Human Activity Recognition Using MotionSense Dataset

Prepared by:
John Michael (MCHJOH015)
Justin Cross (CRSKAT005)

Prepared for:
EEE4114F
Department of Electrical Engineering
University of Cape Town

May 30, 2025

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

[illegible]	May 30, 2025
Justin Cross	Date
[illegible]	May 30, 2025
John Michael	Date

Contents

Chapter 1

Introduction

1.1 Background

Human Activity Recognition (HAR) is the automated process of identifying and classifying physical activities using sensor data. With the rise of smart wearable devices, HAR has advanced significantly in areas such as fitness tracking, healthcare, and smart environments.

This project uses the MotionSense dataset, which captures inertial data from accelerometers and gyroscopes during various activities. This data enables the development of machine learning models to accurately distinguish between actions like walking, jogging, and sitting.

To improve model performance, the project applies preprocessing techniques and employs deep learning architectures, which effectively capture the temporal and spatial patterns in sensor data, contributing to the development of reliable HAR systems.

1.2 Objectives

The primary objectives of this project are to effectively preprocess and segment the MotionSense dataset, ensuring that the raw sensor data is transformed into a format suitable for machine learning applications. A suitable machine learning model is then selected, trained, and rigorously evaluated to assess its performance on unseen test data. Additionally, the proposed approach is compared against existing HAR techniques to benchmark its overall effectiveness and highlight its strengths and potential limitations relative to established methods.

1.3 Report Outline

- **Chapter 2:** Literature review outlining recent advancements in HAR.
- **Chapter 3:** Design and implementation of the proposed HAR model, with comparison to existing approaches.
- **Chapter 4:** Evaluation and analysis of the model's performance on test data.
- **Chapter 5:** Conclusion summarizing the model's effectiveness and overall project outcomes.

Chapter 2

Literature Review

2.1 Introduction to Human Activity Recognition (HAR)

Human Activity Recognition (HAR) is an important area of research with practical applications in our daily lives, such as tracking physical activity through fitness apps, monitoring the elderly for fall detection, and improving health monitoring capabilities. However, building effective HAR systems comes with its challenges , including differences in sensor types (e.g., smartphones, smartwatches, fitness bands), variations in how often data is recorded , and the time-consuming process of manually labelling activity data. Overcoming these hurdles is critical to creating accurate and efficient HAR solutions for real-world use.

2.2 Preprocessing and Input Adaptation Techniques

A common theme across recent works in HAR is the critical role of preprocessing in transforming raw sensor data into formats that are more informative and robust. Effective preprocessing enhances the model's ability to generalize across different subjects, environments, and activity patterns, ultimately improving classification performance.

Ravuri et al.[3] emphasize the importance of denoising techniques to address noise introduced by sensor miscalibration and environmental variability. They discuss widely used filtering methods such as low-pass filters, mean filters, and wavelet filters, which help to improve signal quality prior to feature extraction. Additionally, the study highlights the necessity of data segmentation, as human activities often occur over time spans that exceed typical sensor sampling intervals. To tackle this, continuous data streams are divided into smaller, more manageable segments through various windowing strategies, including time-driven, event-driven, and action-driven approaches.

Building on the importance of signal transformation, Nouriani et al.[1] introduce a high-gain observer (HGO) to estimate roll and pitch angles from raw inertial measurement unit (IMU) data. The signals are then filtered to mitigate drift and converted into spectrograms which are time-frequency representations that are further encoded as RGB images. These spectrogram images enable the application of computer vision-based convolutional neural networks (CNNs), thereby leveraging the power of pretrained image classification models for activity recognition tasks.

In a complementary approach, Saeed et al.[2] propose a self-supervised learning framework that relies on eight pretext transformation tasks , such as flipping, scaling, and permutation—applied to raw input signals. The model is trained to recognize these transformations, allowing it to learn meaningful representations of activity data without requiring manual labels. This method not only reduces the dependency on annotated datasets but also improves the model's robustness.

Collectively, these works demonstrate the value of advanced preprocessing and input adaptation techniques in enhancing the performance and flexibility of HAR systems. From signal denoising and segmentation to the generation of alternative data representations and self-supervised learning, preprocessing remains a foundational element in modern activity recognition pipelines.

