import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

# Load dataset

# Assume data is organized in directories: 'data/train', 'data/validation', 'data/test'

train\_datagen = ImageDataGenerator(rescale=1./255, horizontal\_flip=True, rotation\_range=30)

validation\_datagen = ImageDataGenerator(rescale=1./255)

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

'data/train',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

validation\_generator = validation\_datagen.flow\_from\_directory(

'data/validation',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

# Load pre-trained ResNet50 model + higher level layers

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

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(1024, activation='relu')(x)

predictions = Dense(train\_generator.num\_classes, activation='softmax')(x)

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

# Freeze the layers of the base model

for layer in base\_model.layers:

layer.trainable = False

# Compile the model

model.compile(optimizer=Adam(lr=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(

train\_generator,

epochs=10,

validation\_data=validation\_generator

)

# Unfreeze some layers and fine-tune the model

for layer in base\_model.layers[-50:]:

layer.trainable = True

model.compile(optimizer=Adam(lr=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(

train\_generator,

epochs=10,

validation\_data=validation\_generator

)

# Save the model

model.save('food\_recognition\_model.h5')

# Calorie estimation example

food\_to\_calories = {'apple': 52, 'banana': 89, 'pizza': 266} # Example mapping

def estimate\_calories(predicted\_label):

return food\_to\_calories.get(predicted\_label, 0)

# Usage

# image = load\_image('path\_to\_image') # Implement image loading and preprocessing

# prediction = model.predict(image)

# predicted\_label = decode\_prediction(prediction) # Implement prediction decoding

# estimated\_calories = estimate\_calories(predicted\_label)

# print(f'Estimated Calories: {estimated\_calories}')