UCLouvain

École polytechnique de Louvain

Human Activity Recognition using Deep Learning

Author : **Dorian Spilette**
Supervisor : **Charles Pecheur**
Readers : **Jean Vanderdonckt**, **Amaury Fierens**
Academic year 2023–2024
Master [120] in Computer Science and Engineering

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Charles Pecheur, whose guidance and unwavering support were instrumental in the completion of this thesis. His insightful feedback and rigorous standards have greatly shaped the quality of this work. I am also sincerely thankful to my friend, Charles Rongiones, for his valuable assistance in reviewing the report and offering thoughtful suggestions for improvement.

Disclaimer

This thesis has been enhanced with the assistance of generative AI tools, such as ChatGPT [20], to improve the flow and clarity of the sentences. While AI was used to aid in the refinement of the text, the content, ideas, and conclusions presented are entirely my own.

Contents

List of Figures

List of Abbreviations

FOG	**F**reezing **o**f **G**ait
HAR	**H**uman **A**ctivity **R**ecognition
DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
DBN	**D**eep **B**elief **N**etwork
GRU	**G**ated **R**ecurrent **U**nit
LSTM	**L**ong **S**hort-**T**erm **M**emory
EDA	**E**xploratory **D**ata **A**nalysis
AP	**A**verage **P**recision
mAP	**m**ean **A**verage **P**recision
MLP	**M**ulti-**L**ayer **P**erceptron
SGD	**S**tochastic **G**radient **D**escent
ReLU	**R**ectified **L**inear **U**nit
tanh	**T**angential **H**yperbolic
GPU	**G**raphics **P**rocessing **U**nit
TPU	**T**ensor **P**rocessing **U**nit
CSV	**C**omma-**S**eparated **V**alues

Chapter 1

Introduction

This master's thesis investigates the detection of Freezing of Gait (FOG) events using deep learning techniques. FOG, a debilitating symptom of Parkinson's disease, causes a sudden inability to walk despite the intention to do so. The primary challenge is to develop a machine learning model capable of detecting and classifying FOG episodes from sensor data into three categories: Start Hesitation, Turn, and Walking.

To address this, we utilize a dataset provided for a code competition hosted by Kaggle.com, where data scientists engage in various machine learning challenges. This specific competition, organized by the Michael J. Fox Foundation, involves 1,400 teams striving to detect FOG events in Parkinson's disease patients using wearable sensor data.

Manual detection of FOG episodes via video analysis, while accurate, is time-consuming and not scalable. Leveraging wearable sensors combined with machine learning offers a promising alternative by automating FOG detection. However, the limited availability of current datasets affects model generalizability. This thesis aims to develop a robust model to overcome these challenges.

The primary objective of this thesis is to develop a novel machine learning model for FOG detection using time-series data from wearable sensors, designed to improve accuracy in detecting and classifying FOG episodes. After development, the model will be tested and validated using competition data and further assessed for its adaptability to other similar datasets.

The approach begins with a literature review and analysis of existing solutions on Kaggle to identify effective models for time series problems. Following this, an exploratory data analysis (EDA) informs the modeling process. A novel neural network model tailored for FOG detection will be developed, optimized, and validated using the competition dataset. The model's performance will be compared with existing solutions and tested on an additional dataset to evaluate its generalizability.

This thesis contributes to the field by proposing a novel neural network model for FOG detection, providing a comprehensive comparison of existing models, and

performing an empirical analysis of the new implementation. Key contributions include the development of a new model, a detailed survey of current solutions, and validation of the model's effectiveness and adaptability.

The remainder of this document is structured as follows: the next chapter delves into the specific context of this master thesis, including Freezing of Gait, Human Activity Recognition and Kaggle. Following that, two chapters provide an in-depth review of neural network theories pertinent to FOG detection and analyze existing models and solutions from the competition. Subsequently, the thesis will define the specific problem of FOG detection, outline the challenges, and present the objectives behind the chosen approach.

The following chapters will detail the design, implementation, training, and validation of the proposed neural network model. After summarizing the findings and discussing the implications of the results, the final chapter will propose future research directions to further enhance FOG detection capabilities.

Chapter 2

Context

2.1 Freezing of Gait

The goal of this competition is to detect Freezing of Gait (FOG) events in time-series datasets using wearable sensor data. "Freezing of gait (FOG) is defined as a brief, episodic absence or marked reduction of forward progression of the feet despite the intention to walk. It is one of the most debilitating motor symptoms in patients with Parkinson's disease (PD) as it may lead to falls and a loss of independence" [10].

Despite various theories proposed by researchers to explain the occurrence and triggers of FOG, its underlying causes remain unclear [8]. Objective and accurate quantification of FOG episodes is crucial for advancing the understanding and treatment of this condition. An effective machine learning model will enhance the ability of medical professionals to evaluate, monitor, and ultimately prevent FOG events more efficiently. This can lead to better-personalized treatment plans and preventive measures, thereby improving the quality of life for those suffering from this debilitating symptom of Parkinson's disease.

2.2 Human Activity Recognition

The detection of Freezing of Gait events using wearable sensors is a specific application within the broader field of Human Activity Recognition (HAR). HAR involves using data from wearable sensors to identify and classify human activities, such as walking, running, sitting, and more. In the context of this thesis, the focus is on gait analysis to detect FOG episodes, a critical task for improving the management of Parkinson's disease.

Advancements in wearable technology and machine learning have significantly enhanced HAR capabilities [38]. By leveraging data from accelerometers, gyroscopes, and other sensors, sophisticated models can be developed to recognize complex activity patterns. The success of HAR applications hinges on accurately processing and interpreting time-series data, making it a vital area of research for improving health monitoring and intervention strategies.