2.3 Deep Learning Techniques and Model Architectures

Recent advancements in HAR have leveraged a wide range of deep learning strategies to process and classify complex patterns from sensor data. These approaches vary in how they extract features, handle data, and adapt to different types of inputs.

Nouriani et al.[1] and Saeed et al.[2] both explore feature learning from sensor data, through distinct methodologies. Nouriani et al.[1] employ a physics-informed preprocessing pipeline that transforms filtered inertial data into spectrograms visual representations capturing the time-frequency characteristics of signals. These spectrograms are treated as RGB images and processed using computer vision CNNs originally designed for image classification. This fusion of signal filtering and vision-based deep learning allows for the reuse of powerful pretrained models, improving recognition performance.

Expanding on these ideas, Malekzadeh et al.[4] present a hybrid model that integrates CNNs, Recurrent Neural Networks (RNNs), and Feedforward Neural Networks (FNNs) to enhance classification accuracy. A key contribution of their work is the introduction of the Dimension-Adaptive Pooling (DAP) layer, which adjusts pooling operations based on the input's temporal characteristics. This layer improves the model's adaptability without compromising performance, particularly when dealing with variable-length input data. Building on DAP, the authors also propose a Dimension-Adaptive Transformation (DAT) layer that aligns feature dimensions across heterogeneous sensor modalities, ensuring consistent input representations throughout the network. Notably, their CNN-RNN configuration , with significantly fewer parameters than the FNN baseline, achieves better classification accuracy. The addition of the DAP layer further enhances test performance, highlighting its effectiveness in improving generalization.

Collectively, these studies illustrate how a diverse set of deep learning architectures , including image-based CNNs, temporal self-supervised models, and hybrid adaptive networks which can effectively address the challenges of feature extraction, input variability, and label scarcity in HAR systems.

2.4 Strategies for Dealing with Limited Data

Handling data scarcity remains a critical challenge in developing robust HAR systems capable of performing well in real-world scenarios. Given the high cost of obtaining labeled activity datasets, especially across diverse users and contexts, researchers have explored a range of strategies to mitigate the limitations imposed by limited training data.

Nouriani et al.[1] address the scarcity of labeled data by leveraging transfer learning. Specifically, they repurpose pretrained deep learning models originally trained on large-scale image datasets such as

ImageNet to classify spectrograms derived from filtered inertial measurement unit (IMU) signals. This approach enables the reuse of rich visual feature representations, thereby circumventing the need for extensive labeled HAR datasets and improving classification performance with minimal additional training.

In a contrasting yet complementary approach, Saeed et al.[2] propose a self-supervised learning framework that eliminates the reliance on labeled data altogether. Their method involves training a temporal CNN using a suite of pretext tasks, including signal flipping, permutation, and scaling. The model is trained to recognize these transformations, allowing it to learn robust and generalizable representations directly from raw time-series sensor data. These features can then be used for HAR classification tasks with minimal supervision.

Further expanding on transfer learning, Ravuri et al.[3] examine strategies to address data variability in cross-domain HAR settings, where models must adapt to differences in subjects, locations, and sensor modalities. The authors emphasize domain adaptation as a key technique for aligning heterogeneous data distributions between source and target domains. Domain adaptation works by minimizing discrepancies in feature space, thereby enabling better generalization when models are deployed in new or unseen contexts.

Among the techniques explored, Deng et al.[5] propose a cross-person activity recognition framework based on condensed kernel extreme learning machines. Their model improves cross-subject generalization by incorporating both target samples and high-confidence pseudo-labeled samples into the training process, effectively bridging the domain gap between individuals.

Additionally, recent studies have explored multi-task learning and generative approaches to enhance generalization under limited data conditions. For example, Chen et al.[6] introduce the METIER model , a multi-task deep learning architecture designed to perform both user identification and activity recognition concurrently. By sharing parameters across these tasks and utilizing a shared attention mechanism, the model captures transferable features that enhance recognition accuracy across diverse user domains and sensor inputs.

These methods highlight an increasing emphasis on data-efficient learning paradigms in HAR. By leveraging external datasets, learning from unlabeled transformations, or aligning domain-specific variations, these approaches address the practical challenges posed by limited and heterogeneous training data.

