Configuration Manual

MSc Research Project
Data Analytics

Rehan Shariff
Student ID: x22246339@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rehan Shariff
Student ID:	x22246339@student.ncirl.ie
Programme:	Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Jaswinder Singh
Submission Due Date:	29 January 2025
Project Title:	Configuration Manual
Word Count:	1000
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Rehan Shariff
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rehan Shariff
x22246339@student.ncirl.ie

1 Introduction

This configuration manual provides detailed instructions for setting up, configuring, and deploying the **Physical Activity Recognition** system using wearable sensor data. The system leverages deep learning models, including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, and a hybrid CNN-LSTM architecture, combined with hybrid feature selection techniques to optimize performance. Additionally, the manual covers the deployment of a Flask-based API for real-time activity prediction.

2 System Requirements

Before proceeding with the installation and configuration, ensure that your system meets the following requirements:

2.1 Hardware Requirements

- **Processor**: Intel i5 or higher
- **RAM**: Minimum 8 GB
- **Storage**: At least 20 GB of free space
- **GPU**: NVIDIA GPU with CUDA support (optional, for faster model training)

2.2 Software Requirements

- **Operating System**: Windows 10/11, macOS, or Linux
- **Python**: Version 3.7 or higher
- **Libraries and Dependencies**:
  - numpy
  - pandas
  - scikit-learn
  - TensorFlow
  - Keras
  - Flask
  - joblib
  - plotly
  - flask_cors
- **Development Tools**:
  - Jupyter Notebook or Google Colab

- Integrated Development Environment (IDE) like VS Code or PyCharm
- Git for version control

3 Installation

3.1 Create a Virtual Environment

It is recommended to use a virtual environment to manage dependencies.

3.2 Activate the Virtual Environment

Activate the virtual environment using the appropriate command for your operating system.

3.3 Install Dependencies

Install all required Python libraries using `pip`.

4 Data Preparation

4.1 Download the Dataset

Ensure that the dataset is placed in the designated directory with the following structure:

Figure 1: Dataset Directory Structure

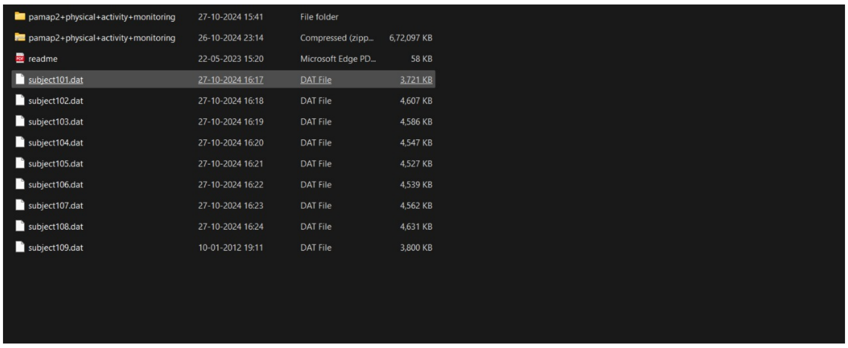


Figure 1: Dataset Directory Structure

4.2 Data Loading and Preprocessing

The data is loaded and preprocessed to handle missing values, normalize features, and encode activity labels.

Figure 2: Data Preprocessing Process

```python
# Load and preprocess data
def load_data(data_dir):
    files = glob.glob(f"{data_dir}/*.dat")
    data = []
    for file in files:
        df = pd.read_csv(file, delim_whitespace=True, header=None)
        data.append(df)
    data = pd.concat(data, ignore_index=True)

    # Assign 54 column names as per dataset structure
    data.columns = (
        ['timestamp', 'activityID', 'heart_rate'] +
        [f'hand_{i}' for i in range(1, 18)] +
        [f'chest_{i}' for i in range(1, 18)] +
        [f'ankle_{i}' for i in range(1, 18)]
    )
    data.replace(to_replace="NaN", value=np.nan, inplace=True)
    return data
```

```python
# Impute missing values, normalize, and encode labels
def preprocess_data(data):
    data.dropna(subset=['activityID'], inplace=True)
    data.fillna(method='ffill', inplace=True)
    X = data.drop(columns=['timestamp', 'activityID'])
    y = data['activityID'].astype(int)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    encoder = LabelEncoder()
    y_encoded = encoder.fit_transform(y)
```

Figure 2: Data Preprocessing Process

5 Feature Selection

5.1 Hybrid Feature Selection Approach

The system employs a hybrid feature selection strategy combining Elastic Net regularization and Random Forest with Mutual Information to identify the most relevant features.

