50.039 Theory and Practice of Deep Learning Project Report

Ho Xiaoyang, Pankti Shah, Ryan Lee

April 2025

1 Introduction

Human Activity Recognition (HAR) plays a critical role in understanding the relationship between physical activity and health outcomes. With aging populations in Singapore and across Asia, there is a growing need for robust, data-driven tools to monitor physical activity in older adults and support independent living.

In this project, we apply deep learning techniques to classify physical activities performed by older adults using time-series data from wearable sensors. We base our work on two HAR datasets: the Human Activity Recognition Trondheim (HARTH) dataset [1, 3] and the Human Activity Recognition 70+ (HAR70+) [6] datasets. Our github repository can be found here.

1.1 Our Team

- Ho Xiaoyang: Model Code, Model Testing and Tuning, Diagrams
- Pantki Shah: Model Testing and Tuning, Research, Report
- Ryan Lee: Model Training and Evaluation Code, Model Testing and Tuning, Report

2 Motivation

Human activity recognition (HAR) is a research field that aims to classify and predict the physical activities and postures of human subjects. Unlike subjective data such as questionnaires, sensor data serve as objective measurements that are less susceptible to bias, recall errors, or interpretation discrepancies. Among various sensing modalities, accelerometers are the most widely used due to their affordability, small form factor, and ability to capture motion dynamics with high temporal resolution.

In recent years, deep learning (DL) has achieved remarkable success in domains such as computer vision, natural language processing, and speech recognition. One of its key advantages is its ability to learn hierarchical feature representations directly from raw data, bypassing the need for hand-crafted features that traditional methods like Support Vector Machines (SVMs) depend on. However, the performance of DL methods often correlates with the scale of the dataset. That is, DL models generally require large, primarily labeled datasets to achieve good results. For objective measurement-based HAR, this means needing annotated sensor recordings, where the performed activity (the ground truth label) is known at each recorded time step. Creating high-quality activity annotations is costly and time-consuming as it requires a demanding human labeling process. As a result, accelerometer-based HAR datasets tend to be relatively small and tightly curated to ensure label quality. Consequently, it is more common to use less data-intensive machine learning approaches for HAR (e.g. XGBoost) instead of DL methods.

To address the limitations posed by small labeled datasets, self-supervised learning (SSL) has emerged as a promising alternative. SSL enables models to learn meaningful representations from unlabeled data through pretext tasks such as contrastive learning or temporal prediction, which are later fine-tuned on labeled data. While SSL reduces the dependence on annotations, it introduces its own complexities. Designing effective pretext tasks that align well with downstream HAR objectives is non-trivial, and SSL models may still require careful tuning and large unlabeled corpora to outperform simpler baselines.

In this project, we instead focus our efforts on investigating the extent to which supervised DL architectures can be effectively applied under limited data conditions. Our objective is to evaluate the performance and generalization capabilities of various DL models in this constrained setting.

3 Using the Github Repository

The codebase was developed and tested using Python 3.12.8. Follow these steps to set up and use the repository:

3.1 Description of the Key Directories and Files:

- **models/**: Model parameters (.pth files) will be saved to this directory by default.
- **notebooks/**: Contains Jupyter notebooks (‘.ipynb‘ files) for exploratory data analysis, model training, and a demo notebook if you wish to skip training and want to try out our trained models.
- **results/**: Training and validation results will be saved to this directory by default. Also saves the results of the trained model (based on last epoch parameters) on the test set.

- **results_demo/**: Results on test set from 'inference_demo.ipynb' are saved here. Currently contains the test results of the model with the best F1 during training for each architecture.
- **src/**: Contains core Python files for data processing and model training:
  - `data_preparation.py`: Code for downloading and processing the dataset.
  - `models.py`: Code for deep learning model classes.
  - `train_eval.py`: Code for training and evaluation of models.
  - `train_eval.py`: Helper functions.

3.2 Usage Instructions

Follow these steps to set up and use the repository:

1. **Clone the repository:**

```
git clone https://github.com/thenoobychocobo/human-activity-recognition-70plus.git
cd human-activity-recognition-70plus
```

2. **Install dependencies:**

```
pip install -r requirements.txt
```

3. **Run the notebooks:** Open and run the Jupyter notebooks in the `notebooks/` folder for exploratory data analysis (`eda.ipynb`), model training (`model_train.ipynb`), or inference demo (`inference_demo.ipynb`). You can skip training if you wish as we already provide trained model parameters in `models` folder.

4 Dataset

4.1 HARTH

The HARTH dataset [1, 3] features recordings collected from 22 participants, each wearing two Axivity AX3 triaxial accelerometers during approximately two hours of free-living activities. One device was mounted on the participant's right thigh and the other on the lower back, with acceleration data sampled at 50 Hz. Ground truth activity labels were obtained through manual frame-by-frame annotation of video recordings captured from a chest-mounted camera worn by the participants.

The recorded data for each participant (subject) is provided in a corresponding `.csv` file, each containing the following columns:

- `timestamp`: date and time of recorded sample
- `back_x`: acceleration of back sensor in x-direction (down) in units of g
- `back_y`: acceleration of back sensor in y-direction (left) in units of g
- `back_z`: acceleration of back sensor in z-direction (forward) in units of g
- `thigh_x`: acceleration of thigh sensor in x-direction (down) in units of g
- `thigh_y`: acceleration of thigh sensor in y-direction (right) in units of g
- `thigh_z`: acceleration of thigh sensor in z-direction (backward) in units of g
- `label`: annotated activity code

The dataset contains the following **annotated activities** with their corresponding label codes:

- (1) Walking
- (2) Running
- (3) Shuffling
- (4) Stairs (ascending)
- (5) Stairs (descending)
- (6) Standing
- (7) Sitting
- (8) Lying
- (13) Cycling (sitting)
- (14) Cycling (standing)
- (130) Transport (sitting)
- (140) Transport (standing)

The HARTH dataset UCI repository page refers to labels 130 and 140 as "Cycling (sit, inactive)" and "Cycling (stand, inactive)". This is somewhat confusing, as it is not clear what activity "inactive cycling" could refer to. This is clarified in the HARTH paper, which refers to labels 130 and 140 as "Transport (sitting)" and "Transport (standing)" respectively. These are defined as the activity of sitting or standing in a bus, car, or train. This work will adopt this more intuitive label naming convention.