2.5 Performance Evaluation and Benchmarking

The reviewed studies employ a variety of datasets and performance metrics to evaluate their HAR models, each emphasizing different strengths and highlighting key trade-offs between accuracy, model complexity, and practical deployment considerations.

Nouriani et al.[1] report an impressive classification accuracy of up to 98% on a real-world, home-collected dataset. This demonstrates the robustness of their approach in capturing critical yet infrequent activities within practical, real-life scenarios. Their results underscore the potential of leveraging pretrained visual models on spectrogram representations of sensor data.

In contrast, Saeed et al.[2] focus on semi-supervised learning frameworks that perform effectively even with severely limited labeled data , achieving strong classification results with as few as 10 labeled samples per class. By utilizing rich feature representations learned through self-supervised transformation prediction tasks, their model enhances generalization capabilities in data-scarce environments.

Ravuri et al.[3] provide a comparative analysis of various HAR architectures, contrasting sensor-based and vision-based methods. Within vision-based models, the 5-CNN architecture, along with the combination of CNNs and autoencoders (AE), achieved classification accuracies exceeding 95%. Similarly, among sensor-based approaches, continuous autoencoder models demonstrated superior performance, with customized CNNs and deep belief networks (DBNs) also surpassing the 95% accuracy threshold.

Malekzadeh et al.[4] conducted a benchmarking study comparing three established Deep Neural Network (DNN) architectures before and after integrating their proposed DAP layer. To ensure fairness, the adaptive models were designed to have equivalent parameter counts to the original non-adaptive models by adjusting the DAP layer dimensions accordingly. Experiments were conducted under consistent conditions, including a fixed sensor sampling rate of 50 Hz and the availability of both gyroscope and accelerometer data. The results demonstrated that incorporating the DAP layer did not degrade classification accuracy, confirming that the added flexibility from this adaptive mechanism can be achieved without compromising overall model performance.

Together, these studies reveal important insights into the balance between model complexity, data availability, and adaptability, providing valuable guidance for the development of effective HAR systems suited to diverse real-world conditions.

2.6 Conclusion and Future Research

This literature review explores the advancements in Human Activity Recognition systems, highlighting how effective preprocessing techniques, such as noise reduction, signal segmentation, and converting signals into visual formats like spectrograms , enhance the performance of deep learning models.

Modern approaches like self-supervised learning and input adaptation allow models to learn meaningful patterns from smaller datasets. Deep learning such as CNNs and hybrid models like CNN-RNNs, have proven especially powerful in recognizing activity patterns, especially when advanced techniques such as adaptive pooling are used.

Despite significant progress, HAR systems still face challenges such as limited labeled data and varying sensor types. To address these issues, strategies like transfer learning and domain adaptation help models perform well in new environments or with new users, without needing to be retrained from scratch. Looking ahead, future research should focus on blending self-supervised learning with domain adaptation, integrating multiple sensor types, and improving the speed and efficiency of these systems for real-world applications , particularly in healthcare, fitness, and smart living environments.

Chapter 3

Design Section

3.1 Data Preprocessing: Cleaning and Preparing Raw Data

Windowed Data Segmentation for Time-Series Processing

To effectively model patterns in the raw sensor data, we applied a **sliding window segmentation** technique during preprocessing. This decision was chosen based on the nature of the dataset, which exhibits time-dependent characteristics , typical in motion data , where the context of a sample is determined not only by its instantaneous value but also by its neighboring values over time.

Function Overview:
The custom function `create_windowed_data()` segments the raw input data into overlapping windows of fixed length (`window_size = 400`), with a step size of `stride = 200`. For each window, the data is reshaped to have the form (features, window size), making it suitable for input into models like 1D CNNs or LSTMs. Corresponding labels are assigned to each window based on the most frequently occurring class within the window (majority voting), which provides robustness against label fluctuations.

Alternative Approaches Considered:

- **Non-overlapping windows:** These reduce computational cost but produce fewer samples and may miss transitions between classes, especially in rapidly changing sequences.
- **Labeling based on final sample in window:** This approach is more prone to noise and outlier values, especially if transitions occur within a window.

