN CELLED CATTLEHEART	0.34	0.67	0.45	18
GREY HAIRSTREAK	0.41	0.53	0.46	17
INDRA SWALLOW	0.67	0.38	0.48	16
IPHICLUS SISTER	0.38	0.63	0.47	19
JULIA	0.41	0.88	0.56	16
LARGE MARBLE	0.31	0.31	0.31	16
MALACHITE	0.64	0.47	0.54	15
MANGROVE SKIPPER	0.25	0.24	0.24	17
MESTRA	0.09	0.12	0.10	17
METALMARK	0.43	0.20	0.27	15
MILBERTS TORTOISESHELL	0.62	0.42	0.50	19
MONARCH	0.60	0.67	0.63	18
MOURNING CLOAK	0.67	0.85	0.75	26
ORANGE OAKLEAF	0.83	0.29	0.43	17
ORANGE TIP	0.94	0.89	0.92	19
ORCHARD SWALLOW	0.50	0.53	0.52	15
PAINTED LADY	0.70	0.44	0.54	16
PAPER KITE	0.80	0.22	0.35	18
PEACOCK	0.93	0.76	0.84	17

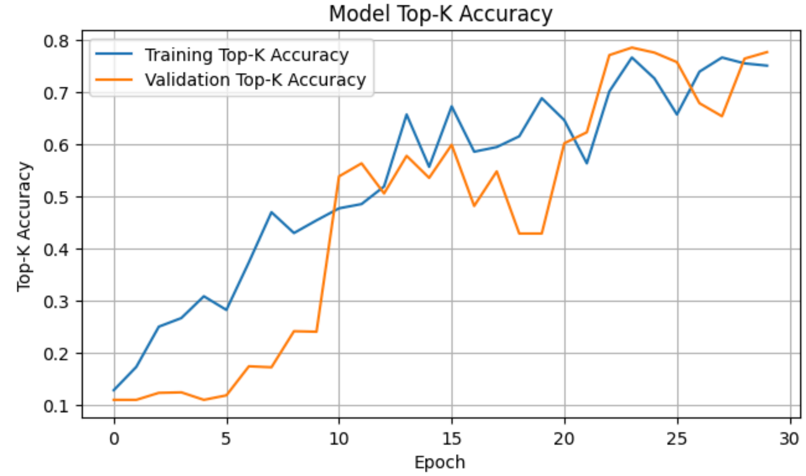
```
         PINE WHITE       0.67      0.12      0.20        17
    PIPEVINE SWALLOW       0.67      0.24      0.35        17
            POPINJAY       0.59      0.59      0.59        17
   PURPLE HAIRSTREAK       0.67      0.12      0.21        16
     PURPLISH COPPER       0.40      0.11      0.17        18
       QUESTION MARK       0.60      0.20      0.30        15
         RED ADMIRAL       0.77      0.62      0.69        16
         RED CRACKER       0.90      0.47      0.62        19
         RED POSTMAN       0.42      0.44      0.43        18
   RED SPOTTED PURPLE       0.50      0.24      0.32        17
      SCARCE SWALLOW       0.52      0.80      0.63        20
   SILVER SPOT SKIPPER       0.28      0.47      0.35        17
       SLEEPY ORANGE       0.35      0.82      0.49        22
           SOOTYWING       0.21      0.50      0.30        18
    SOUTHERN DOGFACE       1.00      0.12      0.21        17
      STRAITED QUEEN       0.67      0.24      0.35        17
    TROPICAL LEAFWING       0.08      0.06      0.07        17
   TWO BARRED FLASHER       0.67      0.67      0.67        15
              ULYSES       0.71      0.71      0.71        17
             VICEROY       0.50      0.44      0.47        16
          WOOD SATYR       0.25      0.07      0.11        14
   YELLOW SWALLOW TAIL       0.57      0.53      0.55        15
     ZEBRA LONG WING       0.59      0.87      0.70        15

            accuracy                           0.44      1300
           macro avg       0.52      0.44      0.43      1300
        weighted avg       0.52      0.44      0.43      1300


Confusion Matrix:
[[12  0  0 ...  0  0  0]
 [ 0  7  0 ...  0  0  0]
 [ 0  0  1 ...  0  0  0]
 ...
 [ 0  1  0 ...  1  0  0]
 [ 0  0  0 ...  0  8  0]
 [ 0  0  0 ...  0  0 13]]
```