2.3 Kaggle

This master's thesis utilizes resources provided by **Kaggle.com**, a platform that hosts machine learning competitions proposed by various organizations, including companies, foundations, and other entities. Each competition presents a specific problem and supplies datasets for participants to analyze, inviting data scientists worldwide to form teams and compete for the best solutions.

Each Kaggle competition runs for a designated period, during which teams strive to achieve the highest rankings on the leaderboard. After the competition closes, the datasets and competition details remain available for educational purposes, enabling participants to continue honing their skills and deepening their understanding of the problem.

The dataset central to this thesis comes from a competition organized by the *Michael J. Fox Foundation*, which ran from March 2023 to June 2023 [14]. Approximately 1,400 teams, each comprising up to five members, participated in this challenge, aiming to detect Freezing of Gait (FOG) events in Parkinson's disease patients using wearable sensor data. The results and rankings of these teams are recorded on the leaderboard, serving as a reference for future performance comparisons and benchmarking by other data scientists. It should be noted that while this thesis uses the competition dataset, it does not constitute official participation in the competition.

Chapter 3

Deep Learning for Time Series

This master's thesis discusses and implements various neural networks. For better comprehension, this chapter explains the mechanisms of deep learning and introduces the main models to be explored in subsequent sections.

3.1 Deep Neural Networks

3.1.1 Definition of Deep Neural Networks

Deep neural networks (DNNs) are machine learning algorithms inspired by the structure and function of the human brain [16]. As illustrated in Figure 3.1, they consist of multiple layers of interconnected nodes, known as neurons, that process input data to learn representations and patterns. These networks are termed 'deep' because they have multiple hidden layers between the input and output layers, allowing them to model complex relationships in data.

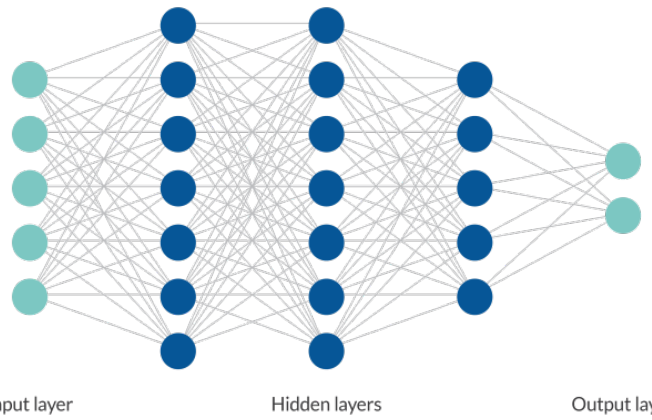


FIGURE 3.1: A simple deep neural network architecture representation. Image from [22]

The basic unit of a DNN is the artificial neuron, designed to mimic the behavior of biological neurons. Each neuron receives input signals, processes them, and produces an output. Neurons are organized into three primary types of layers:

- *The input layer* receives raw data, serving as the network's entry point.
- *The hidden layers* transform the data through various operations.
- *The output layer* produces the final prediction or classification.

3.1.2 Mechanisms of Deep Neural Networks

Deep neural networks operate through a series of steps involving forward propagation and backpropagation. This process enables the network to learn from data and improve its performance over time.

Forward Propagation

During forward propagation, input data passes through the network one layer at a time. Each neuron in a layer performs a weighted sum of its inputs, applies an activation function to introduce non-linearity, and passes the result to the next layer [3]. The operation of a single neuron can be described by the following equation:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.1}$$

where y is the output, x_i are the inputs, w_i are the weights, b is the bias, and f is the activation function.

The output layer produces the network's final output, which could be a classification label, a probability distribution, or a continuous value, depending on the task.

Backpropagation

Backpropagation is an algorithm used to train deep neural networks by minimizing prediction errors. It calculates the gradient of the loss function with respect to each weight in the network using the chain rule. The loss function L measures the difference between the actual target values y and the predicted output $\hat{y}$ [3]. For binary classification problems, the Binary Crossentropy Loss function is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.2}$$

During backpropagation, optimization algorithms like stochastic gradient descent (SGD) and Adam update the network's weights based on computed gradients. In deep networks, the vanishing gradient problem can arise, where gradients become very small and hinder the training process. This problem will be addressed in the implementations of the future models.

Activation Functions

Activation functions introduce non-linearities to the network, enabling the learning of complex patterns [9]. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). These functions are illustrated in Figure 3.2.

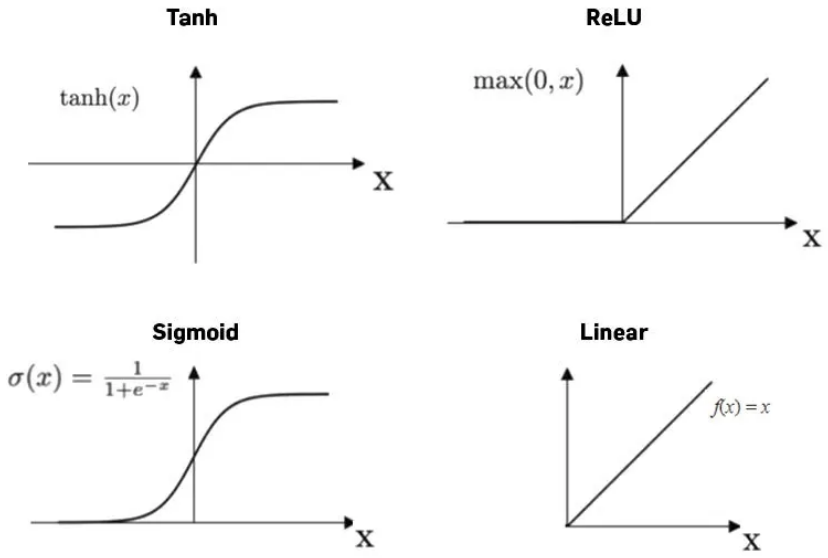


FIGURE 3.2: Common activation functions. Image from [30]

Future implementations will use ReLU for Convolutional Neural Networks and both Tanh and Sigmoid for Gated Recurrent Units, leveraging the strengths of each function for their respective network components.