Benefits:

- **Capturing Local Patterns in Time:** Using fixed-size windows helps the model learn what is happening over short periods. This is useful for time-based data that changes quickly, like different types of movement (e.g., walking vs. running).
- **Reducing Noise with Majority Labels:** By assigning each window the most common label in that segment, we reduce the effect of occasional mistakes or short, unusual events in the data.
- **More Training Data with Overlapping Windows:** Using overlapping windows (50% in this case) gives us more samples from the same dataset, helping the model learn better and avoid overfitting.
- **Maintaining Accurate Labels:** Using the most frequent label in each window ensures that the

label reflects the main activity or state in that segment, which is especially helpful when some classes occur less often or only briefly.

In Conclusion , the Sliding window segmentation with overlapping intervals and majority-voted labels offers a principled approach to structuring sequential data for supervised learning. It balances robustness, computational efficiency, and temporal fidelity, making it an optimal choice for this machine learning pipeline.

3.2 Data Splitting and Resampling Strategy

The dataset was divided sequentially into three parts: **training (70%)**, **validation (15%)**, and **test (15%)** sets. We chose to split the data in order without shuffling to preserve the natural time order of the sensor data. This is important because randomly mixing time-series data can break the sequence patterns, which are crucial for tasks like human activity recognition.

The training set is used to teach the model the underlying patterns and temporal relationships in the data. The validation set, which the model does not see during training, helps us adjust hyperparameters and check if the model is overfitting. Overfitting happens when a model learns the training data too well but fails to perform on new data. Finally, the test set is kept completely separate from training and validation. It is used at the very end to evaluate how well the model performs on completely new, unseen data, similar to how it would behave in real-life use.

This three-part split strategy supports effective model development and evaluation. Having a separate validation set prevents us from tuning the model based on the test data, ensuring an unbiased performance measure. The 70/15/15 split also balances having enough data to train the model while still allowing reliable tuning and testing.

Alternative Strategies Considered:

- **Random Shuffling and Splitting:** This is a common approach where the data is mixed up and then split into training, validation, and test sets. However, for time-series data like ours, this can cause problems. Shuffling breaks the natural time order, which may lead to information from the future leaking into the training set, resulting in unrealistic and overly optimistic performance.
- **K-Fold Cross-Validation:** This technique splits the data into k parts (or 'folds') and trains the model multiple times, each time using a different fold as the validation set. Although it's good for getting a reliable estimate of model performance, it's not ideal for time-series data. It is harder to ensure that the time order is preserved in each fold, and the repeated training makes it more computationally expensive.
- **Non-Overlapping Windows:** In this method, the data is split into segments with no overlap. While this reduces the amount of duplicated data and is faster to process, it can miss important transitions between activities. Overlapping windows help include more continuous information and improve the model's understanding of what happens between activities, making it better at generalizing to new data.

Benefits of the Chosen Setup:

- Keeps the natural time order of the data intact.
- Prevents data leakage and gives a more realistic evaluation of the model.
- Helps the model generalize better by using overlapping windows.
- Makes model performance easier to understand and reproduce.
- Provides a clear and reliable split for training, validation, and testing, suited for time-series data.

In summary, this resampling and splitting strategy fits well with the time-based nature of the HAR task. It supports accurate, reliable model development while preserving important time patterns in the data.

3.3 Model Selection

Model Design: 1D Convolutional Neural Network (CNN1D)

For this project, we designed and implemented a 1D Convolutional Neural Network (CNN1D).The model is designed to recognize patterns over time by applying convolution operations across the input.

The input to the model is a 3D tensor with shape `(batch_size, channels, time_steps)`, for example `(32, 6, 400)`. This means we have 32 samples in a batch, each with 6 sensor channels and 400 time steps.

The model has two main convolutional blocks. Each block includes:

- A 1D convolution layer to detect patterns over time.
- Batch normalization to stabilize training.
- A ReLU activation function to introduce non-linearity.

After the two max-pooling layers, the time dimension shrinks from 400 to 100. The output is then flattened into a single vector and passed through a fully connected (dense) layer. A dropout layer is used here to reduce overfitting. Finally, the output layer produces the predicted scores for each activity.