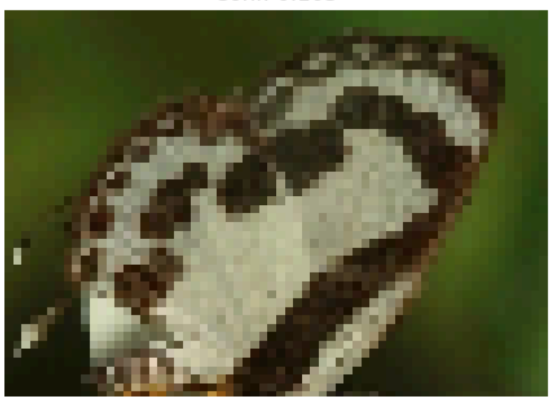







```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
```

True: ELBOWED PIERROT
Pred: MESTRA
Conf: 0.103



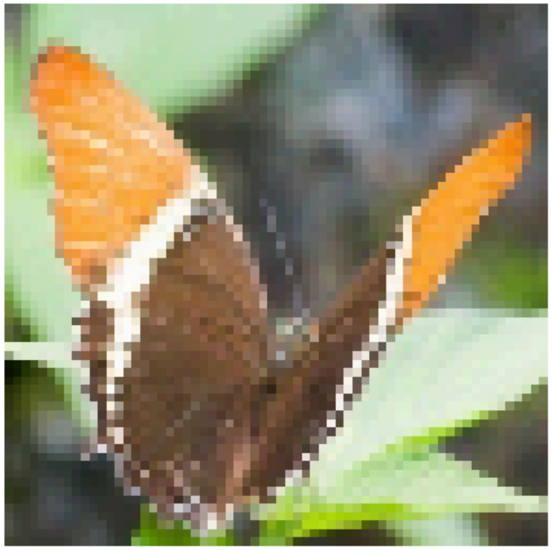
True: AN 88
Pred: AN 88
Conf: 0.815

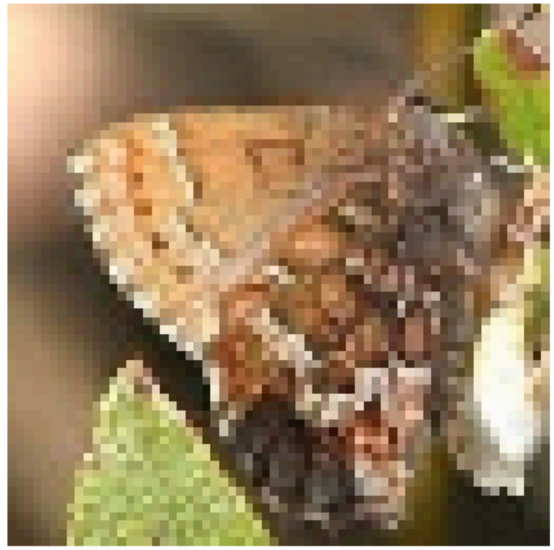
True: PURPLE HAIRSTREAK
Pred: GREY HAIRSTREAK
Conf: 0.509

