```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# --- 1. Load metadata ---
csv_path = '/content/drive/MyDrive/DV project /Simplified/CombinedCSV/standing_metadata.csv'
df = pd.read_csv(csv_path)

# Filter for GestureLabel = 0
gesture0_df = df[df['GestureLabel'] == 0].reset_index(drop=True)

# --- 2. Load skeleton ---
def load_skeleton(file_path):
    """
    Load skeleton data from txt file.
    Shape: (n_frames, n_joints, 3)
    """
    data = np.loadtxt(file_path, delimiter=',')
    n_joints = data.shape[1] // 3
    data = data.reshape((-1, n_joints, 3))
    return data

# --- 3. Feature extraction ---
def extract_features(data):
    """
    Extract mean and std of each joint coordinate across frames.
    Returns a 1D array of features.
    """
    features = []
    features.extend(np.mean(data, axis=0).flatten())  # mean x,y,z for each joint
    features.extend(np.std(data, axis=0).flatten())   # std x,y,z for each joint
    return np.array(features)

# --- 4. Prepare dataset ---
X = []
y = []

for idx, row in gesture0_df.iterrows():
    file_path = '/content/drive/MyDrive/DV project /Simplified/Separated_Files_By_Position/standing/' + row['Filename']
    if not os.path.exists(file_path):
        continue
    data = load_skeleton(file_path)
    features = extract_features(data)
    X.append(features)
    y.append(1 if row['CorrectLabel'] == 1 else 0)

X = np.array(X)
y = np.array(y)

print("Feature shape:", X.shape)
print("Labels shape:", y.shape)
print("Correct labels distribution:", np.bincount(y))

# --- 5. Train/Test split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# --- 6. Random Forest classifier ---
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# --- 7. Predictions ---
y_pred = clf.predict(X_test)

# --- 8. Evaluation ---
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Feature shape: (131, 150)
Labels shape: (131,)
Correct labels distribution: [34 97]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        20
```

```
    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27

Confusion Matrix:
 [[ 7  0]
 [ 0 20]]
```

Start coding or generate with AI.

```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# --- 1. Load metadata ---
csv_path = '/content/drive/MyDrive/DV project /Simplified/CombinedCSV/standing_metadata.csv'
df = pd.read_csv(csv_path)

# --- 2. Load skeleton function ---
def load_skeleton(file_path):
    """
    Load skeleton data from txt file.
    Shape: (n_frames, n_joints, 3)
    """
    data = np.loadtxt(file_path, delimiter=',')
    n_joints = data.shape[1] // 3
    data = data.reshape((-1, n_joints, 3))
    return data

# --- 3. Feature extraction ---
def extract_features(data):
    """
    Extract mean and std of each joint coordinate across frames.
    Returns a 1D array of features.
    """
    features = []
    features.extend(np.mean(data, axis=0).flatten())  # mean x,y,z for each joint
    features.extend(np.std(data, axis=0).flatten())   # std x,y,z for each joint
    return np.array(features)

# --- 4. Function to prepare data for a specific gesture ---
def prepare_gesture_data(gesture_label):
    gesture_df = df[df['GestureLabel'] == gesture_label].reset_index(drop=True)
    X, y = [], []
    for idx, row in gesture_df.iterrows():
        file_path = '/content/drive/MyDrive/DV project /Simplified/Separated_Files_By_Position/standing/' + row['Filename']
        if not os.path.exists(file_path):
            continue
        data = load_skeleton(file_path)
        features = extract_features(data)
        X.append(features)
        y.append(1 if row['CorrectLabel'] == 1 else 0)
    return np.array(X), np.array(y)

# --- 5. Train Random Forest for each gesture ---
results = {}
for gesture_label in range(0, 9):  # Gestures 0 to 8
    print(f"\n=== GestureLabel {gesture_label} ===")
    X, y = prepare_gesture_data(gesture_label)
    if len(y) == 0:
        print("No data found for this gesture.")
        continue

    print("Feature shape:", X.shape)
    print("Labels distribution:", np.bincount(y))

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
    results[gesture_label] = {
        'model': clf,
        'accuracy': acc,
        'y_test': y_test,
        'y_pred': y_pred
    }

print("\nAll gestures processed. Models stored in 'results' dictionary.")