4.2 HAR70+

The HAR70+ dataset [6] contains accelerometer recordings from 18 older adults, aged between 70 and 95 years, who participated in a semi-structured free-living protocol lasting approximately 40 minutes. Participants wore two Axivity AX3 triaxial accelerometers: one placed on the right thigh and the other on the lower back. Data were sampled at 50 Hz, and five participants used walking aids during the recording sessions. Activity labels were assigned through manual frame-by-frame review of video footage collected via a chest-mounted camera.

Similarly to the HARTH dataset, the recorded data for each participant (subject) is provided in a corresponding `.csv` file and contains the same columns as before.

The dataset includes the following **annotated activities**:

- (1) Walking
- (3) Shuffling
- (4) Stairs (ascending)
- (5) Stairs (descending)
- (6) Standing
- (7) Sitting
- (8) Lying

The HAR70+ dataset adopts the same labeling scheme as the HARTH dataset. Notably, labels 2, 13, 14, 130, 140 are absent because none of the participants performed running, cycling, and inactive transportation activities during data collection.

4.3 Rationale for Dataset Selection

The HARTH and HAR70+ datasets were selected to provide a strong and realistic foundation for training and evaluating human activity recognition (HAR) models in free-living settings.

The HARTH dataset offers several key advantages. It contains high-quality acceleration data collected from 22 participants during their regular daily activities, ensuring natural behavior rather than scripted movements. Sensor placements were standardized (right thigh and lower back), and activities were annotated by two experts, resulting in professionally labeled data. Importantly, HARTH fills a gap in publicly available free-living HAR datasets by combining naturalistic recording conditions, fixed sensor placement, and expert annotations. These characteristics make HARTH a valuable resource for developing and benchmarking HAR models in real-world conditions.

The HAR70+ dataset complements HARTH by focusing specifically on an underrepresented population in HAR research: older adults aged 70–95 years,

including individuals who use walking aids. Since most existing HAR datasets are based on young, healthy participants and collected in controlled laboratory settings, they may not generalize well to older adults in real-life conditions. HAR70+ addresses this limitation by providing free-living accelerometer data annotated using video recordings—the gold standard for ground truth labeling—thus capturing the full spectrum from fit to frail older adults. Incorporating HAR70+ allows for the development of models that are better aligned with the needs of aging populations, which is critical for applications such as health monitoring and fall prevention.

Together, HARTH and HAR70+ enable training and evaluation of HAR models across different age groups and physical conditions, with a strong emphasis on ecological validity and robust labeling.

4.4 Dataset Analysis

A detailed code-walkthrough of the dataset exploration can be found in `notebooks/eda.ipynb`. A summary of key findings is as follows:

- The combined datasets include 40 subjects: 22 from HARTH and 18 from HAR70+. However, the HARTH recordings are significantly longer, with approximately 2 hours of data per subject compared to around 40 minutes per subject in HAR70+.
- A discrepancy was identified in the HARTH dataset obtained from the UCI repository: `S006.csv` was recorded at a 100 Hz sampling rate, while all other files were sampled at 50 Hz. According to the HARTH paper, some subjects were originally recorded at 100 Hz but were subsequently downsampled to 50 Hz. It appears that `S006.csv` was inadvertently left undownsampled.
- The average time interval between consecutive samples is approximately 0.02 seconds (20 milliseconds), consistent with the reported 50 Hz sampling rate for both datasets (where $1/50 = 0.02$ seconds). However, occasional larger gaps (e.g., up to 59.62 seconds in `S013.csv` and 4.861 seconds in `501.csv`) were observed, likely due to recording pauses or missing data.
- The datasets exhibit substantial class imbalance. Activities such as standing and walking are heavily overrepresented, whereas activities like ascending stairs or cycling occur far less frequently.

4.5 Dataset Preprocessing

We developed a structured preprocessing pipeline to ensure temporal consistency and data quality. The relevant code can be found in `src/data_preparation.py`.

4.5.1 Fixing Sampling Rate Discrepancies

As discussed previously, `S006.csv` in the HARTH dataset was recorded at 100 Hz, unlike the other files recorded at 50 Hz. To ensure consistency across all subjects, we manually downsampled `S006.csv` by a factor of 2.

4.5.2 Label Remapping

The original activity coding schemes in both datasets were not directly compatible with PyTorch, which requires class labels to be consecutive integers in the range `[0, num_classes-1]`. Therefore, we remapped the activity labels to the following scheme:

- (0) Walking
- (1) Running
- (2) Shuffling
- (3) Stairs (ascending)
- (4) Stairs (descending)
- (5) Standing
- (6) Sitting
- (7) Lying
- (8) Cycling (sitting)
- (9) Cycling (standing)
- (10) Transport (sitting)
- (11) Transport (standing)

4.5.3 Sequence Construction

The accelerometer data in both datasets is sampled at 50 Hz, corresponding to a time interval of 0.02 seconds between consecutive measurements. To transform the continuous time series into a format suitable for supervised learning, we applied a sliding window approach, segmenting the data into overlapping fixed-length sample sequences. Each window consists of a predefined number of consecutive time steps, with a configurable stride controlling the degree of overlap between adjacent sequences.

For our experiments, we set the window size to 250 time steps (equivalent to 5 seconds of data) and the stride to 125 time steps, resulting in a 50% overlap between consecutive windows. The choice of overlapping windows is beneficial as it increases the number of training samples and introduces slight variations between them, promoting model robustness and improving generalization.