Why CNN1D Was Chosen:

- **Efficient Feature Extraction:** 1D convolutions are well-suited for time-series data, as they can learn meaningful temporal patterns in a hierarchical manner.
- **Computational Efficiency:** Compared to more complex models like RNNs, CNNs are faster to train and easier to optimize since there are fewer time-based patterns to capture
- **Simplicity and Scalability:** The architecture is straightforward yet powerful enough to handle multi-channel inputs.

- **Robustness to Noise:** The use of convolution and pooling helps the model focus on dominant patterns and ignore noisy fluctuations in the signal.

Alternatives Considered:

- **Recurrent Neural Networks (RNNs) / LSTM:** While RNNs can model time-based patterns explicitly, they are slower to train and suffer from gradients depelting for long sequences.
- **Traditional ML Methods (e.g. Random Forest):** These require handcrafted features and lack the ability to automatically learn time-based patterns directly from raw data, which limits their performance on complex signals.

Benefits of the CNN1D Approach:

- Captures local and hierarchical patterns in sensor signals effectively.
- Processes fixed-length windows efficiently using shared weights and convolutional kernels.
- Generalizes well across users and activity variations with relatively low overfitting.

In conclusion, the CNN1D model provides an effective trade-off between accuracy, efficiency, and simplicity for time-series classification tasks in human activity recognition.

3.4 Training, Validation, and Testing Strategy

3.4.1 Training Procedure

The model was trained using a supervised learning approach implemented in PyTorch. The objective of the training procedure was to minimize classification error across a labeled dataset by optimizing the model's parameters. The training process adheres to a standard deep learning workflow, which includes iterative updates based on error minimization using backpropagation and gradient descent. A detailed description of the training process is provided below.

Model Preparation

Before training begins, the model is transferred to the appropriate computational device, either a GPU or CPU, depending on hardware availability. This ensures that all computations are executed efficiently. The model is then initialized and configured for training using the Adam optimizer with a learning rate of 0.001. The chosen loss function is CrossEntropyLoss, which is well-suited for multi-class classification tasks. It evaluates the discrepancy between the predicted class probabilities and the true class labels, penalizing incorrect predictions and guiding the model to improve its classification accuracy over time.

Training Loop

The training loop was executed over multiple epochs, during which the model iteratively updated its parameters to minimize prediction error. The key steps of the training procedure are outlined below:

1. **Training Mode Activation:** The model is set to training mode using `model.train()` to ensure that layers such as batch normalization and dropout behave appropriately during training.
2. **Mini-batch Loading:** The training dataset is divided into mini-batches using a `DataLoader`, with each batch containing input tensors and corresponding ground truth labels.
3. **Device Transfer:** Inputs and labels are transferred to the designated computational device (CPU or GPU) for efficient processing.
4. **Gradient Reset:** Gradients from the previous batch are cleared using `optimizer.zero_grad()` to prevent unintended accumulation.
5. **Forward Pass:** The input batch is passed through the model to generate raw predictions (logits).
6. **Loss Computation:** The prediction error is calculated using the `CrossEntropyLoss` function, which quantifies the difference between predicted logits and true class labels.
7. **Backpropagation:** The computed loss is backpropagated through the network to calculate gradients with respect to the model parameters.
8. **Parameter Update:** The optimizer updates the model parameters using the computed gradients to reduce the loss.

Throughout training, key performance metrics such as loss and accuracy are monitored at each epoch to evaluate the model's learning progress.

3.4.2 Validation and Testing Procedure

To monitor the model's generalization performance during training, evaluation was performed after each epoch using a separate validation dataset. This was achieved by switching the model to evaluation mode using `model.eval()`, which ensures that layers such as dropout and batch normalization behave consistently. Additionally, gradient computation was disabled using `torch.no_grad()` to improve computational efficiency and reduce memory usage.

The model was also tested against a dedicated test dataset at the end of each epoch. This allowed for continuous assessment of the model's performance on truly unseen data, providing insight into how well the model generalizes over time. Regular testing also facilitated early detection of overfitting and enabled the comparison of multiple model states throughout training. The best-performing model could then be selected based on test accuracy, ensuring optimal performance for deployment.