Regularization Techniques

The training set is the portion of the data used to train the model while the test set is a separate portion of the data used to evaluate the model's performance on unseen data. Overfitting occurs when a model performs well on the training set but poorly on the test set. Regularization techniques are employed to prevent overfitting and improve the generalization of the network [29], ensuring that the model performs well not just on the training set but also on new, unseen data represented by the test set. Common techniques include:

- *Dropout*: Randomly deactivates a fraction of neurons during training, ensuring the network does not rely too heavily on any single neuron.
- *Batch Normalization*: This technique normalizes the outputs of each layer by adjusting and scaling them, maintaining their mean close to 0 and their standard deviation close to 1. During training, each mini-batch of data is normalized independently, introducing a bit of noise. This noise acts similarly to dropout,

preventing the model from becoming too reliant on specific patterns in the training data, thus enhancing its generalization to unseen data.

Given the complexity of generalizing models for the problem addressed in this thesis, both techniques will be extensively employed.

3.2 1D Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models widely used for their ability to automatically and adaptively learn spatial hierarchies of features from input data. CNNs are particularly well-known for their application in image processing tasks, where they operate on two-dimensional data [21]. This thesis focuses on a variant known as the 1D Convolutional Neural Network (1D CNN), which is more suited for analyzing sequential data organized in a single spatial dimension such as time series analysis, signal processing, and natural language processing. A 1D CNN network typically consists of two main building blocks: **Convolutional layers**, which apply convolution operations to capture local patterns in the data, followed by an **activation function** to introduce non-linearity, and **Pooling layers**, which downsample the data to reduce its dimensionality and computational complexity.

In the following sections of this master's thesis, the 1D Convolutional Neural Network will be one of the two neural network architectures central to our study.

3.2.1 The Convolutional Layer

In time series analysis, the 1D convolutional layer operates by sliding convolutional filters (also called kernels or windows) over the input data along the time dimension. The input to a 1D convolutional layer typically consists of sequences of data points, where each data point is represented by multiple features. For example, in our case, the input is a sequence of tri-axial accelerations from accelerometers. The convolutional layer applies filters that span across the time dimension and encompass all the features at each time step, as shown in Figure 3.3.

By convolving the filters with the input data, the convolutional layer extracts temporal patterns and relationships between features, creating a feature map that highlights important characteristics of the input sequence.

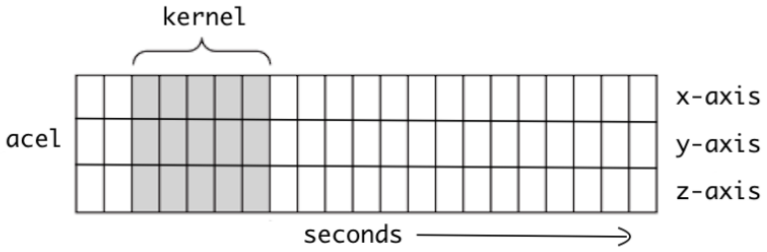


FIGURE 3.3: Input and kernel for a 1D convolution. Image from [33]

The kernel of the convolutional layer has a specified length, which determines its size. Each position in the kernel is associated with a weight that is randomly initialized and adjusted during training through backpropagation.

We define the input sequence as x, the output sequence as y, both of size n. The kernel, w, is of size k. We add the additional constraint that k must be odd and we define $k = 2p + 1$. For a kernel with $k = 5$, we would have $p = 2$. The convolution operation, $y[j] = (x * w)[j]$, can be explicitly defined as:

$$y[j] = \sum_{m=-p}^{p} x[j - m] \cdot w[m] \tag{3.3}$$

As illustrated in Figure 3.4, each convolution operation results in a single output. As the kernel slides over the sequence, it generates a feature map that highlights the features the kernel has learned to recognize. By using multiple kernels, the layer can produce different feature maps, each highlighting various features.

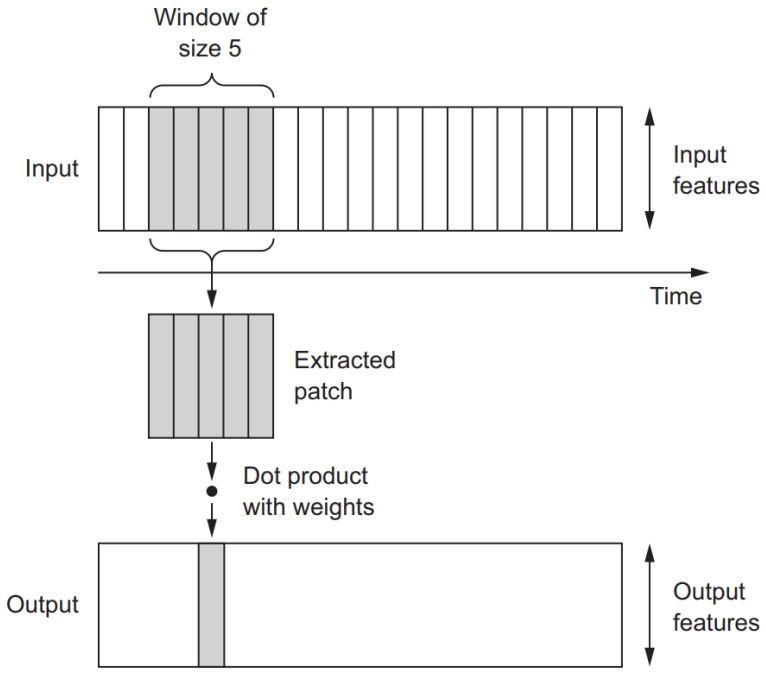


FIGURE 3.4: Feature map of a 1D convolutional layer. Image from [2]

3.2.2 Activation Function

In 1D CNNs, the most commonly used activation function after the convolution operation is the Rectified Linear Unit (ReLU).

$$\text{ReLU}(x) = \max(0, x) \tag{3.4}$$

ReLU is preferred due to its simplicity and efficiency. By converting all negative values to zero and keeping positive values unchanged, ReLU helps the model learn complex patterns without the vanishing gradient problem, which is often encountered with other activation functions like sigmoid or tanh. This makes the training of deep networks more efficient and faster. The ReLU activation function will be used by every Convolutional Neural Network discussed in this master thesis.

