CODE:

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import os
import glob

# Define paths to datasets
testing_videos_path = '/Users/karthik/Desktop/GWU/Sem2/Computer Vision/project
4/Avenue Dataset/testing_videos'
training_videos_path = '/Users/karthik/Desktop/GWU/Sem2/Computer Vision/project
4/Avenue Dataset/training_videos'

# Function to extract frames from a video
def extract_frames(video_path, frame_count=200):
    """Extracts a specified number of frames from a video and resizes them to 224x224."""
    video = cv2.VideoCapture(video_path)
    if not video.isOpened():
        print(f"Failed to open video: {video_path}")
        return np.array([])
    frames = []
    frame_step = max(int(video.get(cv2.CAP_PROP_FRAME_COUNT) / frame_count), 1)
    current_frame = 0
    while len(frames) < frame_count:
        video.set(cv2.CAP_PROP_POS_FRAMES, current_frame)
        ret, frame = video.read()
        if not ret:
            print(f"Stopped reading frames at frame {current_frame} of video: {video_path}")
            break
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        resized_frame = cv2.resize(gray_frame, (224, 224))
        frames.append(resized_frame)
        current_frame += frame_step
    video.release()
    return np.array(frames)

# Function to recursively load all videos from a directory and its subdirectories
def load_data(directory):
    """Loads all videos from a directory and its subdirectories, extracting frames from each
video."""
    all_frames = []
    for video_file in glob.glob(os.path.join(directory, '*.avi')):
        frames = extract_frames(video_file)
        if frames.size == 0:
            print(f"No frames extracted from video: {video_file}")
        else:
            all_frames.append(frames)
    if not all_frames:
        print(f"No frames extracted from any videos in directory: {directory}")
        return np.array([])  # Return an empty array if no frames were extracted
    return np.vstack(all_frames)

# Build a more complex autoencoder model
def build_autoencoder(input_shape=(224, 224, 1)):
    """Builds a convolutional autoencoder with more layers and complexity."""
    input_layer = layers.Input(shape=input_shape)
    x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_layer)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

    # Decoder
    x = layers.Conv2DTranspose(8, (3, 3), strides=2, activation='relu',
padding='same')(encoded)
    x = layers.Conv2DTranspose(16, (3, 3), strides=2, activation='relu', padding='same')(x)
    x = layers.Conv2DTranspose(32, (3, 3), strides=2, activation='relu', padding='same')(x)
    decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

    autoencoder = models.Model(input_layer, decoded)
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
    return autoencoder

# Main Execution Block
if __name__ == "__main__":
    # Load data from training videos
    train_data = load_data(training_videos_path)
    if train_data.size == 0:
        print("Insufficient data for training. Check the dataset directories and contents.")
    else:
        train_data = train_data[..., np.newaxis] / 255.0  # Normalize the frames

        # Initialize and train the model
        autoencoder = build_autoencoder()
        autoencoder.fit(train_data, train_data, epochs=50, batch_size=20)

        # Save the model
        autoencoder.save('autoencoder_model.h5')

        # Load data from testing videos
        test_data = load_data(testing_videos_path)
        test_data = test_data[..., np.newaxis] / 255.0
        if test_data.size == 0:
            print("No testing data available.")
        else:
            # Evaluate on testing data
            predictions = autoencoder.predict(test_data)
            mse = np.mean(np.square(test_data - predictions), axis=(1, 2, 3))
            print("Anomaly scores for testing data:", mse)
```

OUTPUT:

```
Anomaly scores for testing data: [0.00327693 0.0033585  0.00343525 ... 0.00502272
0.00510027 0.00519131]
```