import tkinter as tk  
from tkinter import filedialog, messagebox  
from tkinter import ttk  
import cv2  
from PIL import Image, ImageTk  
import numpy as np  
from tensorflow.keras.models import Sequential, load\_model  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
# Modeli eğitmek için kullanılan fonksiyon  
def train\_model():  
 try:  
 # Eğitim ve doğrulama verilerinin bulunduğu yolları belirtin  
 train\_dir = "dataset\_path/train" # Eğitim verilerinin bulunduğu yol  
 valid\_dir = "dataset\_path/invalid" # Doğrulama verilerinin bulunduğu yol  
  
 # Eğitim ve doğrulama verileri için veri artırma ve normalizasyon  
 train\_datagen = ImageDataGenerator(rescale=1.0/255.0) # Piksel değerlerini 0-1 arasına çeker  
 valid\_datagen = ImageDataGenerator(rescale=1.0/255.0) # Piksel değerlerini 0-1 arasına çeker  
  
 # Verileri yükleyin  
 train\_generator = train\_datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=(128, 128), # Tüm görüntüleri aynı boyuta getirir  
 batch\_size=32, # Her iterasyonda işlenecek görüntü sayısı  
 class\_mode='binary' # İkili sınıflandırma modu  
 )  
  
 valid\_generator = valid\_datagen.flow\_from\_directory(  
 valid\_dir,  
 target\_size=(128, 128),  
 batch\_size=32,  
 class\_mode='binary'  
 )  
  
 # Modelin tanımlanması  
 model = Sequential([  
 Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)), # Özellik çıkarımı için evrişim katmanı  
 MaxPooling2D(2, 2), # Maksimum havuzlama ile boyut küçültme  
 Conv2D(64, (3, 3), activation='relu'),  
 MaxPooling2D(2, 2),  
 Flatten(), # Veriyi düzleştirir, sınıflandırma katmanına hazırlar  
 Dense(128, activation='relu'), # Tam bağlı katman  
 Dense(1, activation='sigmoid') # İkili sınıflandırma için çıktı katmanı  
 ])  
  
 # Modelin derlenmesi  
 model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
 # Modeli eğitin  
 model.fit(  
 train\_generator,  
 epochs=10, # Eğitim için iterasyon sayısı  
 validation\_data=valid\_generator  
 )  
  
 # Eğitilen modeli kaydedin  
 model.save("corrosion\_detection\_model.keras") # Modeli ileride kullanmak için kaydedin  
 messagebox.showinfo("Model Training", "Model başarıyla eğitildi ve kaydedildi.")  
  
 except Exception as e:  
 messagebox.showerror("Training Error", f"Model eğitilirken bir hata oluştu: {e}")  
  
# Görüntü yükleme fonksiyonu (Adım 1)  
def load\_image():  
 file\_path = filedialog.askopenfilename(filetypes=[("Image Files", "\*.jpg \*.jpeg \*.png")]) # Kullanıcının görüntü seçmesini sağlar  
 if file\_path:  
 global img  
 img = cv2.imread(file\_path) # Seçilen görüntüyü OpenCV ile okur  
 img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # BGR formatını RGB'ye dönüştürür  
 img\_pil = Image.fromarray(img\_rgb) # Görüntüyü PIL formatına dönüştürür  
 img\_tk = ImageTk.PhotoImage(img\_pil) # Tkinter için görüntüyü hazırlar  
 panel\_original.config(image=img\_tk) # Orijinal görüntüyü kullanıcıya gösterir  
 panel\_original.image = img\_tk  
 process\_button.config(state=tk.NORMAL) # İşleme butonunu etkinleştirir  
  
# Görüntü ön işleme fonksiyonu (Adım 2)  
def preprocess\_image():  
 global preprocessed\_img  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Görüntüyü gri tonlamaya dönüştürür  
 blurred = cv2.GaussianBlur(gray, (5, 5), 0) # Gürültüyü azaltmak için bulanıklaştırır  
 preprocessed\_img = blurred  
 img\_pil = Image.fromarray(preprocessed\_img) # Ön işlenmiş görüntüyü PIL formatına dönüştürür  
 img\_tk = ImageTk.PhotoImage(img\_pil) # Tkinter için görüntüyü hazırlar  
 panel\_preprocessed.config(image=img\_tk) # Ön işlenmiş görüntüyü gösterir  
 panel\_preprocessed.image = img\_tk  
  
