Report

Group Members

Anusha Salimath , Chandu Dadi, Priyanka Hareshbhai Sorathiya

1. Data Loading and Preparation

The first step involved loading the motion sequence datasets into pandas DataFrames for easy data manipulation. The data was structured into training and testing sets, with CSV files containing joint coordinate data for various activities. The file names were parsed to extract relevant information such as activity type, which was essential for supervised learning. Activity labels representing actions like boxing, drums, guitar, rowing, and violin were mapped to numerical categories to make them suitable for classification tasks. This mapping facilitated efficient processing by machine learning algorithms, which operate on numerical data.

Motivation: Pandas was chosen for its powerful data manipulation capabilities and efficient handling of large datasets. Organizing data in this way ensured consistency and simplified downstream processing.

2. Data Preprocessing

To ensure data quality and consistency, the dataset was examined for missing or NaN values. Handling missing data was critical to prevent the models from learning incorrect patterns. Feature normalization was applied using `StandardScaler`, which standardized the input features (joint coordinates) to have zero mean and unit variance. This step was particularly important for algorithms like SVM and KNN that are sensitive to the scale of input features.

Motivation: Normalization was crucial for models sensitive to feature scales (e.g., SVM, KNN), while tree-based models like Random Forest and XGBoost inherently handle varying scales well, making normalization unnecessary for them.

3. Feature Engineering

The datasets are considered as the sequences in our modeling approach. Each file is take as a single sequence to the model in CNN and flattened approach in the random forest model . To handle varying

sequence lengths in the motion data, the sequences were padded using TensorFlow's `pad_sequences` function, ensuring that all input data had a uniform shape suitable for model training.

Motivation: Including joint angles allowed the model to capture relative joint movements, providing richer information about the performed activity. Padding was necessary for deep learning models that require uniform input dimensions.

4. Model Development

Multiple machine learning models were developed to classify motion sequences effectively:

Traditional Machine Learning Models:

- **Random Forest Classifier:** Chosen for its robustness and ability to handle noisy and high-dimensional data.
- **Support Vector Machine (SVM):** Used due to its effectiveness in handling complex classification boundaries.
- **K-Nearest Neighbors (KNN):** Implemented for its simplicity and interpretability.
- **XGBoost:** Preferred for its scalability and regularization capabilities.

Deep Learning Model:

- **1D Convolutional Neural Network (CNN):**
    - **Layers:** Multiple convolutional layers with ReLU activation to extract spatial-temporal features.
    - **Pooling Layers:** Applied to reduce dimensionality and prevent overfitting.
    - **Fully Connected Layer:** Outputs predictions across the five activity classes.

Motivation: Traditional models were chosen for their efficiency on structured data, while CNNs were used to capture complex temporal patterns inherent in motion sequences.

5. Hyperparameter Tuning

To maximize performance, hyperparameter optimization was conducted:

- **Random Forest:** Tuned **`n_estimators`** (number of trees) and **`max_depth`** to balance bias and variance.
- **SVM:** Optimized kernel types **(linear rbf)** and regularization parameter `C`.

- **XGBoost:** Tuned **`learning_rate`**, **`max_depth`**, and **`n_estimators`** for optimal learning.
- **CNN:** Adjusted learning rate, batch size, number of filters, and kernel size.