```

```
=== GestureLabel 0 ===
Feature shape: (131, 150)
Labels distribution: [34 97]
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        20

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27

Confusion Matrix:
 [[ 7  0]
 [ 0 20]]

=== GestureLabel 1 ===
Feature shape: (139, 150)
Labels distribution: [ 25 114]
Accuracy: 0.9285714285714286
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.60      0.75         5
           1       0.92      1.00      0.96        23

    accuracy                           0.93        28
   macro avg       0.96      0.80      0.85        28
weighted avg       0.93      0.93      0.92        28

Confusion Matrix:
 [[ 3  2]
 [ 0 23]]

=== GestureLabel 2 ===
Feature shape: (116, 150)
Labels distribution: [ 15 101]
Accuracy: 0.9166666666666666
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       0.95      0.95      0.95        21

    accuracy                           0.92        24
   macro avg       0.81      0.81      0.81        24
weighted avg       0.92      0.92      0.92        24

Confusion Matrix:
 [[ 2  1]
 [ 1 20]]

=== GestureLabel 3 ===
Feature shape: (127, 150)
```

```
# stgcn_pipeline.py  (run in Colab / local env)
import os
import math
import numpy as np
import pandas as pd
from glob import glob
from tqdm import tqdm

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# ----------------------------
# USER PARAMETERS - edit here
# ----------------------------
```

```
"
csv_path = '/content/drive/MyDrive/DV project /Simplified/CombinedCSV/standing_metadata.csv'
data_folder = '/content/drive/MyDrive/DV project /Simplified/Separated_Files_By_Position/standing'
gesture_label = 0         # which gesture to train on (0..8). Set to None to use all gestures.
max_len = 150             # pad/truncate frames to this length
batch_size = 16
num_epochs = 30
lr = 1e-3
weight_decay = 1e-4
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
save_model_path = 'stgcn_gesture{}_best.pth'.format(gesture_label if gesture_label is not None else 'all')
# ----------------------------

# ----------------------------
# Graph definition (V joints)
# ----------------------------
# Use your skeleton connections (0-indexed). Adjust if your dataset joint ordering differs.
connections = [
    (0,1),(1,2),(2,3),      # Spine
    (2,4),(4,5),(5,6),(6,7),# Left arm
    (2,8),(8,9),(9,10),(10,11), # Right arm
    (0,12),(12,13),(13,14),(14,15), # Left leg
    (0,16),(16,17),(17,18),(18,19)  # Right leg
]
# We'll infer V from loading a sample file later.

# ----------------------------
# Utilities
# ----------------------------
def normalize_adjacency(A):
    # symmetric normalization D^-1/2 A D^-1/2
    D = np.sum(A, axis=1)
    D_inv_sqrt = np.diag(1.0 / (np.sqrt(D) + 1e-6))
    return D_inv_sqrt @ A @ D_inv_sqrt

def build_adjacency_matrix(V, connections):
    A = np.zeros((V, V), dtype=np.float32)
    for i,j in connections:
        if i < V and j < V:
            A[i,j] = 1
            A[j,i] = 1
    # self connections
    for v in range(V):
        A[v,v] = 1.0
    A = normalize_adjacency(A)
    return A

# ----------------------------
# Dataset
# ----------------------------
class SkeletonDataset(Dataset):
    def __init__(self, df, data_folder, gesture_label=None, max_len=150):
        self.rows = df if gesture_label is None else df[df['GestureLabel'] == gesture_label]
        self.rows = self.rows.reset_index(drop=True)
        self.data_folder = data_folder
        self.max_len = max_len
        # pre-check one file to get V
        sample_path = None
        for i, r in self.rows.iterrows():
            p = os.path.join(self.data_folder, r['Filename'])
            if os.path.exists(p):
                sample_path = p
                break
        if sample_path is None:
            raise FileNotFoundError("No sample files found in dataset folder")