# Görüntü segmentasyon fonksiyonu (Adım 3)  
def segment\_image():  
 global segmented\_img  
 \_, thresholded = cv2.threshold(preprocessed\_img, 100, 255, cv2.THRESH\_BINARY\_INV) # Görüntüyü eşikleme yöntemiyle segmentlere ayırır  
 segmented\_img = thresholded  
 img\_pil = Image.fromarray(segmented\_img) # Segmentasyon sonucunu PIL formatına dönüştürür  
 img\_tk = ImageTk.PhotoImage(img\_pil) # Tkinter için görüntüyü hazırlar  
 panel\_segmented.config(image=img\_tk) # Segmentasyon sonucunu gösterir  
 panel\_segmented.image = img\_tk  
  
# Görüntü sınıflandırma fonksiyonu (Adım 5)  
def classify\_image():  
 try:  
 model = load\_model("corrosion\_detection\_model.keras") # Daha önce eğitilmiş modeli yükler  
 except Exception as e:  
 messagebox.showerror("Model Error", f"Model yüklenirken bir hata oluştu: {e}")  
 return  
  
 resized = cv2.resize(segmented\_img, (128, 128)) # Görüntüyü modelin giriş boyutlarına ayarlar  
 resized\_rgb = cv2.cvtColor(resized, cv2.COLOR\_GRAY2RGB) # Grayscale görüntüyü RGB formatına dönüştürür  
 resized\_rgb = resized\_rgb / 255.0 # Piksel değerlerini 0-1 arasına ölçekler  
 resized\_rgb = resized\_rgb.reshape(1, 128, 128, 3) # Modelin giriş formatına uygun hale getirir  
  
 prediction = model.predict(resized\_rgb) # Model ile tahmin yapar  
  
 if prediction[0] > 0.5:  
 result = "Korozyon Tespit Edildi"  
 # Segmentasyon sonucunu göster  
 img\_pil = Image.fromarray(segmented\_img)  
 img\_tk = ImageTk.PhotoImage(img\_pil.resize((400, 400))) # Görüntüyü daha iyi görünürlük için yeniden boyutlandırır  
 panel\_result.config(image=img\_tk)  
 panel\_result.image = img\_tk  
 else:  
 result = "Korozyon Tespit Edilmedi"  
  
 result\_label.config(text=f"Sonuç: {result}") # Sonucu kullanıcıya gösterir  
  
# GUI ayarları  
root = tk.Tk()  
root.title("Korozyon Tespit Uygulaması")  
  
frame = ttk.Frame(root, padding="10")  
frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))  
  
# Görüntü panelleri  
panel\_original = ttk.Label(frame, text="Orijinal Görüntü")  
panel\_original.grid(row=0, column=0, padx=5, pady=5)  
panel\_preprocessed = ttk.Label(frame, text="Ön İşlenmiş Görüntü")  
panel\_preprocessed.grid(row=0, column=1, padx=5, pady=5)  
panel\_segmented = ttk.Label(frame, text="Segmentasyon")  
panel\_segmented.grid(row=0, column=2, padx=5, pady=5)  
panel\_result = ttk.Label(frame, text="Sonuç Görüntüsü")  
panel\_result.grid(row=0, column=3, padx=5, pady=5)  
  
# Butonlar  
load\_button = ttk.Button(frame, text="Görüntü Yükle", command=load\_image)  
load\_button.grid(row=1, column=0, padx=5, pady=5)  
process\_button = ttk.Button(frame, text="İşle ve Tespit Et", command=lambda: [preprocess\_image(), segment\_image(), classify\_image()])  
process\_button.grid(row=1, column=1, padx=5, pady=5)  
process\_button.config(state=tk.DISABLED)  
  
train\_button = ttk.Button(frame, text="Modeli Eğit", command=train\_model)  
train\_button.grid(row=1, column=2, padx=5, pady=5)  
  
# Sonuç etiketi  
result\_label = ttk.Label(frame, text="Sonuç: Henüz bir işlem yapılmadı.", font=("Arial", 12))  
result\_label.grid(row=2, column=0, columnspan=4, padx=5, pady=5)  
  
root.mainloop()