However, the datasets occasionally exhibit larger-than-expected gaps between consecutive timestamps, likely due to interruptions or pauses in recording. Allowing sliding windows to cross these gaps would create sequences that incorrectly assume temporal continuity. To address this, we identified 'breakpoints'—timestamps where the time difference between two consecutive samples exceeded a defined threshold (set to 0.05 seconds in our experiments).

The data was partitioned into continuous segments based on these breakpoints. Sliding windows were then applied independently within each segment, ensuring that no sample sequence spanned a temporal discontinuity. This guarantees the temporal integrity of each sequence.

4.5.4 Sequence Labeling

Initially, we assigned each sequence the activity label corresponding to its final time step, under the assumption that recent history would be predictive of ongoing activity. However, early experiments revealed issues with this approach. In some cases, sequences contained predominantly one activity, but the final time step captured a different activity, leading to noisy and inconsistent labeling.

This problem is further exacerbated by the subjective nature of human annotation: the exact transition points between activities are often ambiguous, especially in free-living datasets, where transitions can be gradual rather than discrete.

To mitigate this, we adopted a majority-vote strategy: each sequence is labeled according to the most frequent activity class among all its time steps. Majority labeling stabilizes the assigned class, makes the labels more robust against brief transitional noise, and better aligns with the goal of human activity recognition—which is typically to recognize sustained activities rather than momentary transitions. HAR models are expected to predict a consistent activity over an entire sequence, rather than react sensitively to minor frame-by-frame fluctuations.

4.5.5 Subject-Wise Data Partitioning

To promote generalization and reduce the risk of overfitting to individual characteristics, we partitioned the compiled dataset at the participant (subject) level into training, validation, and test splits. Each split contains disjoint sets of subjects (`csv` files), ensuring that no subject appears in more than one subset. This prevents the model from exploiting subject-specific patterns that could artificially inflate performance metrics.

Subject-wise partitioning is also an effective strategy for avoiding temporal leakage. Due to the 50% overlap introduced during sliding window segmentation, adjacent sequences within a subject share many time steps. If standard random splitting were used, it would be likely that overlapping sequences—containing nearly identical temporal information—would be split across the training and validation/test sets. This would compromise the validity of

evaluation results. Subject-wise splitting avoids this problem entirely, as no time steps are shared between splits.

We initially considered using leave-one-subject-out (LOSO) cross-validation, where in each iteration the model is trained on $(N-1)$ subjects and tested on the held-out subject, repeating this process N times. LOSO offers a rigorous evaluation of generalization across subjects. However, due to time constraints and the significant computational burden of training and evaluating a model $N = 40$ times (once per subject), we opted for a fixed subject-wise train-val-test split in this work.

4.5.6 Feature Normalization

To account for signal variability caused by inter-individual differences and sensor inconsistencies, we applied feature-wise z-score normalization to the input signals. Specifically, we computed the mean and standard deviation for each feature dimension using only the training set, and normalized the training, validation, and test sets using these training statistics.

Normalizing the features to have zero mean and unit variance is a standard practice that stabilizes and accelerates model training. In neural networks, normalization helps by ensuring that input features are on comparable scales, which improves the conditioning of the optimization problem, leads to smoother loss surfaces, and facilitates more efficient gradient-based learning. Without normalization, features with larger numerical ranges can dominate the gradient updates, making training unstable or slow.

Importantly, we always use the training set's mean and standard deviation to normalize the validation and test sets. This avoids introducing distributional shifts, as the model has been trained on inputs standardized according to the training distribution.

5 Model Architectures

We experimented with seven different deep learning model architectures for human activity recognition. All models were implemented in PyTorch, and their documented implementations are available in `src/models.py`.

5.1 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a natural choice for time series and sequential data, as they process input sequences one step at a time while maintaining a hidden state that captures information from previous time steps. This enables RNNs to model temporal dependencies, making them well-suited for tasks like human activity recognition (HAR), where patterns evolve over time. However, traditional RNNs often struggle with learning long-term dependencies due to issues like vanishing or exploding gradients. To address this, more advanced variants like GRUs and LSTMs are commonly used.

For our RNNs, we pass the hidden state of the last time step to a linear processing layer and softmax function for multi-class classification.

Unidirectional Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks introduce dedicated gating mechanisms (input, forget, and output gates) to more effectively regulate the flow of information and combat the vanishing gradient problem. By maintaining an explicit memory cell in addition to the hidden state, LSTMs can learn to remember or forget information over longer time horizons. We use a unidirectional LSTM architecture that processes sequences sequentially from start to end, relying purely on past context.

Bidirectional Stacked Long Short-Term Memory (BiLSTM) Networks

To enhance the model's ability to capture dependencies from both past and future contexts, we implemented a bidirectional LSTM (BiLSTM). In this architecture, two LSTMs are run in parallel: one processing the sequence in forward time and the other in reverse. Their outputs are concatenated at each time step, allowing it to capture past, present, and future context at each point in the input time series, providing a broader temporal view that can improve performance in accelerometer-based HAR. Additionally, we stack multiple LSTM layers to allow the model to build hierarchical temporal representations, enabling more complex pattern extraction across different time scales. BiLSTMs have achieved good results in HAR in previous works [8].

