```
import os
import pandas as pd
import cv2  # You will need to install opencv-python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle

# Function to load and preprocess video data
def load_video_data(file_paths, image_size=(64, 64)):
    """Loads video data from file paths, resizes frames, and returns them as a numpy array."""
    frames = []
    for path in file_paths:
        cap = cv2.VideoCapture(path)
        # We'll take the first frame as a representative image for simplicity
        ret, frame = cap.read()
        cap.release()

        if ret:
            # Resize the frame and convert to grayscale
            resized_frame = cv2.resize(frame, image_size)
            gray_frame = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2GRAY)
            frames.append(gray_frame)

    return np.array(frames)

# 1. Load the data
try:
    df = pd.read_csv('/content/hand_gestures.csv')
except FileNotFoundError:
    print("Error: The file 'hand_gestures.csv' was not found. Please ensure it is in the same directory as the script.'
    exit()

# 2. Extract file paths and labels
# Assuming the file structure has columns for each gesture type
# The script will use the 'one' column for this example
gesture_paths = df['one'].tolist()
labels = ['one'] * len(gesture_paths)  # Create labels for the 'one' gesture

# 3. Preprocess the data
print("Loading and preprocessing video frames...")
# This step requires the video files to be accessible
# Make sure your paths in the CSV file are correct
# This is a hypothetical step and will only work if the video files exist
# X = load_video_data(gesture_paths)
# y = np.array(labels)

# Example: generate dummy data since we can't load the real videos
X = np.random.rand(len(gesture_paths), 64, 64)
y = np.array(labels)

# Normalize pixel values
X = X / 255.0

# Reshape data to fit the CNN input (add a channel dimension)
X = X.reshape(-1, 64, 64, 1)

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# 4. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# 5. Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
```

```
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(y_encoded)), activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Print a summary of the model architecture
model.summary()

# 6. Train the model
print("\nTraining the model...")
# Dummy data is used here, replace with real data for actual training
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# 7. Evaluate and save the model
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"\nTest accuracy: {test_accuracy}")

# Save the trained model
model.save('hand_gesture_model.h5')
print("Model saved as 'hand_gesture_model.h5'")
```

```
Loading and preprocessing video frames...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	320
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 128)	589,952
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129