train\_model.py

import os

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.applications.resnet50 import preprocess\_input

from tensorflow.keras.models import Model

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# ✅ Paths

train\_dir = 'archive/data/train'

val\_dir = 'archive/data/val'

# ✅ Parameters

IMG\_SIZE = (224, 224)

BATCH\_SIZE = 16

EPOCHS = 10 # Change to 5 for quick testing

# ✅ Data augmentation

train\_datagen = ImageDataGenerator(

preprocessing\_function=preprocess\_input,

rotation\_range=30,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True

)

val\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input)

train\_data = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=IMG\_SIZE,

batch\_size=BATCH\_SIZE,

class\_mode='categorical'

)

val\_data = val\_datagen.flow\_from\_directory(

val\_dir,

target\_size=IMG\_SIZE,

batch\_size=BATCH\_SIZE,

class\_mode='categorical'

)

# ✅ Load ResNet50 base model

base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze base model layers

for layer in base\_model.layers:

layer.trainable = False

# Unfreeze last few layers for fine-tuning

for layer in base\_model.layers[-10:]:

layer.trainable = True

# ✅ Custom classifier

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.5)(x)

output = Dense(4, activation='softmax')(x) # 4 classes

model = Model(inputs=base\_model.input, outputs=output)

# ✅ Compile the model

model.compile(

optimizer=Adam(learning\_rate=0.0001),

loss='categorical\_crossentropy',

metrics=['accuracy']

)

# ✅ Callbacks

checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', save\_best\_only=True, mode='max')

early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

# ✅ Train the model

history = model.fit(

train\_data,

validation\_data=val\_data,

epochs=EPOCHS,

callbacks=[checkpoint, early\_stop],

verbose=1

)

# ✅ Print final accuracy and loss

final\_train\_acc = history.history['accuracy'][-1]

final\_val\_acc = history.history['val\_accuracy'][-1]

final\_train\_loss = history.history['loss'][-1]

final\_val\_loss = history.history['val\_loss'][-1]

print(f"\nFinal Training Accuracy: {final\_train\_acc:.4f}")

print(f"Final Validation Accuracy: {final\_val\_acc:.4f}")

# ✅ Plot accuracy and loss

plt.figure(figsize=(12, 5))

# Accuracy

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()