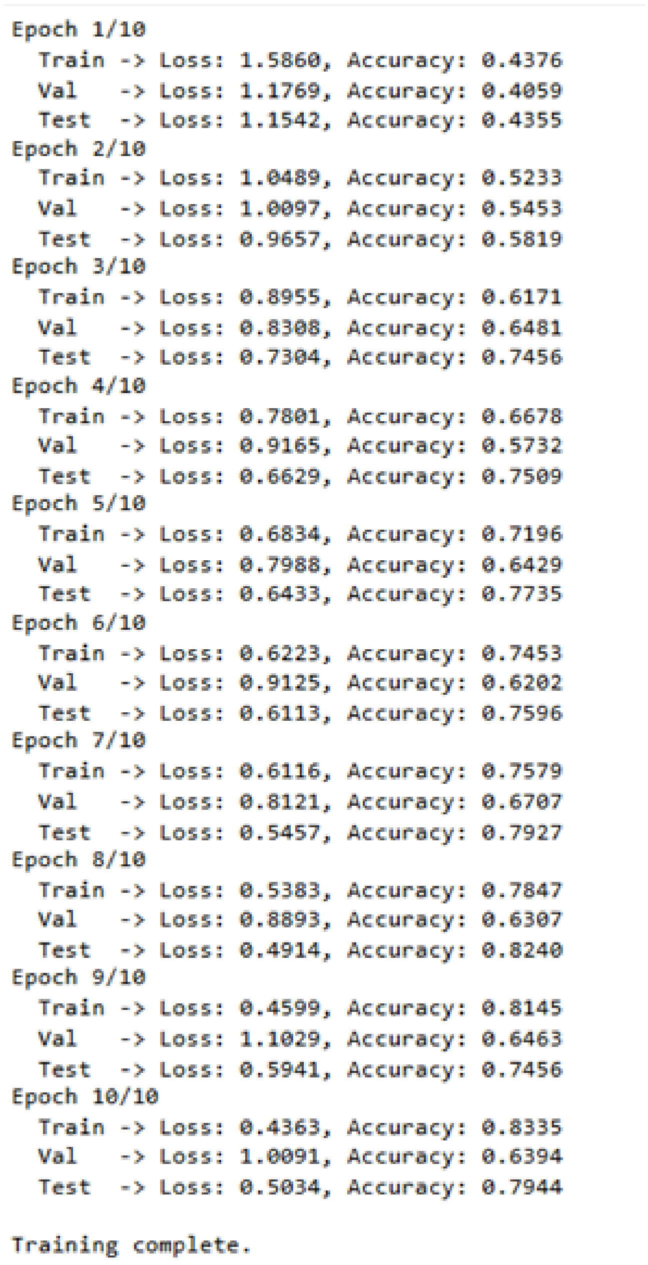
3.4.3 Alternative Optimizers Considered

- **SGD (Stochastic Gradient Descent):** While SGD often provides better generalization, it is sensitive to the choice of learning rate and generally converges more slowly compared to adaptive optimizers like Adam.
- **RMSprop (Root Mean Square Propagation):** RMSprop utilizes an adaptive learning rate for each parameter, which improves training stability; however, Adam typically achieves faster convergence by combining the benefits of RMSprop and momentum.

Chapter 4

Interpretation and analysis of results:

