MINI PROJECT

Humming Bird Classification using Convolutional Neural Network (CNN)

PROBLEM STATEMENT:

Hummingbirds are among the most diverse and visually striking bird species, often distinguished by their vibrant colors, unique feather patterns, and small size. Accurate identification of hummingbird species is essential for biodiversity studies, conservation efforts, and ecological research. Traditional methods of species identification, which rely on manual observation and expertise, are time-consuming and prone to human error.

With the increasing availability of image datasets and advancements in deep learning, CNNs have become a powerful tool for image-based classification tasks. Leveraging these techniques for hummingbird species identification could automate and significantly improve the accuracy of classification. The aim of this project is to develop a Convolutional Neural Network (CNN)-based system capable of accurately classifying images of hummingbirds into their respective species. The system should handle datasets of varying complexity and provide high accuracy for unseen data.

DATASET USED:

The dataset used for this project consists of images of three distinct types of hummingbirds: **Broadtailed (Female)**, **Broadtailed (Male)**, and **Rufous (Male)**. A total of 1,560 images were collected and categorized into three subsets: a **training set** containing 400 images per species (1,200 images in total), a **validation set** with 80 images per species (240 images in total), and a **test set** with 80 images per species (240 images in total). The images exhibit a diverse range of conditions, including different poses, orientations, lighting scenarios, and backgrounds, ensuring that the dataset reflects real-world variations. The dataset is balanced, with an equal number of images for each species in all subsets, which helps prevent bias during model training. All images were preprocessed by resizing them to 128x128 pixels and normalizing pixel values for consistent input to the CNN model.

IMPLEMENTATION AND CODE:

1. Importing the necessary libraries

```
import os
import numpy as np
import random as rn
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from PIL import Image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from plotly.subplots import make_subplots
import plotly.graph_objs as go
%matplotlib inline
import time
from keras import models
import tensorflow as tf
from tensorflow.keras import applications as app
from tensorflow.keras.models import Sequential # Sequential model
initialization
import tensorflow.keras.models as Models
from tensorflow.keras.layers import Dense,BatchNormalization,Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten,AveragePooling2D
from tensorflow.keras.applications import EfficientNetB4
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.callbacks import
EarlyStopping,ReduceLROnPlateau,ModelCheckpoint
from tqdm.keras import TqdmCallback
from tensorflow.keras.callbacks import ReduceLROnPlateau,
ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# For data pipeline building
from functools import partial
import keras
from keras import backend as K # for checking f1 score
from tensorflow import keras
from tensorflow.keras import layers, models
```

2. Mounting the dataset from Drive

```
train_folder = '/content/drive/MyDrive/HummingBirds/train'
val_folder = '/content/drive/MyDrive/HummingBirds/valid'
test_folder = '/content/drive/MyDrive/HummingBirds/test'
```

3. Data Preprocessing

```
# Data preprocessing
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,  # Normalize pixel values
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_valid_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

4. Load Datasets

```
# Load datasets
train_data = train_datagen.flow_from_directory(
    train_folder,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'  # For multi-class classification
)

valid_data = test_valid_datagen.flow_from_directory(
    val_folder,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

test_data = test_valid_datagen.flow_from_directory(
    test_folder,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

5. Building CNN model

```
# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128,
128, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(train_data.num_classes, activation='softmax')])
# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

6. Training the model