True: BROWN SIPROETA
Pred: ORANGE TIP
Conf: 0.132

True: EASTERN PINE ELFIN
Pred: SILVER SPOT SKIPPER
Conf: 0.100

True: POPINJAY
Pred: POPINJAY
Conf: 0.325





```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered

============================================================
DETAILED PERFORMANCE ANALYSIS
============================================================
Overall Accuracy: 0.4423 (44.23%)

Per-class Performance:
----------------------------------------
ADONIS          | Acc: 0.667 | Samples:  18 | Avg Conf: 0.647
AFRICAN GIANT SWALLOWTAIL | Acc: 0.467 | Samples:  15 | Avg Conf: 0.475
AMERICAN SNOOT  | Acc: 0.067 | Samples:  15 | Avg Conf: 0.174
AN 88           | Acc: 0.588 | Samples:  17 | Avg Conf: 0.658
APPOLLO         | Acc: 0.167 | Samples:  18 | Avg Conf: 0.306
ATALA           | Acc: 0.750 | Samples:  20 | Avg Conf: 0.537
BANDED ORANGE HELICONIAN | Acc: 0.650 | Samples:  20 | Avg Conf: 0.441
BANDED PEACOCK  | Acc: 0.647 | Samples:  17 | Avg Conf: 0.632
BECKERS WHITE   | Acc: 0.125 | Samples:  16 | Avg Conf: 0.487
BLACK HAIRSTREAK | Acc: 0.353 | Samples:  17 | Avg Conf: 0.205
BLUE MORPHO     | Acc: 0.133 | Samples:  15 | Avg Conf: 0.368
BLUE SPOTTED CROW | Acc: 0.353 | Samples:  17 | Avg Conf: 0.329
BROWN SIPROETA  | Acc: 0.350 | Samples:  20 | Avg Conf: 0.279
CABBAGE WHITE   | Acc: 1.000 | Samples:  18 | Avg Conf: 0.771
```

```
CAIRNS BIRDWING | Acc: 0.294 | Samples:  17 | Avg Conf: 0.392
CHECQUERED SKIPPER | Acc: 0.579 | Samples:  19 | Avg Conf: 0.339
CHESTNUT        | Acc: 0.235 | Samples:  17 | Avg Conf: 0.321
CLEOPATRA       | Acc: 0.263 | Samples:  19 | Avg Conf: 0.493
CLODIUS PARNASSIAN | Acc: 0.294 | Samples:  17 | Avg Conf: 0.407
CLOUDED SULPHUR | Acc: 0.167 | Samples:  18 | Avg Conf: 0.440
COMMON BANDED AWL | Acc: 0.765 | Samples:  17 | Avg Conf: 0.298
COMMON WOOD-NYMPH | Acc: 0.111 | Samples:  18 | Avg Conf: 0.208
COPPER TAIL     | Acc: 0.105 | Samples:  19 | Avg Conf: 0.234
CRECENT         | Acc: 0.350 | Samples:  20 | Avg Conf: 0.313
CRIMSON PATCH   | Acc: 0.643 | Samples:  14 | Avg Conf: 0.447
DANAID EGGFLY   | Acc: 0.053 | Samples:  19 | Avg Conf: 0.306
EASTERN COMA    | Acc: 0.737 | Samples:  19 | Avg Conf: 0.295
EASTERN DAPPLE WHITE | Acc: 0.556 | Samples:  18 | Avg Conf: 0.457
EASTERN PINE ELFIN | Acc: 0.368 | Samples:  19 | Avg Conf: 0.291
ELBOWED PIERROT | Acc: 0.750 | Samples:  16 | Avg Conf: 0.678
GOLD BANDED     | Acc: 0.667 | Samples:  15 | Avg Conf: 0.446
GREAT EGGFLY    | Acc: 0.250 | Samples:  16 | Avg Conf: 0.276
GREAT JAY       | Acc: 0.263 | Samples:  19 | Avg Conf: 0.318
GREEN CELLED CATTLEHEART | Acc: 0.667 | Samples:  18 | Avg Conf: 0.651
GREY HAIRSTREAK | Acc: 0.529 | Samples:  17 | Avg Conf: 0.468
INDRA SWALLOW   | Acc: 0.375 | Samples:  16 | Avg Conf: 0.385
IPHICLUS SISTER | Acc: 0.632 | Samples:  19 | Avg Conf: 0.601
JULIA           | Acc: 0.875 | Samples:  16 | Avg Conf: 0.642
LARGE MARBLE    | Acc: 0.312 | Samples:  16 | Avg Conf: 0.494
