import os

import sys

import re

import joblib

import pandas as pd

import numpy as np

import tensorflow as tf

import requests

import shutil

from flask import Flask, render\_template, request, redirect, url\_for

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image as keras\_image

from bs4 import BeautifulSoup

from werkzeug.utils import secure\_filename

app = Flask(\_name\_)

# Paths

MODEL\_PATH = "best\_model\_mobilenet.keras"

ENCODER\_PATH = "label\_encoder.pkl"

DATASET\_PATH = "dataset/train"

CSV\_PATH = "dataset/Training\_set.csv"

STATIC\_PATH = "static"

ALLOWED\_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

# Existence checks

for path in [MODEL\_PATH, ENCODER\_PATH, CSV\_PATH]:

if not os.path.exists(path):

sys.exit(f"Required file not found: {path}")

# Load model and encoder once

model = load\_model(MODEL\_PATH)

label\_encoder = joblib.load(ENCODER\_PATH)

df = pd.read\_csv(CSV\_PATH)

df['file\_path'] = df['filename'].apply(lambda x: os.path.join(DATASET\_PATH, x))

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

# Utility: Fetch info from Wikipedia

def fetch\_species\_info(species\_name):

try:

search\_term = species\_name.replace(" ", "") + "(butterfly)"

url = f"https://en.wikipedia.org/wiki/{search\_term}"

headers = {"User-Agent": "Mozilla/5.0 (compatible; ButterflyApp/1.0)"}

response = requests.get(url, timeout=5, headers=headers)

if response.status\_code == 404:

search\_term = species\_name.replace(" ", "\_")

url = f"https://en.wikipedia.org/wiki/{search\_term}"

response = requests.get(url, timeout=5, headers=headers)

if response.status\_code == 200:

soup = BeautifulSoup(response.content, "html.parser")

paragraphs = soup.find\_all("p")

for p in paragraphs:

text = p.get\_text().strip()

if len(text) > 100 and any(k in text.lower() for k in ["butterfly", "species", "family"]):

cleaned = re.sub(r"\[[^\]]\*\]", "", text)

return cleaned

return "No informative content found."

except Exception as e:

print(f"Error while fetching Wikipedia info: {e}")

return "No informative content found."

# Utility: Predict uploaded image

def predict\_species(image\_path):

try:

img = keras\_image.load\_img(image\_path, target\_size=(224, 224))

img\_array = keras\_image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0) / 255.0

prediction = model.predict(img\_array)

class\_index = np.argmax(prediction[0])

species = label\_encoder.inverse\_transform([class\_index])[0]

return species

except Exception as e:

print(f"Prediction error: {e}")

return "Unknown"

# Utility: Copy one image from dataset to static

def get\_image\_path(species\_name):

entry = df[df['label'].str.lower() == species\_name.lower()]

if not entry.empty:

dataset\_path = entry.iloc[0]['file\_path']

static\_path = os.path.join(STATIC\_PATH, os.path.basename(dataset\_path))

if not os.path.exists(static\_path):

try:

shutil.copy(dataset\_path, static\_path)

except Exception as e:

print(f"Failed to copy image: {e}")

return None

return os.path.basename(static\_path) # Return relative to /static

return None

@app.route('/')

def index():

return render\_template("index.html")

@app.route('/search', methods=['POST'])

def search():

species = request.form['species\_name']

image\_path = get\_image\_path(species)

description = fetch\_species\_info(species)

return render\_template("result.html", label=species, image=image\_path, description=description)

@app.route('/predict', methods=['POST'])

def predict():

file = request.files['image']

if file and allowed\_file(file.filename):

filename = secure\_filename(file.filename)

filepath = os.path.join(STATIC\_PATH, filename)

file.save(filepath)

predicted\_species = predict\_species(filepath)

image\_path = get\_image\_path(predicted\_species)

description = fetch\_species\_info(predicted\_species)

# Optionally remove the uploaded file after use

try:

os.remove(filepath)

except Exception as e:

print(f"Failed to remove uploaded file: {e}")

return render\_template("result.html", label=predicted\_species, image=image\_path, description=description)

return redirect(url\_for('index'))

if \_name\_ == '\_main\_':

app.run(debug=False)