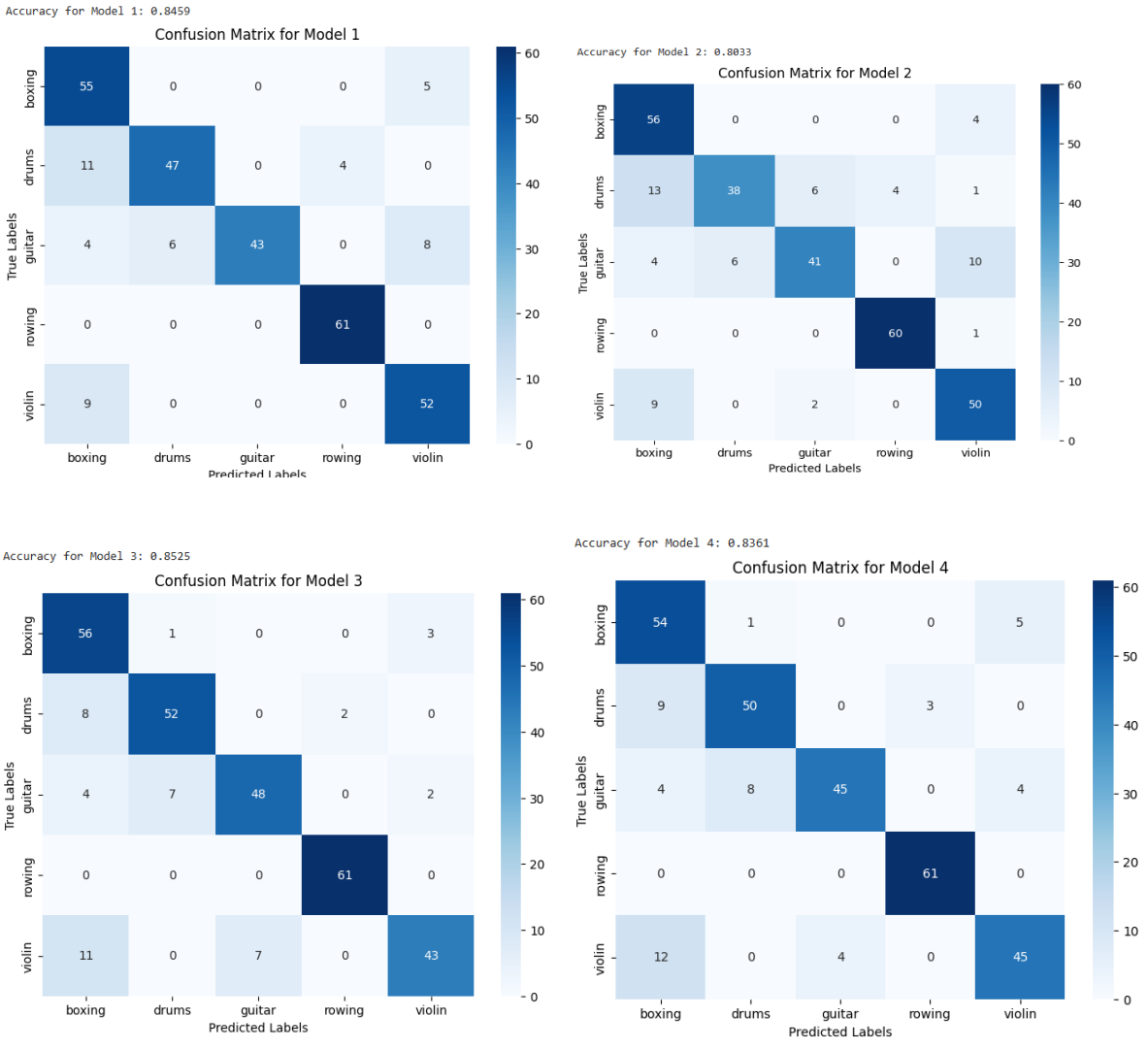
Technical Approach: **`GridSearchCV`** was used for exhaustive search on traditional models, while CNN hyperparameters were tuned manually and through trial-and-error, balancing model complexity and overfitting risk.

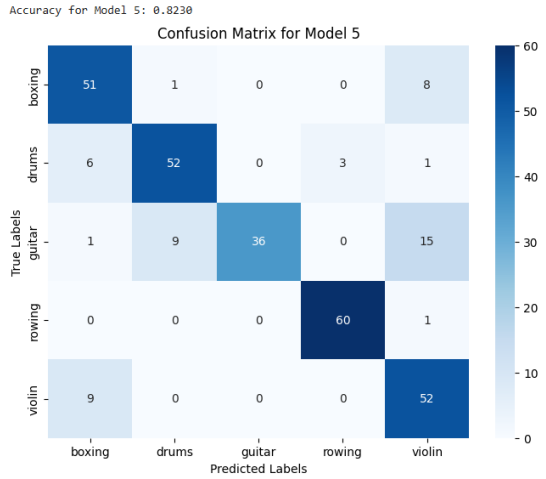
6. Model Evaluation

Models were evaluated using:

- **Accuracy Score:** To measure overall model correctness.
- **Classification Report:** To analyze precision, recall, and F1-score for each activity class.
- **Confusion Matrix:** To visualize misclassifications and performance across different classes.

Motivation: These metrics provided a comprehensive view of model performance, highlighting both overall accuracy and class-specific behaviors.





Model	Features (79)	Features (20)
Random Forest	68	64
Xgboost	73	72
CNN	79	83

CNN Model Runs for 5 times :

Model Run	Accuracy
1	84
2	80
3	85
4	83
5	82

Mean accuracy : 0.83 , std accuracy : 0.02

CNN Model is performing better in our case and it is consistently performing over 80% accuracy for all the 5 runs of the training.

The features that are selected for the second run are taken based on the ones which have an higher influence on the acitivities performed by a human. Most of the columns selected are from the upper portion of the body i.e., arms, wrists, neck, shoulder, elbow. Selecting these features made the models used performed better compared to using all the features. This particular thing helped us in training the models faster and able to experiment more with the models on hyperparameter tuning.

