Project Report: Human Activity Recognition using LSTM

1. Problem Statement

The goal of this project is to develop a model that can accurately classify human physical activities based on time-series data collected from smartphone sensors. Given sequences of sensor readings from an accelerometer and a gyroscope, the task is to classify the activity being performed into one of six categories: Walking, Walking Upstairs, Walking Downstairs, Sitting, Standing, or Laying. This is a multi-class time-series classification problem that has applications in healthcare monitoring, fitness tracking, and context-aware computing.

2. Dataset Description

- **Source**: The project utilizes the **UCI Human Activity Recognition (HAR) Using Smartphones Dataset**.
- **Size and Features**: The dataset is pre-split into a training set and a test set.
  - **Training Set**: 7,352 samples.
  - **Test Set**: 2,947 samples.
  - Each sample consists of a sequence of **128 timesteps**, with **9 features** per timestep. The features are derived from the raw tri-axial signals from the accelerometer and gyroscope and include total acceleration, body acceleration, and body gyroscope readings for the X, Y, and Z axes. The input data has a shape of (samples, 128, 9).
- **Labels**: There are 6 distinct activity classes to be predicted.
- **Preprocessing**: The integer labels (1-6) were first converted to a zero-based index (0-5). Subsequently, they were **one-hot encoded** into a binary vector format (e.g., SITTING as [0,0,0,1,0,0]) to be compatible with the model's output layer and the categorical cross-entropy loss function.

3. Methodology

A deep learning approach was chosen to automatically learn temporal patterns from the raw time-series data without extensive manual feature engineering.

- **Algorithm**: A **Long Short-Term Memory (LSTM)** neural network was selected. LSTMs are a type of Recurrent Neural Network (RNN) specifically designed to handle long-term dependencies in sequence data, making them ideal for analyzing sensor readings over time.
- **Rationale**: The model employs a **stacked LSTM architecture**, where multiple LSTM layers are placed one after another. This allows the network to learn hierarchical representations of the time-series data, with earlier layers capturing low-level temporal features and later layers combining them into more abstract patterns related to specific activities.

4. Architecture Description

The model was constructed using the TensorFlow Keras Sequential API. It consists of two stacked LSTM layers followed by a final Dense output layer.

The architecture is detailed in the summary table below:

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 128, 32)	5,376
lstm_7 (LSTM)	(None, 32)	8,320
dense_3 (Dense)	(None, 6)	198

- **Input Layer**: Implicitly defined with an input shape of (128, 9).
- **First LSTM Layer**: Contains 32 units. The return_sequences=True argument ensures that the output of this layer is the full sequence of hidden states, which is necessary for feeding into the subsequent LSTM layer.
- **Second LSTM Layer**: Contains 32 units. It receives the sequence from the first layer and outputs only the hidden state of the final timestep, effectively providing a summary vector of the entire input sequence.
- **Output (Dense) Layer**: A fully connected layer with 6 neurons, corresponding to the 6 activity classes. It uses a **softmax** activation function to output a probability distribution over the classes.

5. Evaluation & Implementation

- **Dataset Split**: The pre-defined training set (7,352 samples) was used for training the model. The test set (2,947 samples) was used as the **validation set** during training to monitor performance and as the final hold-out set for evaluation.
- **Loss Function**: **Categorical Crossentropy** was used as the loss function, which is standard for multi-class classification problems with one-hot encoded labels.
- **Evaluation Metrics**: Model performance was primarily assessed using **accuracy**. For a more detailed analysis, **Precision**, **Recall**, and **F1-Score** (with a 'weighted' average) were also calculated on the test set.
- **Implementation Details**:
    - **Optimizer**: Adam
    - **Batch Size**: 64
    - **Epochs**: The model was trained for a maximum of 100 epochs.
- **Other Methods**:
    - **Callbacks**: Two Keras callbacks were used to improve training efficiency and prevent overfitting:
        1. **EarlyStopping**: Monitored the validation loss (val_loss) and stopped training if there was no improvement for 5 consecutive epochs. It was configured to restore the weights from the epoch with the best val_loss.
        2. **ReduceLROnPlateau**: Reduced the learning rate by a factor of 0.2 if the

validation loss did not improve for 5 epochs.

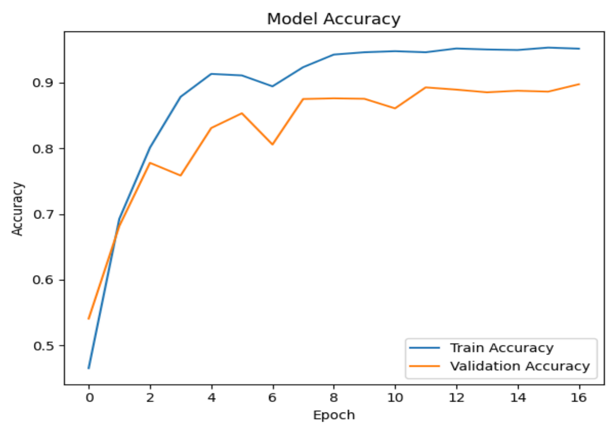
6. Results & Analysis

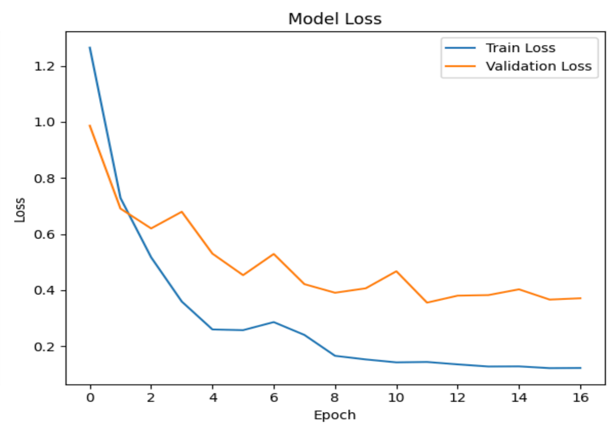
The model achieved a **final test accuracy of 89.24%** with a **test loss of 0.3553**.

Detailed Classification Metrics:

- **Precision**: 0.8934
- **Recall**: 0.8924
- **F1-Score**: 0.8922

Visualizations & Analysis:





7. Future Improvements

While the model achieved a strong baseline performance, several avenues for improvement exist:

- **Hyperparameter Tuning**: Systematically tune hyperparameters like the number of LSTM units, batch size, and the Adam optimizer's learning rate might help in improving further performance.