3.2.3 Pooling Layer

Pooling layers follow convolutional layers to down-sample the data, reducing its dimensionality and computational complexity. This helps control overfitting as the network becomes invariant to small changes in the position of features in the input. They are two common types of pooling:

Max Pooling:

The most common type of pooling in 1D CNNs is max pooling, which takes the maximum value within a specified window. This value captures the most prominent pattern detected by the kernel. Max pooling reduces the sensitivity of the output to minor variations in the input, thereby enhancing the model's ability to generalize. Figure 3.5 illustrates a max pooling operation of size two.

Average Pooling:

Another common type of pooling in 1D CNNs is average pooling, which calculates the average value within a specified window. This value represents the average pattern detected by the kernel over the window, smoothing the input and reducing the impact of outliers. Average pooling reduces the sensitivity of the output to minor variations in the input, thereby enhancing the model's ability to generalize.

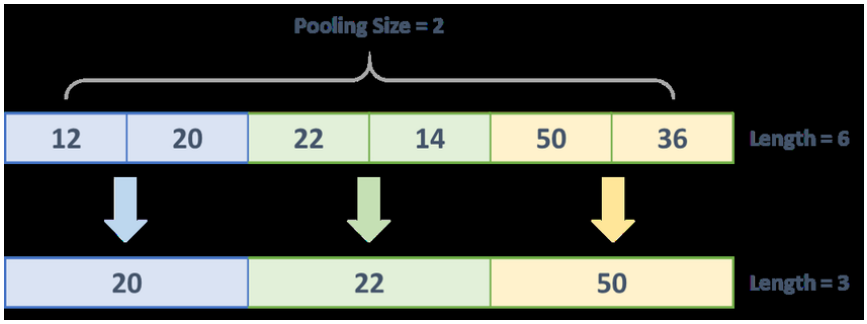


FIGURE 3.5: Max pooling operation. Image from [23]

3.3 Gated Recurrent Units

Gated Recurrent Units (GRUs) are the second neural network architecture that will be important for understanding the following sections. GRUs are a type of Recurrent Neural Network (RNN) designed to capture temporal dependencies in sequential data.

3.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks specialised for the processing of sequential data. Unlike feedforward neural networks, RNNs have connections that loop back, allowing information to persist over time by maintaining a hidden state. This makes them particularly suitable for tasks where the context from earlier in the sequence is relevant to processing later parts such as time series prediction, language modeling, and sequence classification.

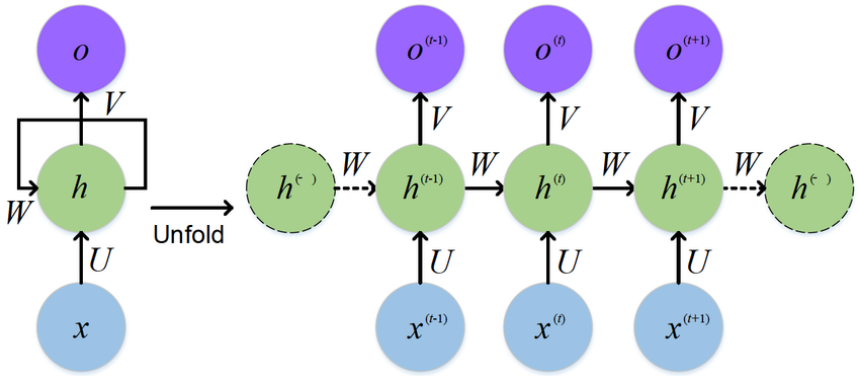


FIGURE 3.6: Architecture of a Recurrent Neural Network. Image from [7]

The main mechanism of RNNs is illustrated in Figure 3.6.

- The hidden state h_t at time step t is computed using the input x_t at that time step and the hidden state h_{t-1} from the previous time step.
- The output y_t at time step t is computed using the input x_t and the and the hidden state h_t at that time step.

3.3.2 Gated Recurrent Units

Gated Recurrent Units (GRUs) are an improved version of RNNs that address the vanishing gradient problem commonly encountered during training [4]. GRUs introduce gating mechanisms that control the flow of information, making it easier for the network to capture long-term dependencies. GRUs use two different gates to control and manage the flow of information, as illustrated in Figure 3.7.

The Update Gate : It determines how much of the previous hidden state to keep. It controls the ability of the model to "remember" past information and the influence of past knowledge on the current state.

The Reset Gate : It determines how much of the previous hidden state to forget. This allows the model to discard irrelevant information and adjust rapidly to what the model considers "important".

