```
# program: Medical Image Segmentation
# purpose: Capstone project
# studies:MSc in Data Science and Artificial Intelligence in module Artificial Intelligence
Applications
# university: European Higher Education Institute, Malta
# author: Philip Kretz
# date: 12/1/2025

# Steps:

# 1. Define the U-Net model architecture.
# 2. Compile the model with a loss function and optimizer.
# 3. Generate placeholder data for inputs and segmentation masks.
# 4. Split the data into training and validation sets.
# 5. Train the model and visualize the accuracy over epochs.
# 6. Save the trained model for later use.

import tensorflow as tf
from tensorflow.keras.layers import Input, Dropout, Conv2D, MaxPooling2D, Conv2DTranspose,
concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# U-Net model: Function to define the architecture of a U-Net model
def unet_model(input_size=(128, 128, 1)):
    inputs = Input(input_size)

    # Encoding (Downsampling): Extract features while reducing spatial dimensions
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)  # First convolution block
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(c1)     # Second convolution
    p1 = MaxPooling2D((2, 2))(c1)                                     # Downsample spatial dimensions by 2x
    p1 = Dropout(0.1)(p1)  # Dropout to reduce overfitting

    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)    # Second convolution block
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)
    p2 = Dropout(0.1)(p2)  # Dropout to reduce overfitting

    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)    # Third convolution block
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)
    p3 = Dropout(0.2)(p3)  # Dropout to reduce overfitting

    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(p3)    # Fourth convolution block
    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
    p4 = MaxPooling2D((2, 2))(c4)
    p4 = Dropout(0.2)(p4)  # Dropout to reduce overfitting
    # Bottleneck: Deepest part of the network where features are most compressed
    c5 = Conv2D(1024, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(p4)
    c5 = Conv2D(1024, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(c5)
    c5 = Dropout(0.5)(c5)

    # Decoding (Upsampling)
    u6 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(u6)
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(c6)

    u7 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(u7)
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(c7)

    u8 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(u8)
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(c8)

    u9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(u9)
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=tf.keras.regularizers.l2(1e-4))(c9)

    # Output layer
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model

# Compile the model
model = unet_model()
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy',
metrics=['accuracy'])

# Simulated dataset
X = np.random.rand(100, 128, 128, 1)
Y = np.random.randint(0, 2, (100, 128, 128, 1))
```

```
# Data augmentation
datagen = ImageDataGenerator(rotation_range=20, width_shift_range=0.1, height_shift_range=0.1,
horizontal_flip=True)
datagen.fit(X)

# Split data
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=42)

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with data augmentation
history = model.fit(datagen.flow(X_train, Y_train, batch_size=16),
                    validation_data=(X_val, Y_val),
                    epochs=50, callbacks=[early_stopping])

# Plot training history
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.show()

# Save the model
model.save('unet_medical_segmentation.keras')
print("Model saved as 'unet_medical_segmentation.keras'.")
```