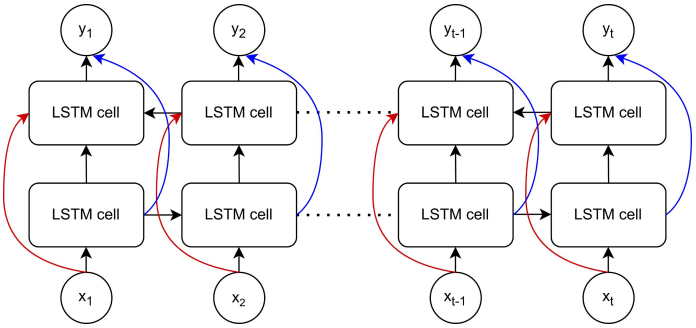


Figure 1: Single Layer of BiLSTM

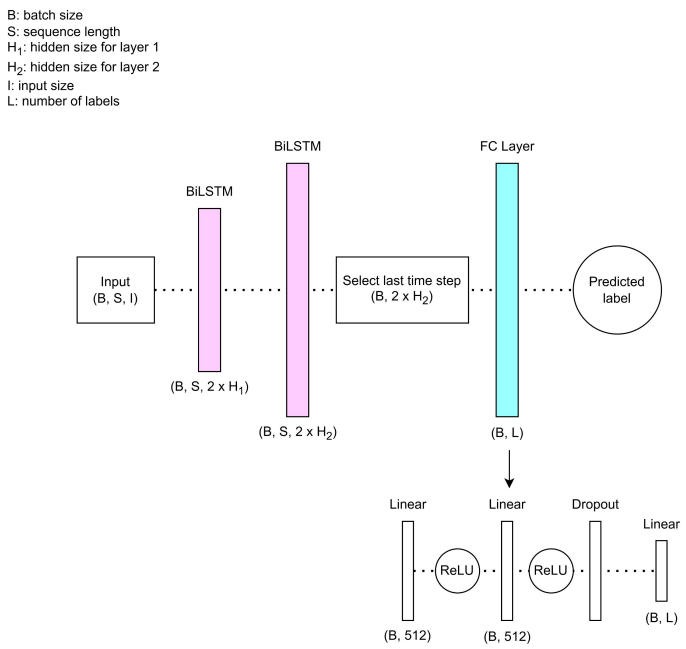


Figure 2: Stacked BiLSTM implementation

Unidirectional Gated Recurrent Unit (GRU) Networks

Gated Recurrent Units (GRUs) simplify the architecture of traditional LSTMs by combining the forget and input gates into a single update gate and merging the hidden and cell states. This leads to fewer parameters and faster training, while still allowing the model to capture dependencies over moderate temporal ranges. GRUs may also capture short term dependencies better than LSTMs. In our setup, we employ a standard unidirectional GRU, where information flows from past to future without any backward context.

5.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models widely used in time series and sequence data analysis. The core principle of CNNs is weight sharing, where a small set of learnable weights, known as a filter or kernel, is applied across the input data via a convolution operation. This allows the model to efficiently detect local patterns and features, such as sudden changes, trends, or periodicities in the input signal.

CNNs differ from fully connected networks in that they use convolutional layers to extract hierarchical features. Early layers capture low-level features, such as edges or temporal spikes, while deeper layers build increasingly abstract representations. By stacking convolutional layers, CNNs can learn to identify both fine-grained local dependencies and higher-level patterns within the data.

While CNNs are not inherently designed to model long-range temporal dependencies (as they do not rely on recurrence), their ability to capture local patterns makes them effective for many time series classification tasks, including human activity recognition (HAR). CNNs excel at handling raw input sequences and transforming them into more compact, informative feature maps that can then be processed by fully connected layers to make predictions.

5.2.1 1D Convolutional Neural Networks (1D CNNs)

In its simplest form, a 1D CNN applies a single convolutional filter over a sequence of input data, extracting local patterns over a fixed-length window of time. Typically, 1D CNNs are used with raw time series data, where the filter slides across the input sequence to identify local features such as edges or spikes.

The convolutional layer is followed by a non-linear activation function, typically the Rectified Linear Unit (ReLU), which introduces non-linearity to the model. After several convolutional layers, pooling layers (e.g., max pooling) are often applied to reduce the dimensionality of the feature maps, helping to highlight the most prominent patterns. Finally, the output is flattened and passed through fully connected layers to generate the final prediction.

In the context of HAR, 1D CNNs are effective at detecting short-term patterns, such as quick body movements, gestures, or changes in posture. While they do not capture long-range dependencies, their ability to extract localized features makes them suitable for tasks where short-term motion cues are critical.

5.2.2 Inception-based 1D Convolutional Neural Networks (Inception CNNs)

To extend the capabilities of basic 1D CNNs, we also explored a multi-resolution approach inspired by the Inception modules proposed by Szegedy et al. [5]. Instead of using a single kernel size within a convolutional layer, the Inception module applies multiple convolutional filters with different kernel sizes in parallel. This allows the network to capture features at multiple temporal scales simultaneously.

For time series data, different kernel sizes can focus on different levels of temporal granularity. Smaller kernels detect fine-grained, local patterns, while larger kernels capture broader, more global trends. The outputs from each kernel are concatenated to form a richer feature map, which combines information across different scales.

We hypothesize that Inception CNNs should be useful for HAR tasks because they can simultaneously model both quick, subtle movements (captured by smaller kernels) and slower, sustained motions (captured by larger kernels). This multi-resolution approach improves the model's ability to learn complex temporal dynamics by integrating diverse levels of information in a single layer.

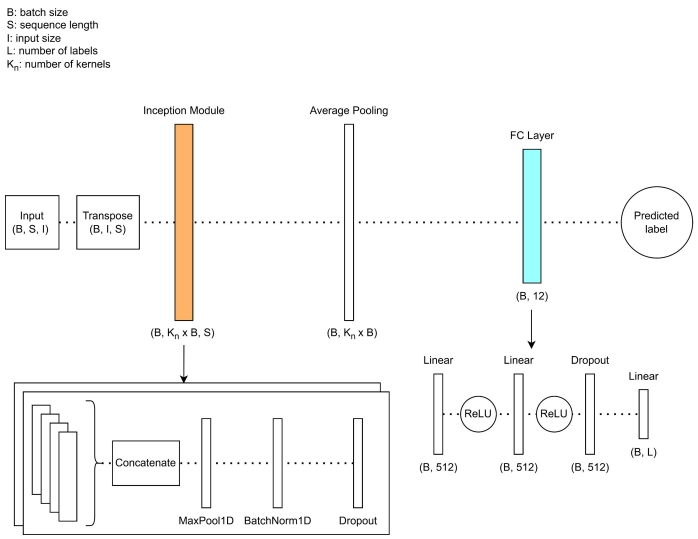


Figure 3: Inception-based 1D CNN

5.3 Transformer-Based Models

While RNNs process sequences sequentially, Transformers operate entirely through self-attention mechanisms, allowing them to model temporal relationships between any two time steps regardless of their distance. This architecture enables efficient parallelization during training and better captures long-range dependencies without relying on recurrence. Given their success across a wide range of sequence modeling tasks, including natural language processing and time series analysis, we explored Transformer-based architectures for human activity recognition (HAR). Since our input sequences for training are of a decent length (250), we are hoping that Transformers have an edge in capturing long-range dependencies and contextual information better than other architectures.