The governing equations for GRUs are :

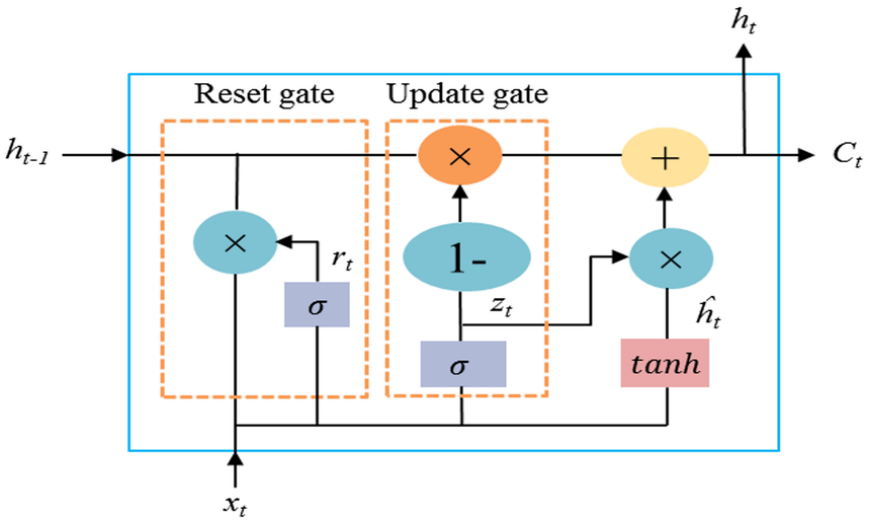


FIGURE 3.7: Architecture of a GRU cell. Image from [17]

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{3.5}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3.6}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{3.7}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{3.8}$$

where W_z, U_z, W_r, U_r, W_h, and U_h are weight matrices, and b_z, b_r, and b_h are bias vectors. σ is the sigmoid activation function, and tanh is the hyperbolic tangent activation function. The $\odot$ operator denotes the element-wise product. z_t and r_t represent the update and reset gates, respectively, while $\tilde{h}_t$ and h_t denote the candidate hidden state and final hidden state, respectively.

3.4 Advanced Layer Techniques

In addition to the core neural network architectures presented earlier, three advanced techniques will be used to enhance deep learning models: bi-directional wrappers, time-distributed wrappers and residual connections.

3.4.1 Bi-directional Layers/Wrappers

Bi-directional layers are a powerful extension of recurrent neural networks (RNNs) such as GRUs. In a standard RNN, the model processes the sequence of data in a single direction, typically from past to future. However, in many applications, future context can be just as important as past context. Bi-directional layers address this by processing the input data in both directions, capturing dependencies from both past and future contexts.

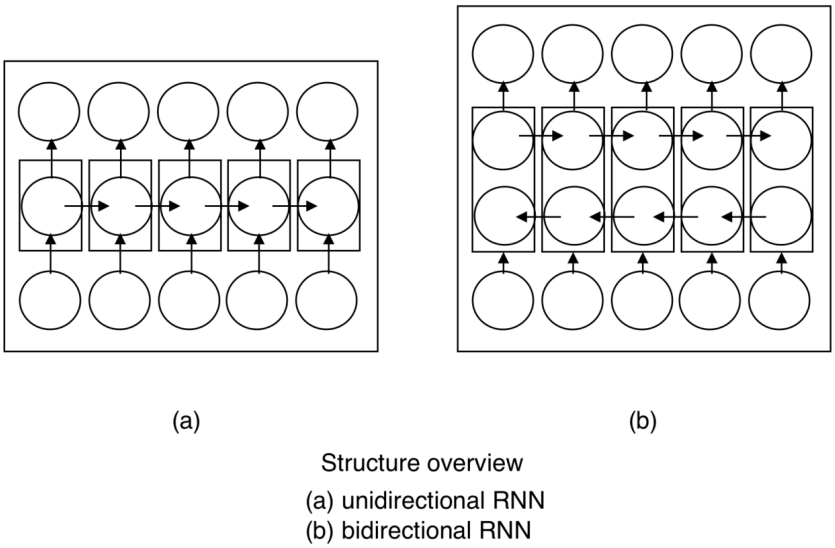


FIGURE 3.8: Structure overview of bi-directional RNNs . Image from [34]

A Bi-directional wrapper takes an RNN layer as input and returns a layer that runs the original RNN cell in both forward and backward directions. The resulting architecture is shown in Figure 3.8.

3.4.2 Time-distributed Layers/Wrappers

The time-distributed wrapper is used to apply the same layer or operation independently to every time step of a sequence. This is particularly useful in sequence-to-sequence models, where the same dense layer or convolutional layer needs to be applied to each time step individually.

For example, consider a video where each frame is treated as a time step, representing the video in the dataset as a sequence of frames. When processing the video, you might want to apply a 2D convolutional layer to each frame of the video independently. The time-distributed wrapper allows this by distributing the layer across the temporal dimension. This ensures that the same operation is applied to each frame, preserving the temporal structure of the data. This wrapper effectively enables the model to learn patterns across multiple dimensions of the data, whether spatial, temporal, or other structured sequences.

3.4.3 Residual Connections

Residual connections are a technique commonly used in neural networks, including Recurrent Neural Networks (RNNs), to facilitate the training of deeper models. The idea is to add the input of a layer directly to its output, bypassing the layer's transformation. The resulting architecture can be observed on Figure 3.9 where the input to a layer is directly added to its output, creating a shortcut path.

In a standard neural network layer, the output y is a function of the input x, typically represented as:

$$y = f(x) \tag{3.9}$$

In a residual connection, the output y is the sum of the layer's transformation and the input x:

$$y = f(x) + x \tag{3.10}$$

The main benefits of residual connections include [15]:

- **Easier Optimization**: By providing a direct path for the gradient to flow, residual connections make it easier to optimize deep networks. This helps in mitigating the vanishing gradient problem.
- **Learning Identity Mappings**: Residual connections enable the network to learn identity mappings (where the output is the same as the input) more easily.
- **Improved Performance**: Empirical evidence shows that networks with residual connections often achieve better performance on a variety of tasks compared to their plain counterparts.