        sample = np.loadtxt(sample_path, delimiter=',')
        self.V = sample.shape[1] // 3

    def __len__(self):
        return len(self.rows)

    def load_file(self, path):
        data = np.loadtxt(path, delimiter=',')  # (T, V*3)
        T = data.shape[0]
        V = data.shape[1] // 3
        data = data.reshape((T, V, 3))  # (T, V, 3)
        return data.astype(np.float32)

    def pad_truncate(self, x):
        # x: (T, V, C=3) -> desired shape (max_len, V, 3)
        T = x.shape[0]
        if T >= self.max_len:
            return x[:self.max_len]
```

```
            else:
                pad = np.zeros((self.max_len - T, x.shape[1], x.shape[2]), dtype=x.dtype)
                return np.concatenate([x, pad], axis=0)

        def __getitem__(self, idx):
            row = self.rows.loc[idx]
            file_path = os.path.join(self.data_folder, row['Filename'])
            if not os.path.exists(file_path):
                raise FileNotFoundError(file_path)
            x = self.load_file(file_path)  # (T, V, 3)
            x = self.pad_truncate(x)       # (max_len, V, 3)
            # transpose to (C, T, V)
            x = np.transpose(x, (2, 0, 1)).copy()  # (3, T, V)
            y = 1 if row['CorrectLabel'] == 1 else 0
            return torch.tensor(x, dtype=torch.float32), torch.tensor(y, dtype=torch.long)

    def collate_fn(batch):
        xs, ys = zip(*batch)
        xs = torch.stack(xs, dim=0)  # (N, C, T, V)
        ys = torch.stack(ys).long()
        return xs, ys

    # ----------------------------
    # Minimal ST-GCN building blocks
    # ----------------------------
    class GraphConv(nn.Module):
        def __init__(self, in_channels, out_channels, A):
            super().__init__()
            # A is numpy adjacency (V,V) -> convert to torch
            self.register_buffer('A', torch.tensor(A, dtype=torch.float32))
            self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)
        def forward(self, x):
            # x: (N, C, T, V)
            # apply conv over channels then multiply by adjacency
            # conv expects (N, C, T, V) -> conv over (C -> outC) with 1x1
            x = self.conv(x)  # (N, outC, T, V)
            # multiply in joint dimension: einsum
            A = self.A  # (V,V)
            x = torch.einsum('nctv,vw->nctw', x, A)
            return x

    class TemporalConv(nn.Module):
        def __init__(self, in_channels, out_channels, kernel_size=9, stride=1):
            super().__init__()
            padding = (kernel_size - 1) // 2
            self.tconv = nn.Conv2d(in_channels, out_channels, kernel_size=(kernel_size,1),
                                   padding=(padding,0), stride=(stride,1))
            self.bn = nn.BatchNorm2d(out_channels)
            self.relu = nn.ReLU()
        def forward(self, x):
            # x: (N, C, T, V) treat time as height
            x = self.tconv(x)
            x = self.bn(x)
            x = self.relu(x)
            return x

    class STBlock(nn.Module):
        def __init__(self, in_channels, out_channels, A):
            super().__init__()
            self.gcn = GraphConv(in_channels, out_channels, A)
            self.tcn = TemporalConv(out_channels, out_channels)
            self.residual = nn.Identity() if in_channels == out_channels else nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1),
                nn.BatchNorm2d(out_channels)
            )
            self.relu = nn.ReLU()
        def forward(self, x):
            # x: (N, C, T, V)
            y = self.gcn(x)
            y = self.tcn(y)
            res = self.residual(x)
            return self.relu(y + res)

    # ----------------------------
    # ST-GCN Model
    # ----------------------------
    class STGCN(nn.Module):
