import torch

import torchvision.transforms as transforms

from flask import Flask, jsonify, request

from PIL import Image

from torchvision import datasets

import torch.nn as nn

import gdown

from annoy import AnnoyIndex

from torchvision.models import mobilenet\_v3\_small, MobileNet\_V3\_Small\_Weights

import os

# Initialisation du dispositif

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

GENRES = ['action', 'comedy', 'animation', 'documentary', 'drama', 'fantasy', 'horror', 'romance', 'science Fiction', 'thriller']

# Chemin vers les poids du modèle

MODEL\_PATH = "model\_genre\_classifier1.pth"

# Chargement du modèle

GOOGLE\_DRIVE\_FILE\_ID = "1GfB\_aUBAudIyBmUrbSAcYGwXd2KnBr64"

if not os.path.exists(MODEL\_PATH):

    print("Model weights not found. Downloading from Google Drive...")

    gdown.download(f"https://drive.google.com/uc?id={GOOGLE\_DRIVE\_FILE\_ID}", MODEL\_PATH, quiet=False)

mobilenet = mobilenet\_v3\_small(weights=MobileNet\_V3\_Small\_Weights.IMAGENET1K\_V1)

model = nn.Sequential(

    mobilenet.features,

    mobilenet.avgpool,

    nn.Flatten(),

    nn.Linear(576, 256),

    nn.ReLU(),

    nn.Dropout(0.5),

    nn.Linear(256, len(GENRES))

)

model.load\_state\_dict(torch.load(MODEL\_PATH, map\_location=device))

model = model.to(device)

model.eval()

# Transformation des images

transform = transforms.Compose([

    transforms.Resize((224, 224)),

    transforms.ToTensor(),

    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Charger l'index Annoy

index = AnnoyIndex(256, 'angular') # 256 est la dimension des vecteurs extraits par le modèle

index.load('rec\_imdb.ann') # Charger votre index préexistant

# Initialisation de l'application Flask

app = Flask(\_\_name\_\_)

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

def predict():

    if 'file' not in request.files:

        return jsonify({"error": "No image file provided"}), 400

    file = request.files['file']

    if file.filename == '':

        return jsonify({"error": "No image selected"}), 400

    try:

        img\_pil = Image.open(file.stream)

    except Exception as e:

        return jsonify({"error": f"Unable to process image: {str(e)}"}), 400

    tensor = transform(img\_pil).to(device).unsqueeze(0)

    with torch.no\_grad():

        outputs = model(tensor)

        \_, predicted = torch.max(outputs, 1)

        predicted\_genre = GENRES[predicted.item()]

    return jsonify({"predicted\_genre": predicted\_genre})

@app.route('/')

def home():

    return '''

    <!doctype html>

    <title>Upload an Image</title>

    <h1>Upload an Image for Genre Prediction</h1>

    <form method="POST" action="/predict" enctype="multipart/form-data">

      <input type="file" name="file">

      <input type="submit" value="Upload">

    </form>

    '''

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

def recommend():

if 'file' not in request.files:

return jsonify({"error": "No image file provided"}), 400

file = request.files['file']

if file.filename == '':

return jsonify({"error": "No image selected"}), 400

try:

img\_pil = Image.open(file.stream)

except Exception as e:

return jsonify({"error": f"Unable to process image: {str(e)}"}), 400

# Transformation de l'image

tensor = transform(img\_pil).to(device).unsqueeze(0)

# Extraire le vecteur de caractéristiques

with torch.no\_grad():

features = model[0](tensor) # Utilisation de la partie convolutionnelle

features = features.flatten(start\_dim=1)

# Convertir les features en un vecteur 256 dimensions (ajuster en fonction de votre modèle)

feature\_vector = features.cpu().numpy().flatten()

# Trouver les 5 films les plus similaires

similar\_movie\_ids = index.get\_nns\_by\_vector(feature\_vector, 5)

# Vous pouvez renvoyer des informations supplémentaires comme les noms de films ici.

return jsonify({"similar\_movies": similar\_movie\_ids})

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host='0.0.0.0', port=5000, debug=True)

**Recommend\_web\_app.py:**

import gradio as gr

from PIL import Image

import requests

import io

# Fonction qui envoie l'image à l'API Flask et récupère les films similaires

def recommend\_movies(image):

# Convertir l'image de numpy array (format attendu par Gradio) en image PIL

image = Image.fromarray(image.astype('uint8'))

# Sauver l'image au format binaire dans un buffer

img\_binary = io.BytesIO()

image.save(img\_binary, format="PNG")

# Envoyer l'image en tant que données binaires à l'API Flask pour la recommandation

response = requests.post("http://127.0.0.1:5000/recommend", files={"file": img\_binary.getvalue()})

# Si la requête est réussie, extraire la liste des films similaires

if response.status\_code == 200:

similar\_movies = response.json().get("similar\_movies", "Error: No similar movies found")

return similar\_movies # Retourner les films similaires

# Si une erreur se produit dans la requête

return f"Error: {response.status\_code}"

# Interface Gradio

gr.Interface(

fn=recommend\_movies,

inputs="image", # Entrée sous forme d'image

outputs="json", # Sortie sous forme de JSON avec les films similaires

live=True,

description="Upload a movie poster to get 5 most similar movies based on their poster.",

).launch(server\_port=7860, debug=True, share=True)