Since HAR is formulated here as a many-to-one task—taking a sequence of time steps as input and producing a single activity label—we do not require a decoder component, as there is no need to generate an output sequence. Instead, the input sequence is encoded using self-attention to produce a rich, contextualized representation suitable for classification.

Base Transformer Encoder

We closely follow the original Transformer architecture proposed by Vaswani et al. [7]. The base model begins with a linear projection of the six input features into a higher-dimensional embedding space, followed by the addition of sinusoidal positional encodings to inject order information into the sequence. The embedded inputs are then processed by stacked Transformer encoder layers, each consisting of multi-head self-attention and position-wise feedforward

networks. After encoding, the resulting sequence embeddings are aggregated and passed through a final linear classification head. This setup enables the model to flexibly attend to relevant time steps across the sequence without the limitations of sequential processing.

Transformer Encoder with CNN Tokenizer and RoPE

We also developed an experimental variant of the Transformer architecture by introducing two key modifications:

- **CNN Tokenizer**: We were surprised that the CNNs did as well as they did during our early experiments and decided to take it further by integrating it into our Transformer architecure. Instead of directly applying a linear projection to the raw input features, we first process the input signal using lightweight 1D convolutional layers. These convolutional layers act as a "tokenizer", extracting local patterns and producing embeddings that are more structured and informative before being passed into the Transformer encoder. From our early experiments, we suspect that the accelerometer time series data exhibits strong local correlations—dependencies between nearby time steps—that convolutional filters are particularly well-suited to capture. Compared to a simple linear projection, CNN tokenizers can learn more localized, translation-invariant features, improving the richness of the initial embeddings and easing the burden on the Transformer's self-attention mechanism. This approach is inspired by similar ideas seen in recent time series Transformer models, where local feature extraction has been found beneficial for stabilizing training and improving performance. Additionally, the convolutional layers typically downsample the input sequence, reducing its length before Transformer processing. This downsampling is advantageous because the self-attention mechanism has $O(L^2)$ time and memory complexity with respect to sequence length L. By shortening the sequence, we significantly lower computational cost while preserving essential local temporal structures.

- **Rotary Positional Embeddings (RoPE)**: Instead of using traditional sinusoidal positional encodings, we employ Rotary Positional Embeddings (RoPE) [4] to encode positional information. In the original Transformer architecture, positional encodings are added to the input embeddings prior to the self-attention layers. By contrast, RoPE incorporates positional information directly into the attention mechanism by applying a rotation operation to the query and key vectors in a manner that encodes relative positional relationships. This allows the model to reason about relative distances between tokens natively, rather than depending solely on absolute position encodings. For structured signals like time series, where relative time differences are often more important than absolute timestamps, this approach can improve generalization and extrapolation performance. Additionally, because RoPE integrates position information

into the attention computation itself, it enables positional effects to naturally interact with attention scores, potentially leading to richer and more flexible modeling of temporal dependencies.

This experimental architecture aims to combine the local sensitivity of CNNs with the long-range dependency modeling capabilities of Transformers.

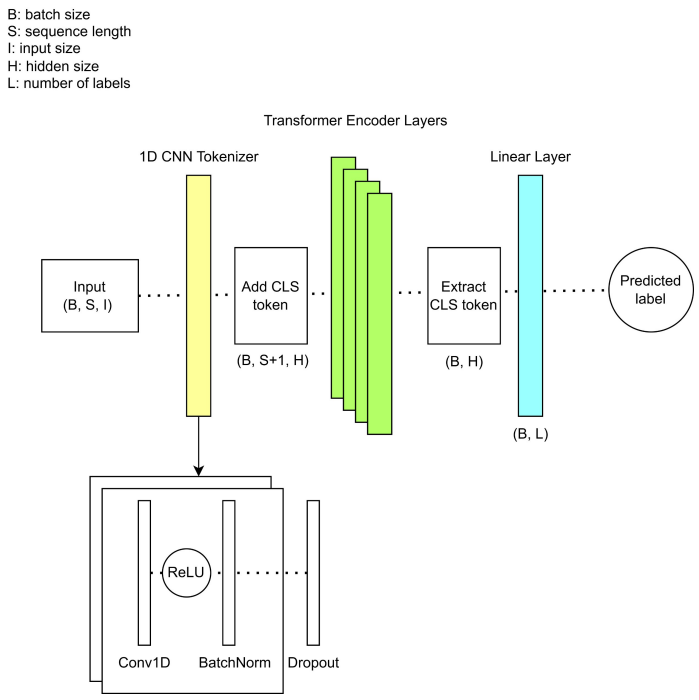


Figure 4: Tranformer Encoder with CNN Tokenizer and RoPE

5.3.1 Learnable Classification Token

Inspired by the BERT architecture [2], we prepend a special classification token (`[CLS]`) to the beginning of every input sequence in both of our Transformer-based models. This `[CLS]` token is implemented as a learnable embedding vector, initialized randomly and updated during training alongside the model parameters.

As the input sequence passes through the stacked Transformer encoder layers, the `[CLS]` token participates in self-attention interactions with all other time steps in the sequence. Through this process, the `[CLS]` token aggregates information from the entire sequence into its own embedding. After encoding, we extract the final representation of the `[CLS]` token and pass it through a linear classification head followed by a softmax layer to predict the output class.

