Yoga Pose Classification Model

Overview

This project implements a machine learning pipeline to classify yoga poses from images. It uses the MoveNet pose estimation model to extract human body keypoints from images, preprocesses these keypoints, and then trains a neural network to classify the pose. The project is structured for extensibility and clarity, with modular code for data handling, preprocessing, pose estimation, and model training.

Directory and File Structure

```
classification model/
│
├── data.py
├── training.py
├── proprocessing.py
├── movenet.py
├── movenet_thunder.tflite
├── weights.best.hdf5
├── train_data.csv
├── test_data.csv
│
├── model/
│   ├── model.json
│   └── group1-shard1of1.bin
│
├── csv_per_pose/
│   ├── chair.csv
│   ├── cobra.csv
│   ├── dog.csv
│   ├── no_pose.csv
│   ├── shoudler_stand.csv
│   ├── traingle.csv
│   ├── tree.csv
│   └── warrior.csv
│
└── yoga_poses/
    ├── train/
    │   ├── chair/
    │   ├── cobra/
    │   ├── dog/
    │   ├── no_pose/
    │   ├── shoudler_stand/
    │   ├── traingle/
    │   ├── tree/
    │   └── warrior/
    └── test/
        ├── chair/
        ├── cobra/
        ├── dog/
        ├── no_pose/
        ├── shoudler_stand/
        ├── traingle/
```

```
├── tree/
└── warrior/
```

Detailed Explanation of Each File

1. `data.py`

- **Purpose:**
  Provides data structures and utility functions for pose estimation, adapted from the TensorFlow example repository.
- **Key Components:**
  - **BodyPart (Enum):** Enumerates the 17 keypoints detected by pose estimation models (e.g., NOSE, LEFT_EYE, RIGHT_SHOULDER, etc.).
  - **Point, Rectangle, KeyPoint, Person (NamedTuple):** Data structures to represent 2D points, bounding boxes, keypoints, and detected persons.
  - **person_from_keypoints_with_scores:** Converts raw keypoint output from MoveNet into a `Person` object, computes bounding boxes, and averages keypoint scores.
  - **Category (NamedTuple):** Represents a classification result (label and score).

2. `training.py`

- **Purpose:**
  Loads preprocessed data, defines the neural network architecture, trains the model, evaluates it, and exports it for deployment.
- **Key Components:**
  - **load_csv:** Loads a CSV file, drops the filename, extracts class labels, and prepares features and one-hot encoded targets.
  - **get_center_point, get_pose_size, normalize_pose_landmarks:** Functions for normalizing pose keypoints, centering, and scaling them for model input.
  - **landmarks_to_embedding:** Converts 17 keypoints (x, y, score) into a flattened embedding vector.
  - **preprocess_data:** Applies embedding transformation to all samples.
  - **Model Definition:**
    - Input: 34-dimensional vector (flattened, normalized keypoints).
    - Hidden Layers: Dense (128, relu6) → Dropout (0.5) → Dense (64, relu6) → Dropout (0.5).
    - Output: Dense (number of classes, softmax).
  - **Training Loop:**
    - Uses early stopping and model checkpointing.
    - Trains for up to 200 epochs, batch size 16.
    - Evaluates on test data and prints loss/accuracy.
    - Exports the model to TensorFlow.js format (`model/` directory).

3. `proprocessing.py`

- **Purpose:**
  Preprocesses images to extract pose keypoints using MoveNet, saves per-pose CSVs, and combines them into training/testing datasets.
- **Key Components:**
  - **Movenet Model Download:** Downloads the MoveNet Thunder TFLite model if not present.
  - **detect:** Runs pose detection on an image tensor, with multiple inference passes for stability.
  - **Preprocessor Class:**
    - **init:** Sets up input/output paths, creates output directories, and lists pose classes.
    - **process:** For each pose class, iterates over images, runs pose detection, filters by keypoint confidence, and writes keypoints to CSV. Skips invalid or low-confidence images.
    - **all_landmarks_as_dataframe:** Merges all per-class CSVs into a single DataFrame, adds class labels, and writes to a combined CSV.
  - **Script Execution:**
    - Runs preprocessing for both training and testing images, generating `train_data.csv` and `test_data.csv`.