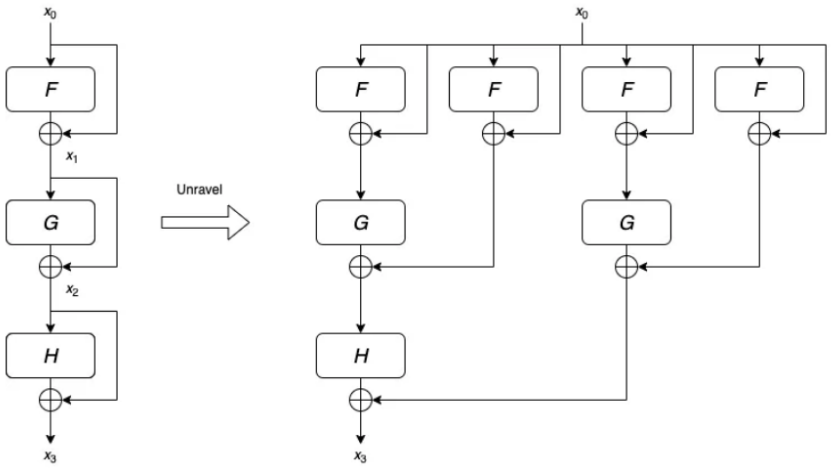


FIGURE 3.9: Unraveled view of Residual Connections. Image from [36]

Chapter 4

Related Work

Before going deeper, this thesis will analyze the state-of-the-art in Parkinson's detection and the broader scope of human activity recognition using wearable sensors. The first section will explore scientific articles about human activity recognition, while the second section will review previous solutions to the detection of FOG in the competition.

4.1 Advances in Human Activity Recognition

As FOG detection is a specific case of Human Activity Recognition (HAR) targeting gait analysis using wearable sensors, it is useful to first look at broader HAR advancements. Zhang et al. [38] provides an overview of advances in HAR using wearable sensors and deep learning, describing the different possible models and the challenges encountered.

Autoencoders are used for unsupervised feature extraction and noise reduction in sensor data. Convolutional Neural Networks (CNNs) are effective in extracting spatial features from time-series data. RNNs, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) variants, are ideal for capturing temporal patterns in activities. Deep Belief Networks (DBNs), though less common due to their training complexity, are utilized for hierarchical feature learning in HAR. Hybrid models, which combine CNNs, RNNs, and other architectures, leverage the strengths of each model type to improve recognition accuracy and robustness.

Among the difficulties in applying deep learning to HAR, a lack of quality data poses a significant challenge. Without sufficient and representative data, models may fail to generalize well to new, unseen data. Noise and artifacts in wearable sensor data can also impact model performance. Additionally, training deep models can be challenging due to issues like vanishing gradients. Efficient training strategies and balancing model complexity with available computational resources are key to overcoming these hurdles.

Ignatov [11] proposes a CNN architecture for local feature extraction from accelerometer data and investigate the impact of time series length on accuracy. The model demonstrates state-of-the-art performance at low computational cost.

Tan et al. [27] present a novel method for detecting heel strikes (HS) and toe-offs (TO) during the gait cycle using a modified LSTM network approach. The model, tested on the MAREA database, demonstrates superior performance compared to six state-of-the-art GED algorithms.

The transformer model, developed primarily for natural language processing and vision tasks, was introduced by Vaswani et al. in "Attention is All You Need" [32]. In an article published in March 2022, Dirgová Luptáková, Kubovčík, and Pospíchal [5] highlight an adaptation of the transformer model for time-series analysis of motion signals. The model achieved high accuracy and the authors suggest the expected future relevance of the transformer model for human activity recognition.

Among the different models presented by Zhang et al. [38] for HAR tasks, hybrid models, such as the CNN-GRU model, effectively leverage the advantages of both architectures. Given the outstanding performance of the individual models in the Kaggle competition (see 4.2), it is beneficial to explore the potential of the hybrid model further.

In their study, Dua, Singh, and Semwal [6] propose a multi-input CNN-GRU model for Human Activity Recognition using wearable sensors. This model combines the strengths of Convolutional Neural Networks for local feature extraction and Gated Recurrent Units for capturing long-term dependencies in time series data. "The accuracies obtained on UCI-HAR, WISDM, and PAMAP2 datasets are 96.20%, 97.21%, and 95.27% respectively" [6]. This end-to-end approach requires minimal pre-processing and does not rely on handcrafted features, demonstrating superior performance over existing models. The architecture proposed in this article can be seen on figure 4.1.

4.2 Different solutions from Kaggle

Due to the competition being closed, participants have been able to share their code and discuss about the development of their models. A lot of material is therefore available to observe the different solutions proposed by the Kaggle community. This section will examine the strengths and weaknesses of the most common solutions.

4.2.1 The Most Popular Solution: The CNN

Using a Convolutional Neural Network architecture as the main building block for the FOG detection model was the most researched solution in this competition. Nineteen different users who employed this model made their solutions public after the competition closed. This subsection will take a closer look at one of these solutions that reached twentieth place, developed by user *Moro* [18].

This solution uses a specific architecture presented in Figure 4.2 for its FOG event detection. The Multi-Layer Perceptron (MLP) is not discussed in this thesis and is simply used for multi-class label classification.