The intuition behind this design is that the model learns to use the `[CLS]` token as a dedicated representation of the "summary" of the input sequence. During training, the network learns how to encode global, task-relevant information into the `[CLS]` embedding through attention-based interactions, enabling effective sequence-level classification without the need for manually pooling or aggregating features across time steps. In other words, it is used as the aggregate sequence representation for classification purposes.

The self-attention mechanism in the encoder is bidirectional (i.e., it is not causally masked), allowing the `[CLS]` token to attend to all other tokens in the sequence during encoding. Since the `[CLS]` token is always appended at the beginning of the input, its relative position remains fixed across all samples, ensuring consistency in how positional encodings are applied. This setup enables the `[CLS]` token to flexibly aggregate information from the entire sequence without being constrained by the ordering of time steps.

6 Experiments

6.1 Training Details

A code walkthrough for model training can be found in `notebooks/model_training.ipynb`.

For sample sequence extraction, we fixed the input sequence length to 250 time steps (corresponding to 5 seconds) and applied a stride of 125 time steps (50% overlap) during sliding window segmentation. Subjects were randomly shuffled and partitioned into training, validation, and test splits with a ratio of 32:4:4, ensuring subject-wise disjoint sets.

We used a batch size of 256, which we found to offer a good balance: it was sufficiently large to stabilize gradient updates and reduce stochasticity in optimization, large enough to exploit GPU parallelism, but small enough to allow reasonable training times given our compute constraints.

Due to limited time and computational resources, we conducted only basic hyperparameter tuning. We found the Adam optimizer with a learning rate of 0.001 to consistently yield the best performance. We briefly experimented with other optimizers available in PyTorch (e.g., SGD, RMSProp, AdamW), but observed only marginal differences. However, larger learning rates frequently resulted in unstable training, where both the training and validation losses exhibited periodic spikes. Since the training loss increased as well, this instability was not due to classic overfitting; instead, it is likely attributable to overly aggressive parameter updates, causing the optimizer to overshoot local minima.

To mitigate overfitting, we applied L2 regularization via the `weight_decay` parameter of the Adam optimizer, finding 1e-5 to be optimal. We also experimented with L1 regularization by manually adding a penalty term to the loss function. However, since the observed differences were marginal and implementation was simpler, we ultimately chose to proceed with L2 regularization for all experiments.

For the loss function, we used PyTorch's `torch.nn.CrossEntropyLoss`,

which internally applies the softmax operation to the logits before computing the negative log-likelihood. Each model configuration was trained for 30 epochs. We observed that validation performance typically plateaued around epoch 15, but continued training for 30 epochs to allow models the opportunity for late improvements. During training, we saved the model parameters every 5 epochs, and also saved the model parameters with the highest F1 score (our main metric) throughout training.

It is important to note that we determined the best hyperparameter values based solely on the model's performance on the validation set. The test set was strictly held out and only evaluated once all model configurations and hyperparameters were finalized. This practice is crucial to prevent data leakage, where information from the test set could inadvertently influence model development, leading to overly optimistic performance estimates. If hyperparameters were tuned directly using the test set, the model could overfit to the specific characteristics of the test data, thus invalidating claims about generalization. By reserving the test set purely for final evaluation and making sure that it remains unseen during model tuning, we ensure that the reported test metrics more accurately reflect the model's expected performance on truly unseen data.

6.2 Evaluation Metrics

We evaluated all models using five key metrics:

- **Accuracy (Micro Average)**: The overall proportion of correctly classified samples across all classes. However, due to significant class imbalance in the dataset, micro-average accuracy was often misleadingly high even with minimal training.
- **Accuracy (Macro Average)**: The unweighted average of accuracies computed independently for each class, treating all classes equally regardless of frequency. This provided a more balanced view of model performance.
- **Precision**: For each class, precision measures the proportion of true positive predictions out of all positive predictions made. High precision indicates few false positives.
- **Recall**: For each class, recall measures the proportion of true positive predictions out of all actual positive instances. High recall indicates few false negatives.
- **F1-Score**: The harmonic mean of precision and recall. F1-score balances both metrics and is particularly useful in imbalanced classification settings where optimizing for either precision or recall alone may be insufficient.

Unless otherwise stated, we report macro-averaged scores for precision, recall, and F1 to ensure equal treatment of all classes, regardless of their relative frequencies.

We chose F1 score as the main evaluation metric because it provides a balanced measure of model performance, especially in the presence of class imbalance. Unlike accuracy, which can be misleading when classes are unevenly distributed, F1 considers both precision and recall. As the harmonic mean of these two metrics, it ensures that the model's ability to correctly classify both positive and negative instances is accounted for, making it particularly suitable for tasks with skewed class distributions, such as human activity recognition.

6.3 Testing

From our (limited) hyperparameter tuning, we tested the seven models with the following configurations:

- `HarLSTM`: `hidden_size=30, num_layers=1, dropout_prob=0.0`
- `HarGRU`: `hidden_size=30, num_layers=1, dropout_prob=0.0`
- `HarBiLSTM`: `hidden_units=[32,64], dropout_prob=0.1`
- `HarCNN`: `num_kernels=64, kernel_sizes=[7,7], dropout_prob=0.1, pooling=False,` $\texttt{batch}_n ormalization = False$`HarInceptionCNN : num_kernels=64, num_layers=2, kernel_sizes=[3,5`
- `HarTransformer`: `hidden_size=32, dim_feedforward=None, num_layers=4, num_heads=4, dropout_prob=0.1, max_sequence_length=5000`
- `HarTransformerExperimental`: `hidden_size=32, dim_feedforward=None, num_layers=4, num_heads=4, dropout_prob=0.1, cnn_kernel_sizes=[5,3], cnn_stride=1, cnn_padding=1, max_sequence_length=5000`

The results of each model's predictions across the 12 labels are shown in Table 1 below.