7. Model Saving and Deployment

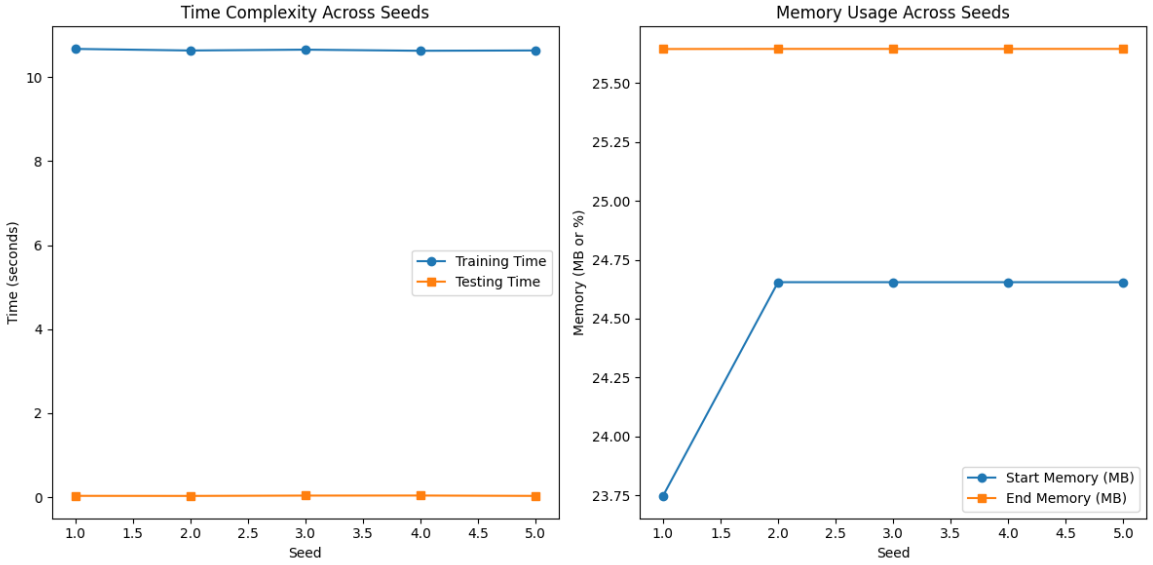
The best-performing model was saved using the **`Joblib`** library, enabling future use without retraining. This allowed seamless integration into deployment pipelines for real-world applications.

Motivation: `Joblib` was selected for its efficiency in serializing large machine learning

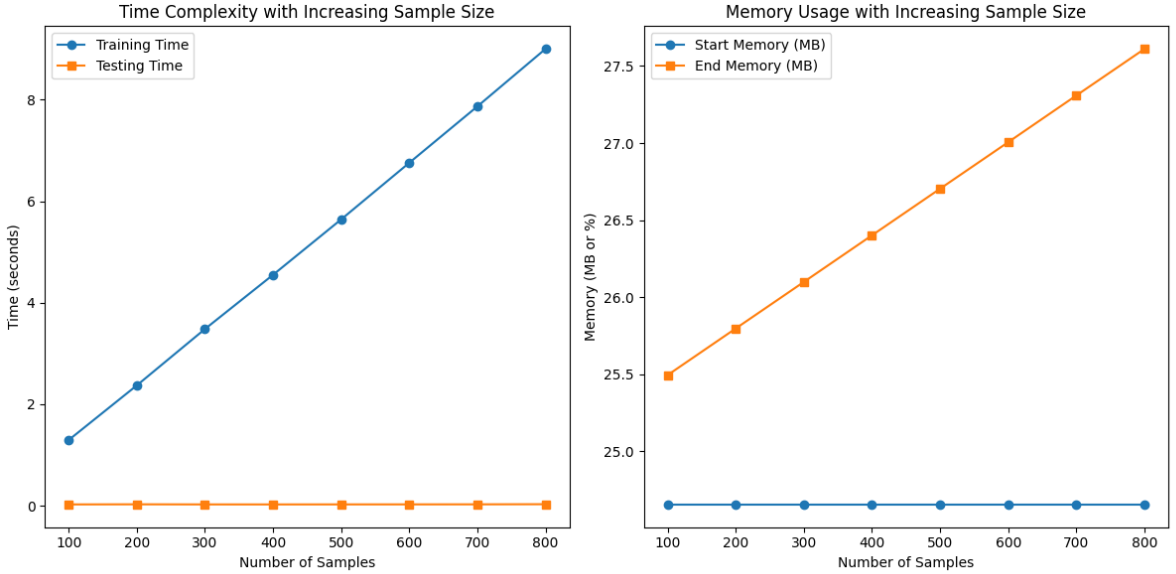
models, ensuring rapid loading and deployment.

8. Time and Memory Complexity

Time complexity of the CNN model when ran for 5 times with different seed rates to initialize the weights of the models. This plot shows that most of the times the models takes the same amount of time to train the model with different starting weights.



The CNN models are trained for different data sizes to check how the sample sizes affect the running time of the model and the memory consumption of the model during training and the inference mode.



The linear complexity of the above plot shows linear time complexity which is dependent on the sample size in this case but number of epochs and different batch size may affect it as well. The memory complexity graph also shows the same fashion of the time complexity.

9. Overall Pipeline Summary

The complete pipeline included the following structured steps:

1. **Data Loading:** Organized and loaded data efficiently into DataFrames.
2. **Preprocessing:** Handled missing data and normalized features as required.
3. **Feature Engineering:** Selected meaningful features and padded sequences for uniformity.
4. **Model Development:** Built and evaluated traditional and deep learning models.
5. **Hyperparameter Tuning:** Optimized models for best performance.
6. **Model Evaluation:** Assessed using accuracy, classification reports, and confusion matrices.
7. **Model Deployment:** Saved and prepared the best model for real-world use.

This end-to-end workflow was designed to maximize classification accuracy while ensuring scalability, robustness, and efficiency.