Figure 3: Feature Selection Process

```python
# Feature selection with Elastic Net, ensuring a minimum number of features
def select_features_elasticnet(X, y, alpha=0.01, l1_ratio=0.5, min_features=5):
    enet = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)
    enet.fit(X, y)
    selected_features = enet.coef_ != 0
    if selected_features.sum() < min_features:
        selected_indices = np.argsort(np.abs(enet.coef_))[-min_features:]
        selected_features[selected_indices] = True
    selected_indices = np.where(selected_features)[0]
    X_selected = X[:, selected_indices]
    return X_selected, selected_indices
```

```python
# Baseline feature selection with Random Forest and Mutual Information
def select_features_baseline(X, y, min_features=5):
    rf = RandomForestClassifier()
    rf.fit(X, y)
    rf_features = rf.feature_importances_ > np.mean(rf.feature_importances_)

    mi = mutual_info_classif(X, y)
    mi_features = mi > np.mean(mi)

    selected_features = rf_features | mi_features

    if selected_features.sum() < min_features:
        selected_indices = np.argsort(rf.feature_importances_)[-min_features:]
        selected_features[selected_indices] = True

    selected_indices = np.where(selected_features)[0]
```

Figure 3: Feature Selection Process

5.2 Saving Selected Features

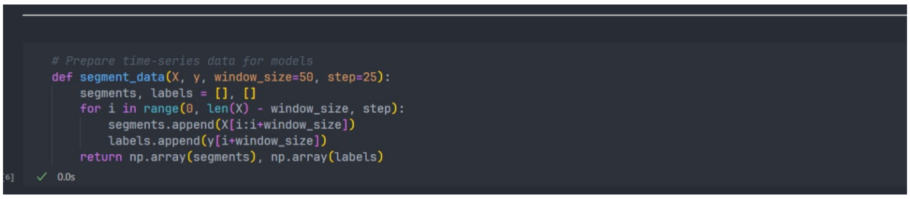
After feature selection, the scaler, selected feature indices, and label encoder are saved for future use.

6 Data Segmentation for Time-Series Modeling

6.1 Segmenting the Data

The dataset is divided into overlapping segments suitable for sequential models.

Figure 4: Data Segmentation Process



```python
# Prepare time-series data for models
def segment_data(X, y, window_size=50, step=25):
    segments, labels = [], []
    for i in range(0, len(X) - window_size, step):
        segments.append(X[i:i+window_size])
        labels.append(y[i+window_size])
    return np.array(segments), np.array(labels)
```

Figure 4: Data Segmentation Process

6.2 Splitting the Data

The segmented data is split into training and testing sets using an 80-20 split.

7 Model Architecture

7.1 Convolutional Neural Network (CNN)

The CNN model captures spatial patterns within each time step.

Figure 5: CNN Model Architecture

```python
# Build CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv1D(64, kernel_size=3, activation='relu', input_shape=input_shape),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

# Build LSTM model
def create_lstm_model(input_shape, num_classes):
    model = Sequential([
        LSTM(64, input_shape=input_shape),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Figure 5: CNN Model Architecture

7.2 Long Short-Term Memory (LSTM)

The LSTM model focuses on capturing temporal dependencies in the data.

Figure 6: LSTM Model Architecture

```python
# Build LSTM model
def create_lstm_model(input_shape, num_classes):
    model = Sequential([
        LSTM(64, input_shape=input_shape),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Figure 6: LSTM Model Architecture

7.3 CNN-LSTM Hybrid Model

Combining CNN and LSTM layers to capture both spatial and temporal patterns.

Figure 7: CNN-LSTM Model Architecture

```python
# Build CNN-LSTM model
def create_cnn_lstm_model(input_shape, num_classes):
    model = Sequential([
        Conv1D(64, kernel_size=3, activation='relu', input_shape=input_shape),
        MaxPooling1D(pool_size=2),
        LSTM(64),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Figure 7: CNN-LSTM Model Architecture

8 Model Training and Evaluation

8.1 Training the Models

Each model is trained with early stopping to prevent overfitting. Training time and performance metrics are recorded.

8.2 Evaluating Model Performance

After training, models are evaluated using metrics such as accuracy, precision, recall, and F1 score.

Figure 8: Model Evaluation Metrics

```python
# Train and evaluate models
def evaluate_model(model, X_train, X_test, y_train, y_test):
    start_time = time.time()
    early_stopping = EarlyStopping(
        monitor='val_loss', patience=5, restore_best_weights=True
    )
    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=32,
        validation_split=0.2,
        callbacks=[early_stopping],
        verbose=1
    )
    training_time = time.time() - start_time

    y_pred_probs = model.predict(X_test)
    y_pred = y_pred_probs.argmax(axis=1)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)
    conf_matrix = confusion_matrix(y_test, y_pred)