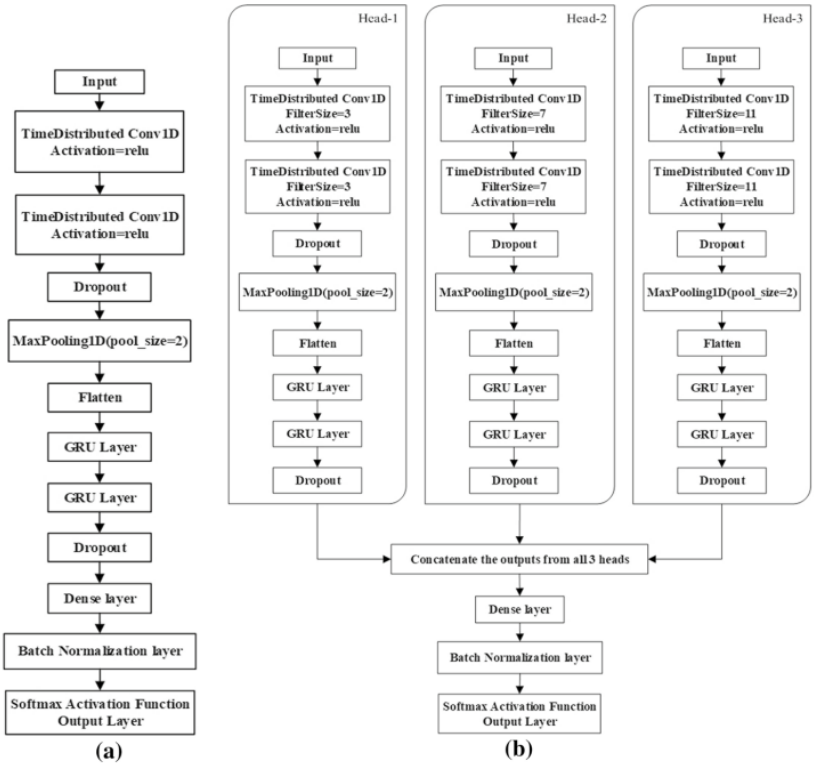


FIGURE 4.1: (a) Single-input CNN-GRU model for HAR. (b) Multi-input CNN-GRU model for HAR. Image from [6]

The strengths and weaknesses of using a CNN for event detection in time-series datasets will be examined [25], and how these translate into the model proposed by user *Moro* will be analyzed. These insights will be crucial for designing our own solution.

+ : Local Feature Detection

CNNs are adept at automatically learning spatial hierarchies of features, allowing them to capture short-term dependencies or patterns in time series data through convolutional filters.

- Convolutional layers process data in small windows, enabling the detection of local events over time.
- Moro's model used windows of size 32 and kernels ranging from size 3 to 5. Thirty-two timesteps represent one-third of a second in the dataset, emphasizing the ability to capture short-term dependencies and detect local events over time.

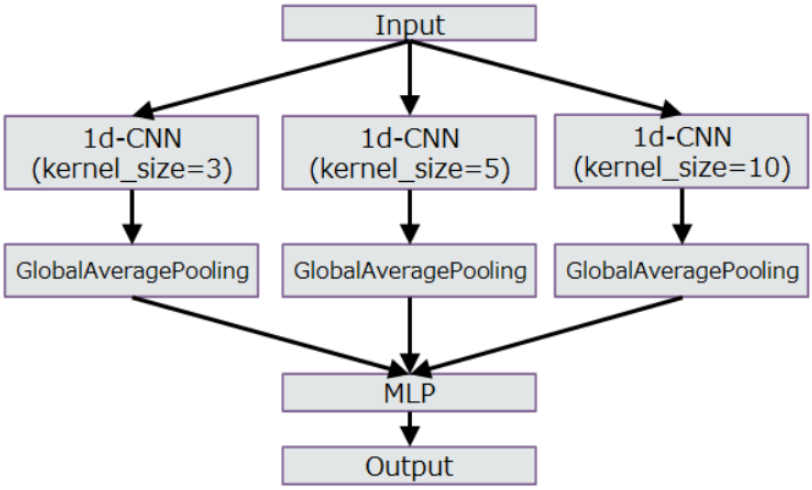


FIGURE 4.2: Architecture of the model used by user Moro. This model reached 20th place in the competition. Image from [18]

+ : Parameter Sharing

The convolutional layers in CNNs use the same set of weights (filters) across the entire time series.

- Parameter sharing allows the detection of the same feature at different points in time, ensuring invariance to the exact timing of the event.
- Moro's model uses different kernel sizes (3, 5, and 10) to capture various temporal patterns, ensuring that features are detected across different scales and timings.

+ : Temporal Invariance

Pooling layers in CNNs provide temporal invariance by down-sampling the feature maps.

- This helps the model focus on the most salient features, making it robust to slight variations in the timing or duration of events.
- In Moro's model, Global Average Pooling is used after each convolutional layer, summarizing the feature maps and enhancing robustness to timing variations.

+ : Efficient Computation

CNNs reduce the number of parameters through parameter sharing and local connectivity. Local connectivity means that each neuron in a convolutional layer is connected only to a small, localized region of the input data.

- This reduction in parameters leads to improved computational efficiency and a reduced risk of overfitting, which is particularly important when dealing with limited acceleration data.

- Moro's approach maintains computational efficiency thanks to the pooling layers and local connectivity. This enables the model to train for five epochs in approximately eleven minutes.

- : Limited Temporal Context Understanding

CNNs can struggle with understanding long-term dependencies and temporal relationships over extended periods.

- For events that depend on patterns spanning a long duration, CNNs might miss these signals because their receptive field is limited to shorter segments of the time series.

- Moro's solution, using windows spanning one-third of a second, is unable to detect long-term dependencies effectively.

- : Hyperparameter Sensitivity

CNNs have many hyperparameters that require careful tuning to achieve optimal performance.

- Selecting inappropriate hyperparameters can lead to suboptimal models, and the process of hyperparameter tuning can be computationally intensive and time-consuming.

- Moro's solution requires careful optimization of multiple correlated hyperparameters.

4.2.2 A Leading Solution: The Gated Recurrent Unit

The Gated Recurrent Unit (GRU) was used by different users and performed exceptionally well, allowing some solutions to reach the top ten of the leaderboard. User *Zinxhira* reached fourth place with a variant of the GRU using residual connections,

Similar to the earlier examination of the CNN, the advantages and weaknesses of the GRU for event detection in sequence data will be assessed. These insights will once again play an important role when designing our own solution

+ : Capturing Long-Term Dependencies

GRUs are adept at capturing long-term dependencies in sequence data through gating mechanisms that control the flow of information.

