1. Importing Libraries

Standard libraries from TensorFlow/Keras, Matplotlib, Seaborn, and Scikit-learn are imported. These are used for building, training, evaluating, and visualizing the model.

2. Setting Parameters

Paths for the train and test datasets are set. Image dimensions are standardized to 224x224 pixels, and a batch size of 32 is used.

3. Image Preprocessing & Augmentation

Training images are augmented using techniques like rotation, zoom, shifts, brightness changes, shearing, and flipping. This improves the model's robustness and helps prevent overfitting.

Test images are only rescaled (normalized) to ensure that evaluation is done on clean, unaltered images.

4. Creating Data Generators

ImageDataGenerator is used to read and process images from folders. It automatically labels the images based on their folder names.

5. Building the CNN Model

The CNN consists of:

- A stack of 3 convolutional blocks, each with:
- Convolutional layer
- Batch normalization
- Max pooling
- A flattening layer
- A dense layer with 256 neurons
- Dropout for regularization

A final softmax output layer with 4 neurons (one per posture class)

This model is designed to be lightweight yet effective for small to medium datasets.

6. Compiling the Model

The model is compiled using:

- Adam optimizer with a learning rate of 0.0001
- Categorical cross-entropy loss, suitable for multi-class classification
- Accuracy metric to monitor performance

7. Training the Model

Training is done for up to 100 epochs, but with smart interruption using callbacks:

- EarlyStopping halts training when validation loss stops improving for 10 epochs.
- ModelCheckpoint saves the best model based on validation accuracy.
- ReduceLROnPlateau decreases the learning rate when progress slows.
- This setup ensures the model doesn't overfit and converges effectively.

8. Plotting Training Progress

Two plots are created:

- Accuracy plot showing training and validation accuracy per epoch.
- Loss plot showing training and validation loss per epoch.
- These plots help visualize how well the model learns over time.

9. Evaluating Model Performance

After training:

- The model is evaluated on the test dataset.
- A classification report is generated showing precision, recall, F1-score for each class.
- A confusion matrix is displayed as a heatmap to visualize correct vs incorrect predictions across all classes.
- This helps you understand where the model performs well or struggles.

10. Saving the Model

- Two versions of the trained model are saved:
- Best model: Automatically saved during training when validation accuracy is highest.
- Final model: Saved explicitly at the end of training.
- These saved models can later be used for making predictions on new or live images.

✅ Summary

This CNN-based classification pipeline achieves solid accuracy (80–86%+) on posture classification using a relatively simple and efficient model. It uses essential deep learning practices such as data augmentation, dropout, batch normalization, and early stopping.