    return acc, prec, rec, f1, training_time, conf_matrix
```

```python
data_dir = "dataset"
data = load_data(data_dir)
X, y, encoder, scaler = preprocess_data(data)
```

Figure 8: Model Evaluation Metrics

9 Model Comparison

9.1 Visualizing Performance Metrics

Performance metrics for all models are visualized using Plotly to facilitate comparison.

Figure 9: Model Comparison Bar Chart

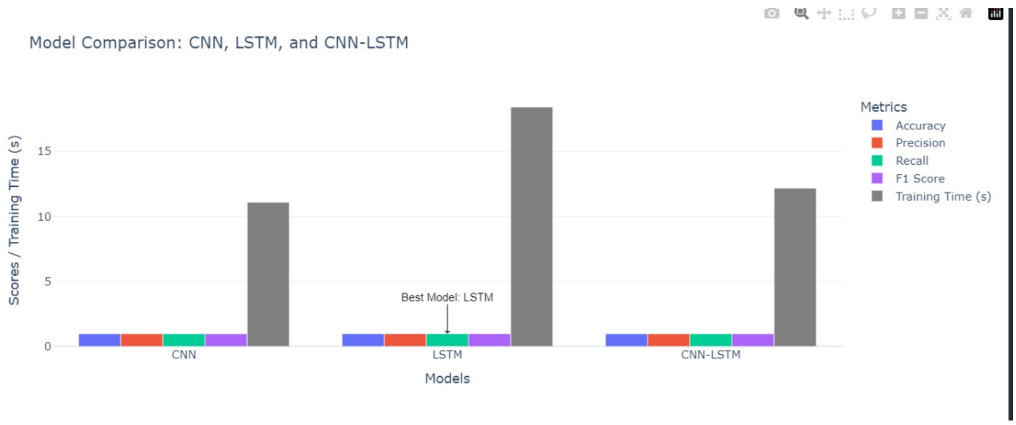


Figure 9: Model Comparison Bar Chart

9.2 Identifying the Best Model

The model with the highest F1 score is identified as the best-performing model.

10 Saving and Deploying the Best Model

10.1 Saving the Model

The best-performing model is saved using TensorFlow's model saving utilities.

Figure 10: Saving the Best Model

Figure 10: Saving the Best Model

10.2 Setting Up the Flask API

A Flask API is set up to serve the saved model for real-time predictions.

Figure 11: Flask API Setup

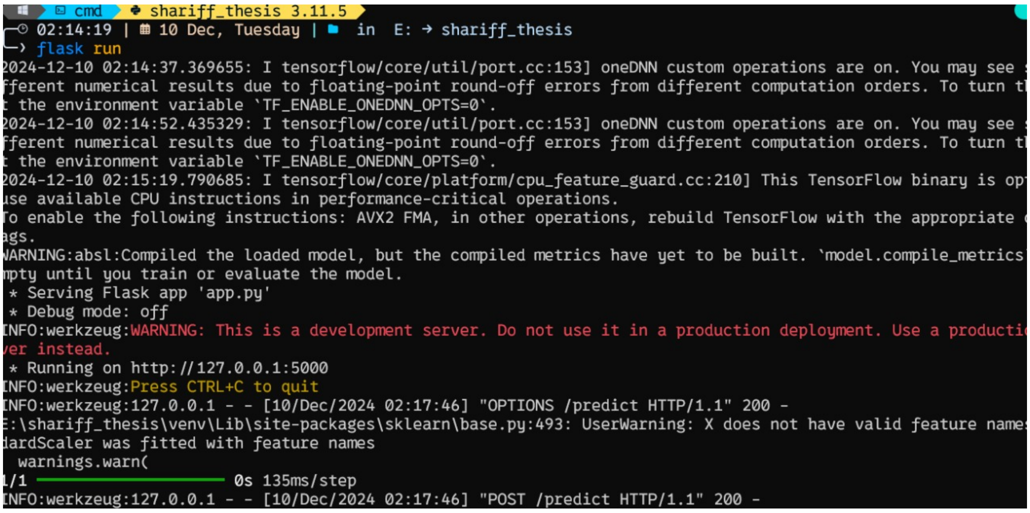


Figure 11: Flask API Setup

10.3 Running the Flask API

Start the Flask server to make the API available for predictions.

Figure 12: Running Flask API

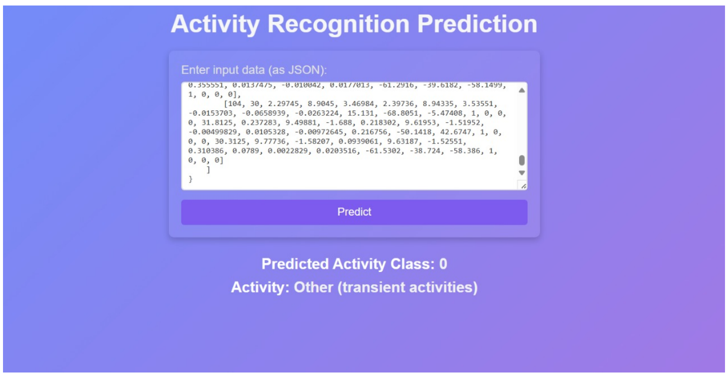


Figure 12: Flask API Running Output

11 Making Predictions via the API

11.1 API Endpoint

The API provides a single endpoint `/predict` that accepts POST requests with sensor data for activity prediction.

11.2 Request Format

Send a JSON payload with the key `input`, containing a 2D array of sensor data.

11.3 Response Format

The API responds with a JSON containing the predicted class, label, prediction probabilities, and Mean Squared Deviation Ratio (MSDR).

12 Conclusion

This manual provides a step-by-step guide to configuring and deploying a deep learning-based physical activity recognition system. By following the instructions, users can set up the environment, preprocess data, perform hybrid feature selection, train and evaluate models, and deploy the best model using a Flask API.