        def __init__(self, in_channels, A, num_class=2, layers_channels=[64,64,128,256]):
            super().__init__()
            self.register_buffer('A', torch.tensor(A, dtype=torch.float32))
            layers = []
            c = in_channels
            for ch in layers_channels:
```

```
            layers.append(STBlock(c, ch, A))
            c = ch
        self.st_blocks = nn.Sequential(*layers)
        self.pool = nn.AdaptiveAvgPool2d((1,1))  # average over time and joints
        self.fc = nn.Linear(c, num_class)
    def forward(self, x):
        # x: (N, C, T, V)
        x = self.st_blocks(x)
        x = self.pool(x)  # (N, C, 1, 1)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

# ----------------------------
# Training / evaluation helpers
# ----------------------------
def train_epoch(model, loader, optimizer, criterion, device):
    model.train()
    losses = []
    preds = []
    trues = []
    for x,y in loader:
        x = x.to(device)    # (N,C,T,V)
        y = y.to(device)
        out = model(x)
        loss = criterion(out, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
        preds += out.argmax(dim=1).detach().cpu().tolist()
        trues += y.detach().cpu().tolist()
    return np.mean(losses), accuracy_score(trues, preds)

def eval_epoch(model, loader, criterion, device):
    model.eval()
    losses = []
    preds = []
    trues = []
    with torch.no_grad():
        for x,y in loader:
            x = x.to(device)
            y = y.to(device)
            out = model(x)
            loss = criterion(out, y)
            losses.append(loss.item())
            preds += out.argmax(dim=1).detach().cpu().tolist()
            trues += y.detach().cpu().tolist()
    return np.mean(losses), accuracy_score(trues, preds), trues, preds

# ----------------------------
# Main: prepare data, model, train
# ----------------------------
def main():
    # load metadata
    df = pd.read_csv(csv_path)
    if gesture_label is not None:
        df_use = df[df['GestureLabel'] == gesture_label].reset_index(drop=True)
    else:
        df_use = df.copy().reset_index(drop=True)
    print("Total records used:", len(df_use))

    # dataset and dataloaders
    ds = SkeletonDataset(df_use, data_folder, gesture_label=gesture_label, max_len=max_len)
    V = ds.V
    print("Detected joints V =", V)
    # ensure connections are compatible - build adjacency based on detected V
    A = build_adjacency_matrix(V, connections)

    # split train/test stratified
    labels = [1 if r['CorrectLabel'] == 1 else 0 for _, r in ds.rows.iterrows()]
    labels = np.array(labels)
    # simple split: 80/20 stratified
    from sklearn.model_selection import train_test_split
    idx_all = np.arange(len(ds))
    train_idx, test_idx = train_test_split(idx_all, test_size=0.2, random_state=42, stratify=labels)
    train_rows = ds.rows.loc[train_idx].reset_index(drop=True)
    test_rows = ds.rows.loc[test_idx].reset_index(drop=True)

    train_ds = SkeletonDataset(train_rows, data_folder, gesture_label=None, max_len=max_len)
    test_ds  = SkeletonDataset(test_rows, data_folder, gesture_label=None, max_len=max_len)

    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, collate_fn=collate_fn)
```

```
    test_loader  = DataLoader(test_ds,  batch_size=batch_size, shuffle=False, collate_fn=collate_fn)

    # model
    model = STGCN(in_channels=3, A=A, num_class=2).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)

    best_acc = 0.0
    for epoch in range(1, num_epochs+1):
        train_loss, train_acc = train_epoch(model, train_loader, optimizer, criterion, device)
        val_loss, val_acc, y_true, y_pred = eval_epoch(model, test_loader, criterion, device)
        scheduler.step()
        print(f"Epoch {epoch:02d} | Train loss {train_loss:.4f} acc {train_acc:.4f} | Val loss {val_loss:.4f} acc {val_acc:.