4.1 Results



```
Epoch 1/10
  Train -> Loss: 1.5860, Accuracy: 0.4376
  Val   -> Loss: 1.1769, Accuracy: 0.4059
  Test  -> Loss: 1.1542, Accuracy: 0.4355
Epoch 2/10
  Train -> Loss: 1.0489, Accuracy: 0.5233
  Val   -> Loss: 1.0097, Accuracy: 0.5453
  Test  -> Loss: 0.9657, Accuracy: 0.5819
Epoch 3/10
  Train -> Loss: 0.8955, Accuracy: 0.6171
  Val   -> Loss: 0.8308, Accuracy: 0.6481
  Test  -> Loss: 0.7304, Accuracy: 0.7456
Epoch 4/10
  Train -> Loss: 0.7801, Accuracy: 0.6678
  Val   -> Loss: 0.9165, Accuracy: 0.5732
  Test  -> Loss: 0.6629, Accuracy: 0.7509
Epoch 5/10
  Train -> Loss: 0.6834, Accuracy: 0.7196
  Val   -> Loss: 0.7988, Accuracy: 0.6429
  Test  -> Loss: 0.6433, Accuracy: 0.7735
Epoch 6/10
  Train -> Loss: 0.6223, Accuracy: 0.7453
  Val   -> Loss: 0.9125, Accuracy: 0.6202
  Test  -> Loss: 0.6113, Accuracy: 0.7596
Epoch 7/10
  Train -> Loss: 0.6116, Accuracy: 0.7579
  Val   -> Loss: 0.8121, Accuracy: 0.6707
  Test  -> Loss: 0.5457, Accuracy: 0.7927
Epoch 8/10
  Train -> Loss: 0.5383, Accuracy: 0.7847
  Val   -> Loss: 0.8893, Accuracy: 0.6307
  Test  -> Loss: 0.4914, Accuracy: 0.8240
Epoch 9/10
  Train -> Loss: 0.4599, Accuracy: 0.8145
  Val   -> Loss: 1.1029, Accuracy: 0.6463
  Test  -> Loss: 0.5941, Accuracy: 0.7456
Epoch 10/10
  Train -> Loss: 0.4363, Accuracy: 0.8335
  Val   -> Loss: 1.0091, Accuracy: 0.6394
  Test  -> Loss: 0.5034, Accuracy: 0.7944

Training complete.
```

Figure 4.1: Model's performance on training data, validation data and unseen test data.

4.2 Interpretation and analysis of results

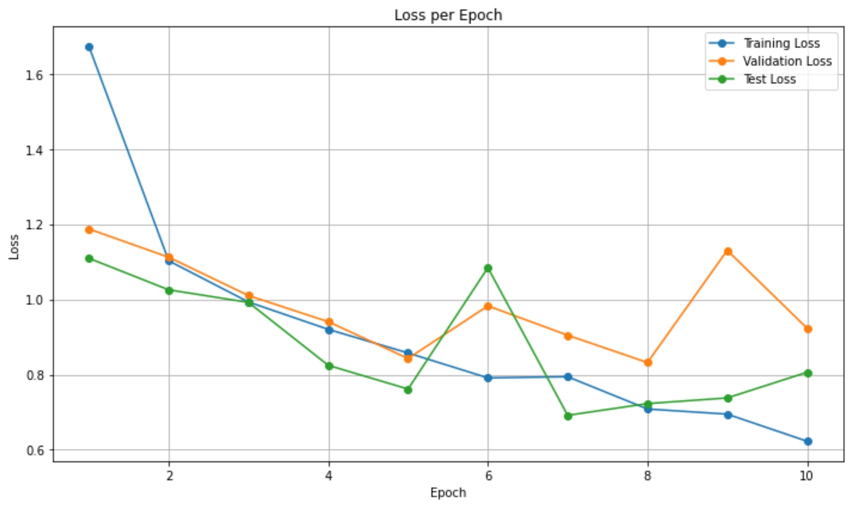


Figure 4.2: Loss function curve comparing losses across training, validation and testing.

In this training process, the dataset was divided into three distinct parts: training, validation, and test datasets. Each of these serves a specific purpose. The training set is used by the model to learn patterns and adjust hyperparameter. The validation set is used to monitor how well the model generalizes during training , as it helps detect overfitting by evaluating performance on unseen data. The test set, which is only used after training, provides an unbiased estimate of the model's performance on completely new data.

At the start of training (Epoch 1), the loss values for all three datasets , were relatively high, indicating that the model was still under fitted and had not yet learned meaningful patterns. Over the course of training, particularly by Epoch 10, the training loss dropped significantly from 1.5860 to 0.4363, and the test loss followed a similar trend, decreasing from 1.1542 to 0.5034. This steady decline shows that the model was successfully minimizing prediction errors on both seen (training) and unseen (test) data, suggesting good learning progress and improving generalization.

However, the validation loss behaved differently. While it initially decreased in the first few epochs, it remained fairly constant or even slightly increased after about Epoch 7. For example, it started at 1.1769, dropped to 0.8121 by Epoch 7, but then rose again to 1.0091 by Epoch 10. This pattern may indicate the beginning of overfitting, where the model continues to improve on training data but starts to memorize it rather than generalize well to new data. This is also supported by the observation that the training accuracy keeps increasing, reaching 83.4% by Epoch 10 , while the validation accuracy fluctuates and slightly drops in the later epochs.