4. `movenet.py`

- **Purpose:**
  Implements the MoveNet pose estimation model wrapper, handling model loading, inference, and smart cropping.
- **Key Components:**
  - **Movenet Class:**
    - **init:** Loads the TFLite model, sets up input/output details.
    - **init_crop_region:** Computes a default crop region to ensure the person is centered and scaled.
    - **_torso_visible, _determine_torso_and_body_range, _determine_crop_region:** Helper methods for smart cropping based on detected keypoints.
    - **_crop_and_resize, _run_detector:** Internal methods for image preprocessing and running inference.
    - **detect:** Main method to run pose estimation on an input image, returning a `Person` object with keypoints and scores.

5. `movenet_thunder.tflite`

- **Purpose:**
  The pre-trained MoveNet Thunder model in TensorFlow Lite format, used for fast and accurate pose estimation.

6. `weights.best.hdf5`

- **Purpose:**
  Stores the best weights of the trained classification model, used for inference or further training.

7. `train_data.csv` / `test_data.csv`

- **Purpose:**
  Combined CSVs containing all keypoints and labels for training and testing, generated by the preprocessing pipeline.

8. `csv_per_pose/`

- **Purpose:**
  Contains per-pose CSV files, each storing keypoints for all images of a specific pose class (e.g., `chair.csv`, `cobra.csv`, etc.).

9. `model/`

- **Purpose:**
  Stores the exported trained model in TensorFlow.js format for deployment in web applications.
  - **model.json:** Model architecture and metadata.
  - **group1-shard1of1.bin:** Model weights.

10. `yoga_poses/`

- **Purpose:**
  Contains all raw images, organized by pose and split into `train/` and `test/` sets. Each subfolder corresponds to a yoga pose class.

Data Flow and Pipeline

1. **Image Collection:**
   - Images are stored in `yoga_poses/train/` and `yoga_poses/test/`, organized by pose class.
2. **Pose Estimation (Preprocessing):**
   - `proprocessing.py` uses `movenet.py` and `movenet_thunder.tflite` to extract 17 keypoints (x, y, score) from each image.
   - Keypoints are saved in per-pose CSVs (`csv_per_pose/`) and then merged into `train_data.csv` and `test_data.csv`.
3. **Data Normalization and Embedding:**
   - `training.py` normalizes keypoints (centering, scaling) and flattens them into 34-dimensional vectors for model input.
4. **Model Training:**

    - A neural network is trained on the processed data, with early stopping and checkpointing.
    - The best model is saved as `weights.best.hdf5` and exported to TensorFlow.js format.

5. **Evaluation and Deployment:**

    - The model is evaluated on test data.
    - The exported model can be used in web applications for real-time yoga pose classification.

How to Use

1. Prepare Data

- Place your images in the appropriate folders under `yoga_poses/train/` and `yoga_poses/test/`, organized by pose class.

2. Preprocess Data

- Run `proprocessing.py` to extract keypoints and generate `train_data.csv` and `test_data.csv`.

3. Train the Model

- Run `training.py` to train the classifier. The best weights will be saved, and the model will be exported for web use.

4. Inference

- Use the trained model (`weights.best.hdf5` or the TensorFlow.js model) to classify new images.

Major Code Blocks and Their Roles

- **Keypoint Extraction:**

    - `movenet.py` and `proprocessing.py` work together to extract and store pose keypoints from images.

- **Data Normalization:**

    - `training.py` normalizes and embeds keypoints for robust model training.

- **Model Architecture:**

    - Defined in `training.py` as a simple feedforward neural network with dropout for regularization.

- **Training and Evaluation:**

    - `training.py` handles the full training loop, validation, checkpointing, and evaluation.

- **Export and Deployment:**
    - Model is exported to TensorFlow.js format for use in web applications.

Notes

- The code is modular and can be extended to support more poses or different model architectures.
- The MoveNet model is efficient and suitable for real-time applications.
- The preprocessing pipeline ensures only high-confidence keypoints are used for training, improving model accuracy.

Credits

- Some modules (notably `data.py` and `movenet.py`) are adapted from the official TensorFlow examples.

If you need further breakdowns (e.g., function-by-function documentation or code comments), let me know!