Model Architecture	Micro Accuracy	Macro Accuracy	F1-score	Precision	Recall
BiLSTM	90.80	60.24	60.52	74.01	60.24
GRU	90.39	59.49	57.92	57.75	59.49
CNN	92.91	74.65	74.84	89.78	92.90
InceptionCNN	92.86	69.51	69.05	70.68	69.51
LSTM	88.29	46.53	45.73	4559	4654
Transformer	92.09	69.32	70.19	7449	69.33
ExperimentalTransformer	91.67	69.14	66.88	69.53	69.14

Table 1: Performance Metrics for Various Model Architectures over 12 labels. Find Confusion Matrix in the appendix

We are primarily focused on classifying physical activities, and upon analyzing the activity labels, it can be observed that certain labels represent similar activities. To simplify the task and reduce noise, we decided to merge certain labels. We follow the HARTH paper [3] and merged the labels "shuffling,"

"transport (standing)," and "standing" into a single label, "standing." Similarly, "sitting" and "transport (sitting)" were combined into "sitting." This merging process was implemented by summing the corresponding columns and rows in the confusion matrix, effectively reducing the number of labels from the original set to nine distinct physical activities. Label merging also has the effect of reducing class imbalance. Table 2 provides an overview of each model's average F1-score, precision, and recall, focusing on these nine merged activity labels. The performance of all models showed significant improvement across all metrics after this merging process.

Model Architecture	**Micro Accuracy**	**Macro Accuracy**	**F1-score**	**Precision**	**Recall**
BiLSTM	92.40	76.73	76.92	80.17	76.73
GRU	91.15	74.81	73.23	73.61	74.81
CNN	94.28	92.90	91.13	89.78	92.90
InceptionCNN	94.40	89.51	88.50	87.83	89.51
LSTM	89.19	60.63	60.53	61.51	60.63
Transformer	93.38	83.23	84.17	8563	83.23
ExperimentalTransformer	93.20	87.77	83.93	84.24	87.77

Table 2: Performance Metrics for Various Model Architectures over merged 9 labels. Find the confusion matrix in the Appendix

7 Discussion

Using the F1-score as our primary evaluation metric for the reasons discussed previously, our results indicate that the CNN model achieves the best overall performance. The confusion matrices (refer to Appendix below) offer further insight: while certain classes are predicted accurately, others consistently underperform. Notably, the poorly predicted classes tend to be those with fewer training examples, reinforcing the idea that class imbalance remains a significant challenge. This highlights the potential benefits of strategies aimed at mitigating class imbalance or developing models that are inherently more robust to such distributional skew.

We hypothesize that the strong performance of the CNN can be attributed to the nature of the data itself. Physical activity sensor data often exhibits strong local temporal dependencies—short-term patterns across consecutive time steps that are indicative of specific movements. Convolutional kernels are well-suited to capture such localized features, as they can learn translation-invariant detectors for characteristic patterns within a small receptive field. In contrast, models that rely on sequential processing (e.g., LSTM, GRU) may be less effective when the primary discriminative information is contained within short, fixed windows rather than long-term temporal structures. Moreover, the inductive bias of convolutions toward locality and parameter sharing likely contributes to better generalization, especially under limited data regimes.

All models exhibit difficulty distinguishing the shuffle activity. Interestingly, the confusion matrices reveal that while the Transformer struggles more than the CNN in recognizing ascending and descending stairs activities, it performs significantly better in identifying the Transport (sit) action. Moreover, each model appears to confuse different sets of labels, suggesting that differences in how they capture short-term versus long-term dependencies may influence their respective strengths and weaknesses.

8 Future Work and Possible Extensions

Due to time constraints, several avenues remain for future work to further enhance model performance and robustness.

First, although label merging helped mitigate class imbalance, a more principled approach would be to incorporate class weights directly into the loss function (weighted class updates), penalizing misclassifications of underrepresented classes more heavily. This would allow the model to balance learning across classes without relying solely on label consolidation.

Second, exploring semi-supervised learning (SSL) methods could be highly beneficial, particularly given the abundance of unlabeled sensor data. Techniques such as pseudo-labeling, self-training, or consistency regularization could enable the model to learn richer representations even when labeled data is limited.

Third, more extensive and systematic hyperparameter optimization could

reveal better configurations that improve generalization. Our hyperparameter tuning was relatively limited due to computational constraints. Additionally, employing a leave-one-subject-out (LOSO) cross-validation scheme could better evaluate model robustness across unseen subjects while making fuller use of the available data.

Finally, further architectural exploration remains promising. Future work could investigate hybrid CNN-Transformer stacks, pretraining with contrastive learning objectives, or dynamic tokenization strategies that group time steps into variable-length patches instead of using a fixed-size kernel for encoding.

References

[1] Kerstin Bach, Atle Kongsvold, Hilde Bårdstu, Ellen Marie Bardal, Håkon S. Kjærnli, Sverre Herland, Aleksej Logacjov, and Paul Jarle Mork. A Machine Learning Classifier for Detection of Physical Activity Types and Postures During Free-Living. *Journal for the Measurement of Physical Behaviour*, pages 1–8, December 2021.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Aleksej Logacjov, Kerstin Bach, Atle Kongsvold, Hilde Bremseth Bårdstu, and Paul Jarle Mork. HARTH: A Human Activity Recognition Dataset for Machine Learning. *Sensors*, 21(23):7853, November 2021.

[4] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

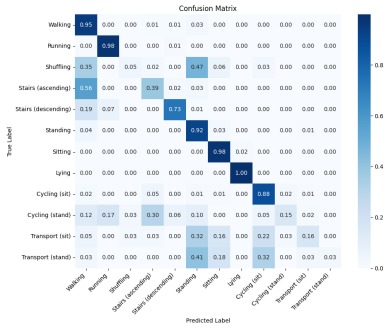
[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[6] Astrid Ustad, Aleksej Logacjov, Stine Øverengen Trollebø, Pernille Thingstad, Beatrix Vereijken, Kerstin Bach, and Nina Skjæret Maroni. Validation of an Activity Type Recognition Model Classifying Daily Physical Behavior in Older Adults: The HAR70+ Model. *Sensors*, 23(5):2368, January 2023.