The accuracy across all datasets followed a general upward trend throughout training, especially for the training and test sets. Accuracy measures how often the model correctly predicts the output, and as the epochs progressed, the model became more precise. This rise in accuracy reflects the model moving from an underfitted state, where it had high bias and poor performance, toward a more optimized state with better predictive power. However, as accuracy on training data approached very high levels while validation accuracy plateaued or declined, it suggests that model variance was increasing , this is a sign of overfitting.

Despite this, the test accuracy improved consistently, ending at 79.4%, indicating that the model was still able to generalize well to completely new data. This strong test performance shows that, while there may be signs of overfitting to the validation set, the model retains a good balance between bias and variance overall.

In summary, the model learned effectively over the 10 epochs, reducing loss and improving accuracy on both the training and test datasets. The validation results, however, highlight the importance of monitoring for overfitting. Techniques such as regularization could be useful in future iterations to improve performance further and maintain better generalization.

4.3 Practical Work - Github Repository

Below is a link to the Github repository where the code for the machine learning model is found.

GitHub Notebook:CRSKAT005_MCHJOH015_EEE4114F_Project.ipynb

Chapter 5

Conclusion

The purpose of this project was to develop a robust HAR system based on the MotionSense dataset, by applying comprehensive data preprocessing and deep learning techniques to accurately train a classification model.

This report began with an introduction to the background of HAR and its relevance in real-world applications. It then outlined the key objectives of the project, which included implementing effective preprocessing strategies, selecting an appropriate model architecture, and conducting performance benchmarking.

The literature review was presented in Chapter 2, offering insights into various preprocessing methods, deep learning architectures, and approaches to address the challenges of limited labeled data. These studies provided a strong foundational understanding that informed the subsequent stages of model development.

The bulk of the work for this project was detailed in Chapter 3, which focused on critical design decisions and methodological comparisons. This included the development of a preprocessing pipeline, the implementation of data segmentation and splitting strategies, and the evaluation of different model architectures. A 1D Convolutional Neural Network was ultimately selected due to its suitability for processing time-series data and its computational efficiency.

In Chapter 4, the performance of the selected and trained model was presented and analyzed. The results included a critical evaluation of training, validation, and testing metrics, revealing both the strengths of the model and areas for improvement. Notably, the validation set underperformed relative to the training and test sets, suggesting potential overfitting. This highlights the need for future enhancements, such as incorporating regularization techniques or domain adaptation strategies, to improve generalization.

To conclude, the project successfully met its objective of designing a machine learning model capable of learning activity patterns from the MotionSense dataset. The results demonstrate that the model shows promising potential for real-world deployment in HAR applications, with opportunities for further refinement to enhance its robustness and accuracy.

Bibliography

[1] A. Nouriani, R. McGovern, and R. Rajamani, "Activity recognition using a combination of high gain observer and deep learning computer vision algorithms," *Intelligent Systems with Applications*, vol. 18, p. 200213, Mar. 2023. [Online]. Available: https://doi.org/10.1016/j.iswa.2023.200213

[2] A. Saeed, T. Ozcelebi, and J. Lukkien, "Multi-task self-supervised learning for human activity detection," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 2, pp. 1–30, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3328932

[3] A. Ravuri, "A systematic literature review on human activity recognition," *Journal of Electrical Systems*, vol. 20, pp. 1175–1191, 04 2024.

[4] M. Malekzadeh, R. Clegg, A. Cavallaro, and H. Haddadi, "DANA," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 3, pp. 1–27, Sep. 2021.

[5] W.-Y. Deng, Q.-H. Zheng, and Z.-M. Wang, "Cross-person activity recognition using reduced kernel extreme learning machine," *Neural Networks*, vol. 53, pp. 1–7, May 2014.

[6] L. Chen, Y. Zhang, and L. Peng, "Metier: a deep multi-task learning based activity and user recognition model using wearable sensors," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–18, 2020.

[7] EEE4114F Course Staff, "Human Activity Recognition," *Department of Electrical Engineering, University of Cape Town*, Cape Town, South Africa, 2025, motionsense.ipynb.