MALACHITE       | Acc: 0.467 | Samples:  15 | Avg Conf: 0.373
MANGROVE SKIPPER | Acc: 0.235 | Samples:  17 | Avg Conf: 0.263
MESTRA          | Acc: 0.118 | Samples:  17 | Avg Conf: 0.439
METALMARK       | Acc: 0.200 | Samples:  15 | Avg Conf: 0.315
MILBERTS TORTOISESHELL | Acc: 0.421 | Samples:  19 | Avg Conf: 0.496
MONARCH         | Acc: 0.667 | Samples:  18 | Avg Conf: 0.331
MOURNING CLOAK  | Acc: 0.846 | Samples:  26 | Avg Conf: 0.723
ORANGE OAKLEAF  | Acc: 0.294 | Samples:  17 | Avg Conf: 0.382
ORANGE TIP      | Acc: 0.895 | Samples:  19 | Avg Conf: 0.843
ORCHARD SWALLOW | Acc: 0.533 | Samples:  15 | Avg Conf: 0.408
PAINTED LADY    | Acc: 0.438 | Samples:  16 | Avg Conf: 0.285
PAPER KITE      | Acc: 0.222 | Samples:  18 | Avg Conf: 0.451
PEACOCK         | Acc: 0.765 | Samples:  17 | Avg Conf: 0.403
PINE WHITE      | Acc: 0.118 | Samples:  17 | Avg Conf: 0.500
PIPEVINE SWALLOW | Acc: 0.235 | Samples:  17 | Avg Conf: 0.192
POPINJAY        | Acc: 0.588 | Samples:  17 | Avg Conf: 0.328
PURPLE HAIRSTREAK | Acc: 0.125 | Samples:  16 | Avg Conf: 0.299
PURPLISH COPPER | Acc: 0.111 | Samples:  18 | Avg Conf: 0.187
QUESTION MARK   | Acc: 0.200 | Samples:  15 | Avg Conf: 0.250
RED ADMIRAL     | Acc: 0.625 | Samples:  16 | Avg Conf: 0.556
RED CRACKER     | Acc: 0.474 | Samples:  19 | Avg Conf: 0.333
RED POSTMAN     | Acc: 0.444 | Samples:  18 | Avg Conf: 0.549
RED SPOTTED PURPLE | Acc: 0.235 | Samples:  17 | Avg Conf: 0.279
SCARCE SWALLOW  | Acc: 0.800 | Samples:  20 | Avg Conf: 0.566
SILVER SPOT SKIPPER | Acc: 0.471 | Samples:  17 | Avg Conf: 0.279
SLEEPY ORANGE   | Acc: 0.818 | Samples:  22 | Avg Conf: 0.404
```

```
SLEEPY ORANGE   | Acc: 0.818 | Samples:  22 | Avg Conf: 0.404
SOOTYWING       | Acc: 0.500 | Samples:  18 | Avg Conf: 0.337
SOUTHERN DOGFACE | Acc: 0.118 | Samples:  17 | Avg Conf: 0.384
STRAITED QUEEN  | Acc: 0.235 | Samples:  17 | Avg Conf: 0.247
TROPICAL LEAFWING | Acc: 0.059 | Samples:  17 | Avg Conf: 0.375
TWO BARRED FLASHER | Acc: 0.667 | Samples:  15 | Avg Conf: 0.273
ULYSES          | Acc: 0.706 | Samples:  17 | Avg Conf: 0.674
VICEROY         | Acc: 0.438 | Samples:  16 | Avg Conf: 0.402
WOOD SATYR      | Acc: 0.071 | Samples:  14 | Avg Conf: 0.287
YELLOW SWALLOW TAIL | Acc: 0.533 | Samples:  15 | Avg Conf: 0.450
ZEBRA LONG WING | Acc: 0.867 | Samples:  15 | Avg Conf: 0.781

Confidence Statistics:
Correct predictions - Mean: 0.546, Std: 0.271
Wrong predictions   - Mean: 0.316, Std: 0.188

Saving model...
Model and preprocessing components saved!

============================================================
ALEXNET MODEL SUMMARY
============================================================
Dataset: Butterfly Image Classification
Architecture: AlexNet CNN
Training samples: 4159
Validation samples: 1040
Testing samples: 1300
Number of classes: 75
Image size: (64, 64)
Batch size: 64
Epochs trained: 30
Final test accuracy: 0.4423 (44.23%)
Final test top-k accuracy: 0.7692 (76.92%)
```