        if val_acc > best_acc:
            best_acc = val_acc
            torch.save({
                'model_state': model.state_dict(),
                'optimizer_state': optimizer.state_dict(),
                'epoch': epoch,
                'val_acc': val_acc
            }, save_model_path)
            print("Saved best model.", save_model_path)

    # final evaluation
    print("\n=== Final evaluation on test set ===")
    _, acc, y_true, y_pred = eval_epoch(model, test_loader, criterion, device)
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_true, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

if __name__ == '__main__':
    main()
```

```
Total records used: 131
Detected joints V = 25
Epoch 01 | Train loss 0.4965 acc 0.7115 | Val loss 0.8369 acc 0.7407
Saved best model. stgcn_gesture0_best.pth
Epoch 02 | Train loss 0.3036 acc 0.8750 | Val loss 1.0495 acc 0.7407
Epoch 03 | Train loss 0.2968 acc 0.8365 | Val loss 1.1705 acc 0.7407
Epoch 04 | Train loss 0.2266 acc 0.9135 | Val loss 0.2498 acc 0.8889
Saved best model. stgcn_gesture0_best.pth
Epoch 05 | Train loss 0.2501 acc 0.9423 | Val loss 0.2270 acc 0.9259
Saved best model. stgcn_gesture0_best.pth
Epoch 06 | Train loss 0.2275 acc 0.9135 | Val loss 0.7763 acc 0.7778
Epoch 07 | Train loss 0.2303 acc 0.9519 | Val loss 0.1554 acc 0.8889
Epoch 08 | Train loss 0.2389 acc 0.8942 | Val loss 1.4048 acc 0.7407
Epoch 09 | Train loss 0.1923 acc 0.9423 | Val loss 0.1783 acc 0.9259
Epoch 10 | Train loss 0.1879 acc 0.9327 | Val loss 0.1580 acc 0.9259
Epoch 11 | Train loss 0.1499 acc 0.9712 | Val loss 0.1304 acc 0.9259
Epoch 12 | Train loss 0.1753 acc 0.9327 | Val loss 0.1216 acc 0.9259
Epoch 13 | Train loss 0.1604 acc 0.9519 | Val loss 0.1040 acc 0.9259
Epoch 14 | Train loss 0.1472 acc 0.9712 | Val loss 0.1055 acc 0.9259
Epoch 15 | Train loss 0.1423 acc 0.9615 | Val loss 0.1013 acc 0.9630
Saved best model. stgcn_gesture0_best.pth
Epoch 16 | Train loss 0.1363 acc 0.9712 | Val loss 0.1361 acc 0.9630
Epoch 17 | Train loss 0.0994 acc 0.9808 | Val loss 0.1153 acc 0.9630
Epoch 18 | Train loss 0.1229 acc 0.9615 | Val loss 0.0855 acc 1.0000
Saved best model. stgcn_gesture0_best.pth
Epoch 19 | Train loss 0.1115 acc 0.9808 | Val loss 0.0897 acc 1.0000
Epoch 20 | Train loss 0.1028 acc 0.9712 | Val loss 0.0911 acc 0.9630
Epoch 21 | Train loss 0.1148 acc 0.9712 | Val loss 0.1002 acc 0.9630
Epoch 22 | Train loss 0.1197 acc 0.9712 | Val loss 0.0835 acc 0.9630
Epoch 23 | Train loss 0.1213 acc 0.9808 | Val loss 0.0845 acc 0.9630
Epoch 24 | Train loss 0.1307 acc 0.9615 | Val loss 0.1205 acc 0.9630
Epoch 25 | Train loss 0.1891 acc 0.9615 | Val loss 0.1459 acc 1.0000
Epoch 26 | Train loss 0.1049 acc 0.9808 | Val loss 0.0804 acc 1.0000
Epoch 27 | Train loss 0.1611 acc 0.9615 | Val loss 0.0952 acc 0.9259
Epoch 28 | Train loss 0.1199 acc 0.9808 | Val loss 0.1370 acc 0.9259
Epoch 29 | Train loss 0.1220 acc 0.9808 | Val loss 0.1036 acc 0.9259
Epoch 30 | Train loss 0.1026 acc 0.9615 | Val loss 0.0694 acc 1.0000

=== Final evaluation on test set ===
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        20

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27

Confusion Matrix:
```

```
[[ 7  0]
 [ 0 20]]
```

Start coding or generate with AI.