```
# Train the model
history = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=20,
    batch_size=32
)
```

7. Evaluating and Saving the model

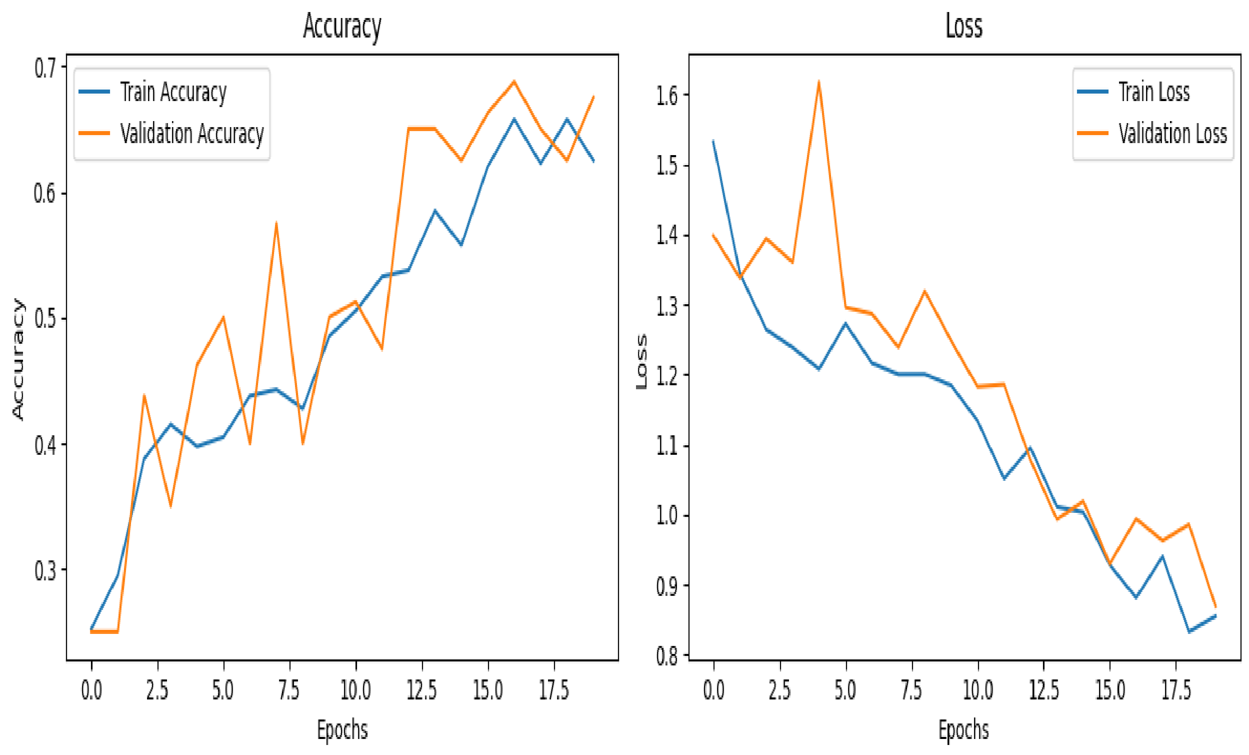
```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_data)
print(f"Test Accuracy: {test_accuracy:.2f}")
model.save("hummingbird_classifier.h5")
```

Output: ***Test Accuracy: 0.79***

8. Plotting the Accuracy and Loss

```
# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.legend()
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.tight_layout()
plt.show()
```



9. Loading the trained model and class names

```
# Load the trained model
model = tf.keras.models.load_model("hummingbird_classifier.h5")
# Load class names
class_indices = train_data.class_indices
class_names = {v: k for k, v in class_indices.items()}
```

10. Prediction on a random image

```
# Function to preprocess and predict
def predict_hummingbird_class(img_path):
    # Load and preprocess the image
    img = image.load_img(img_path, target_size=(128, 128))
    # Resize to match the input size of the model
    img_array = image.img_to_array(img)  # Convert to numpy array
    img_array = img_array / 255.0  # Normalize to [0, 1]
    img_array = np.expand_dims(img_array, axis=0)

    # Predict the class
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions)  # Get the index
of the highest probability
    predicted_class_name = class_names[predicted_class_index]
    print(f"Predicted Class: {predicted_class_name}
(Confidence:{predictions[0][predicted_class_index]:.2f})")
    return predicted_class_name

# Example usage
image_path = "bird2.jpeg"
plt.imshow(image.load_img(image_path))
plt.show()
predicted_class = predict_hummingbird_class(image_path)
```

OUTPUT:

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
Predicted Class: Broadtailed_female (Confidence: 0.40)
```

RESULT:

Thus the Hummingbird Classification implemented using CNN is built successfully and the output is verified.