```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import TimeDistributed, Conv1D, MaxPooling1D, Flatten, LSTM, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

```
#  Loading  raw sensor data (X) and labels (y)
samples = 1000
time_steps = 128
n_features = 9
n_classes = 6

# Simulated random data (Replace with your actual data loading)
X = np.random.rand(samples, time_steps, n_features)
y = np.random.randint(0, n_classes, samples)

# Normalize features
scaler = StandardScaler()
X_reshaped = X.reshape(-1, n_features)  # Flatten time and samples for scaling
X_scaled = scaler.fit_transform(X_reshaped)
X = X_scaled.reshape(samples, time_steps, n_features)

# Encode labels to categorical
y_cat = to_categorical(y, num_classes=n_classes)

# Split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y_cat, test_size=0.2, random_state=42)
```


```
samples = 1000
time_steps = 128
n_features = 9
n_classes = 6

# Simulated random data (Replace with your actual data loading)
X = np.random.rand(samples, time_steps, n_features)
y = np.random.randint(0, n_classes, samples)

# Normalize features
scaler = StandardScaler()
X_reshaped = X.reshape(-1, n_features)  # Flatten time and samples for scaling
X_scaled = scaler.fit_transform(X_reshaped)
X = X_scaled.reshape(samples, time_steps, n_features)

# Encode labels to categorical
y_cat = to_categorical(y, num_classes=n_classes)

# Split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y_cat, test_size=0.2, random_state=42)
```


```
def create_subsequences(X, n_steps=4, n_length=32):
    # X shape: (samples, time_steps, features)
    samples, time_steps, n_features = X.shape
    X_subseq = X.reshape((samples, n_steps, n_length, n_features))
    return X_subseq

n_steps = 4
n_length = 32

X_train_subseq = create_subsequences(X_train, n_steps, n_length)
X_test_subseq = create_subsequences(X_test, n_steps, n_length)
```

```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu'),
                          input_shape=(n_steps, n_length, n_features)))
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=3, activation='relu')))
```

```
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/wrapper.py:27: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistributed)	(None, 4, 30, 64)	1,792
time_distributed_1 (TimeDistributed)	(None, 4, 28, 64)	12,352
time_distributed_2 (TimeDistributed)	(None, 4, 14, 64)	0
time_distributed_3 (TimeDistributed)	(None, 4, 896)	0
lstm (LSTM)	(None, 100)	398,800
dense (Dense)	(None, 100)	10,100
dense_1 (Dense)	(None, 6)	606

Total params: 423,650 (1.62 MB)
Trainable params: 423,650 (1.62 MB)
Non-trainable params: 0 (0.00 B)

```
history = model.fit(X_train_subseq, y_train, epochs=20, batch_size=64,
                    validation_split=0.2, verbose=1)
```

```
Epoch 1/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 7s 165ms/step - accuracy: 0.1704 - loss: 1.8144 - val_accuracy: 0.1937 - val_loss: 1.7935
Epoch 2/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step - accuracy: 0.2195 - loss: 1.7673 - val_accuracy: 0.1813 - val_loss: 1.7967
Epoch 3/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.2442 - loss: 1.7502 - val_accuracy: 0.1937 - val_loss: 1.7947
Epoch 4/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.2629 - loss: 1.7124 - val_accuracy: 0.1625 - val_loss: 1.8189
Epoch 5/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.4895 - loss: 1.5994 - val_accuracy: 0.1187 - val_loss: 1.8733
Epoch 6/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 82ms/step - accuracy: 0.5144 - loss: 1.3888 - val_accuracy: 0.1625 - val_loss: 2.1359
Epoch 7/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.4823 - loss: 1.3408 - val_accuracy: 0.1875 - val_loss: 1.9815
Epoch 8/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - accuracy: 0.6834 - loss: 1.0480 - val_accuracy: 0.1750 - val_loss: 2.0885
Epoch 9/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.8638 - loss: 0.7102 - val_accuracy: 0.1688 - val_loss: 2.3935
Epoch 10/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.9166 - loss: 0.4767 - val_accuracy: 0.2188 - val_loss: 2.7239
Epoch 11/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - accuracy: 0.8934 - loss: 0.3856 - val_accuracy: 0.1750 - val_loss: 2.8610
Epoch 12/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.9705 - loss: 0.2141 - val_accuracy: 0.1688 - val_loss: 2.9465
Epoch 13/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 1.0000 - loss: 0.0890 - val_accuracy: 0.1750 - val_loss: 3.2382
Epoch 14/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 1.0000 - loss: 0.0593 - val_accuracy: 0.1562 - val_loss: 3.4472
Epoch 15/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 1.0000 - loss: 0.0272 - val_accuracy: 0.1437 - val_loss: 3.6983
Epoch 16/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 1.0000 - loss: 0.0172 - val_accuracy: 0.1500 - val_loss: 3.7578
Epoch 17/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 0.1437 - val_loss: 3.8622
Epoch 18/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 1.0000 - loss: 0.0079 - val_accuracy: 0.1437 - val_loss: 3.9334
Epoch 19/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 1.0000 - loss: 0.0063 - val_accuracy: 0.1500 - val_loss: 3.9971
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 1.0000 - loss: 0.0051 - val_accuracy: 0.1500 - val_loss: 4.0463
```

```
# Predict on test data
y_pred_probs = model.predict(X_test_subseq)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)

print("Classification Report:")
print(classification_report(y_true, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_true, y_pred))
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.25      0.20      0.22        41
           1       0.12      0.10      0.11        39
           2       0.10      0.13      0.11        31
           3       0.23      0.30      0.26        23
           4       0.10      0.09      0.09        35
           5       0.30      0.32      0.31        31

    accuracy                           0.18       200
   macro avg       0.18      0.19      0.18       200
weighted avg       0.18      0.18      0.18       200

Confusion Matrix:
[[ 8  8 13  3  4  5]
 [ 5  4 10  5  7  8]
 [ 4  3  4  4 12  4]
 [ 4  7  2  7  1  2]
 [ 7  6  9  6  3  4]
 [ 4  4  3  6  4 10]]
```

```
model.save('har_cnn_lstm_model.h5')

# To load model later
# from tensorflow.keras.models import load_model
# model = load_model('har_cnn_lstm_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.