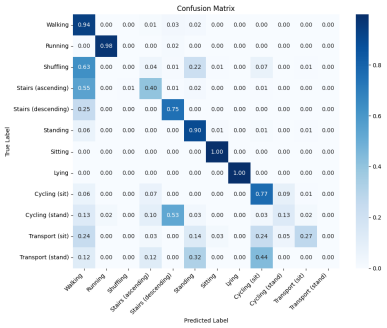
[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[8] Shilong Yu and Long Qin. Human activity recognition with smartphone inertial sensors using bidir-lstm networks. pages 219–224, 09 2018.

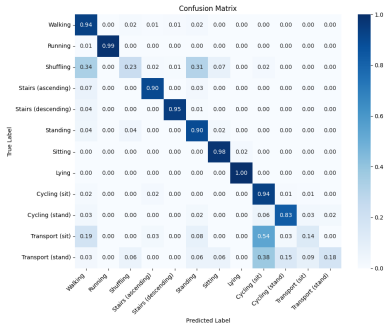
9 Appendix



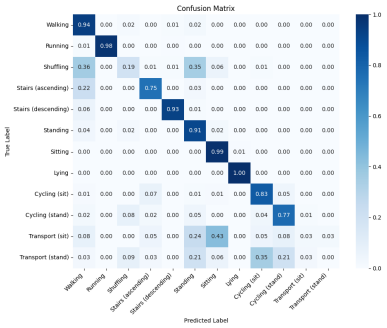
BiLSTM



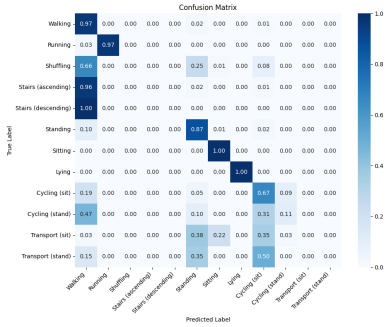
GRU



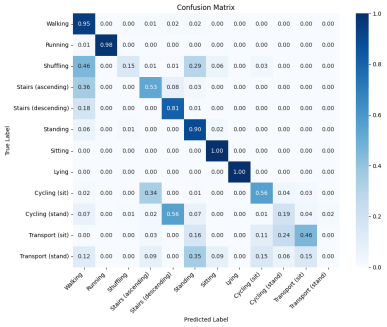
CNN



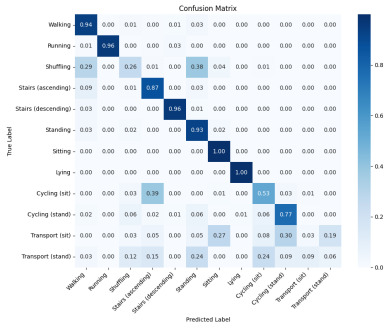
InceptionCNN



LSTM

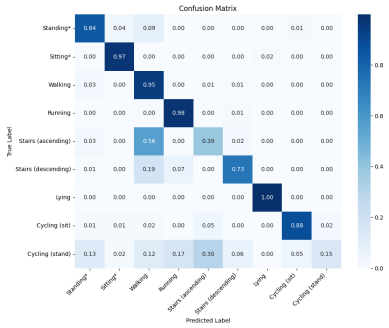


Transformer

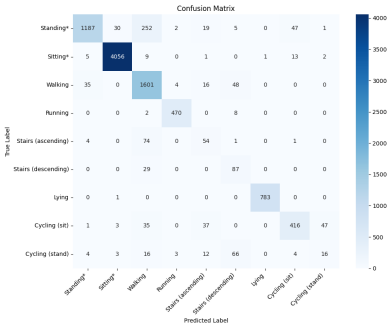


ExperimentalTransformer

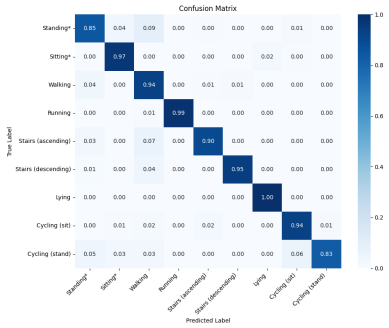
Table 3: Confusion Matrices for 12 labels



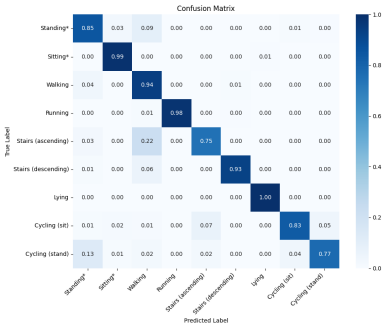
BiLSTM



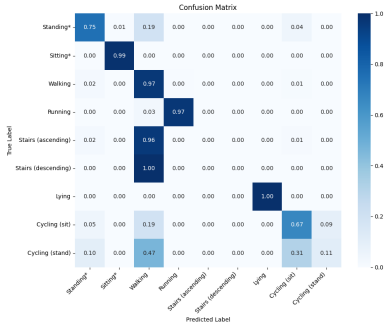
GRU



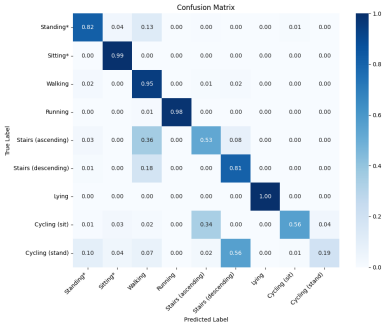
CNN



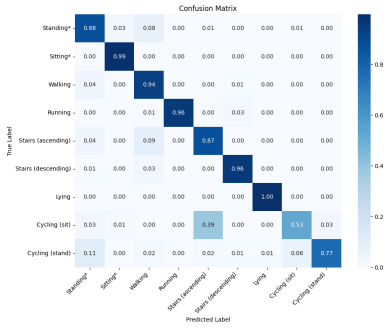
InceptionCNN



LSTM



Transformer



ExperimentalTransformer

Table 4: Confusion Matrices for 9 labels