- GRUs use update and reset gates to manage information flow, helping retain relevant information over long sequences.
- Zinxhira's solution treated each individual time series as a sequence, no matter the length. This emphasizes the model's ability to capture long-term dependencies.

+ : Robustness to Noise

GRUs are less prone to overfitting and more robust to noise in the input data due to their gating mechanisms.

- The update and reset gates help filter out irrelevant information, making the model more robust to noise and variability in the input data.

+ : Computational Efficiency

Compared to other recurrent neural networks like LSTMs, GRUs are computationally more efficient due to their simpler structure.

- GRUs have fewer parameters than LSTMs, leading to faster training and inference times.
- This efficiency makes GRUs suitable for applications where computational resources are limited or when faster processing is required.

- : Limited Capacity for Extremely Long Sequences

Despite their ability to capture long-term dependencies, GRUs may still struggle with extremely long sequences.

- GRUs can still face gradient vanishing or exploding issues with very long sequences.
- The lack of control in the sequence length in this solution may not be optimal for a GRU.

- : Information Loss Due to Downsampling

The interesting solutions found on Kaggle for this competition that primarily used a GRU involved downsampling the data.

- While downsampling reduces the sequence length, making it more manageable for the GRU, it can also remove critical information needed for accurate event detection.
- Zinxhira's solution required downsampling the sequence by a factor two, reducing the sequence length and the number of measurements available by half.

4.2.3 The Highest Performance Solution : a hybrid model

The first place in the competition was achieved by user Urazalinov [31] with a hybrid model that combines a transformer encoder and two bi-directional LSTM layers.

The transformer encoder is part of the transformer model. It captures dependencies and relationships within the input sequence, effectively performing feature extraction. It uses self-attention mechanisms to weigh the importance of different parts of the input sequence relative to each other, allowing the model to understand context and relationships without regard to the position of the data points [32].

The bi-directional LSTM layers capture long-term dependencies and sequential patterns over time by processing the input sequence in both forward and backward directions. This bi-directional approach ensures that the LSTM layers can utilize information from both past and future contexts. Similar to the GRUs presented earlier, LSTMs are particularly effective at modeling temporal dynamics and sequential patterns.

A notable downside of this implementation, compared to the CNN and GRU approaches presented earlier, is its high computational requirements. The final model proposed by Urazalinov required over two hours of training time using a TPU VM v3-8. The efficiency of this model will later be compared with our own implementation.

Designing a hybrid model for FOG event detection is notably challenging because deeper networks present a higher risk of overfitting. Multiple users reported not being able to implement an effective hybrid model and chose to limit themselves to shallower networks [18][40]. Despite these challenges, the results clearly indicate that hybrid models are the most effective for this task. This is demonstrated by the substantial margin between the first and second place finishes on the competition leaderboard achieved using this approach [13].

Based on the observed potential of hybrid models in this competition, this approach will be considered for future implementations.

Chapter 5

Problem Statement

In this chapter, this thesis will closely examine the problem of detecting FOG events using accelerometer data. By exploring and understanding the specific characteristics of the dataset, this chapter outlines the various challenges encountered during the implementation. Understanding these challenges underscores the relevance of this problem and demonstrates how the proposed solution in this thesis can advance the state-of-the-art in FOG detection using deep learning.

5.1 Exploratory Data Analysis

5.1.1 Dataset Overview and Context

The data discussed comprises data series collected as patients were filmed performing tasks likely to provoke a freezing of gait (FOG) episode. During these tasks, patients wore a lower-back 3D accelerometer sensor. Experts reviewed the video recordings and annotated each frame, describing the type of event occurring [14].

While this manual process is relatively reliable and sensitive, it is extremely time-consuming, labor-intensive, and requires specialized expertise. Although effective, this method is not scalable and limits the ability to quickly and efficiently analyze FOG occurrences. The use of wearable sensors combined with machine learning techniques presents a promising alternative, allowing for the potential identification of FOG episodes without the need for video analysis. For this reason, the goal of this competition is to implement a model that can detect the presence or absence of a FOG event and the type of event occurring for each frame.

In this competition, Kaggle provides different datasets collected under distinct circumstances:

- The **tDCS FOG** (`tdcsfog`) dataset, comprising data series collected in the lab, as subjects completed a FOG-provoking protocol.
- The **DeFOG** (`defog`) dataset, comprising data series collected in the subject's home, as subjects completed a FOG-provoking protocol.

- The **Daily Living** (`daily`) dataset, comprising one week of continuous 24/7 recordings from sixty-five subjects (of which forty-five exhibit FOG symptoms).

The `daily` dataset was not videotaped or annotated by experts. No detection is required for the `daily` series, and this dataset can be discarded with no real impact on the detection model.

The `defog` and `tdcsfog` datasets are very similar in terms of composition and structure. Two approaches were used during the competition to handle these datasets: the first approach involved combining both datasets into a single comprehensive FOG dataset and training a single model for detection [40]; the second approach involved training two independent models with similar architectures, each tailored to one of the datasets [31]. Both methods yielded great results, indicating that a model performing well on one dataset will likely perform similarly on the other. Given that the time series in the `defog` dataset are significantly longer, resulting in extended training times, we will focus on the `tdcsfog` dataset for our exploration and experimentation. This approach allows for faster iteration and experimentation. Insights gained from the `tdcsfog` dataset will be applicable to the `defog` dataset. After developing our final architecture, we will combine the datasets to ensure more representative and robust results.

5.1.2 File and Field Description

The **tDCS FOG** dataset comprises different files and folders:

• **train/** contains the different data series in the training set. Each data series corresponds to an experiment. The first rows of a specific experiment can be observed in Table 5.1. The fields present are:

- `Time`: An integer timestep, recorded at 128Hz (128 timesteps per second).
- `AccV`, `AccML`, and `AccAP`: Acceleration from a lower-back sensor on three axes: V - vertical, ML - mediolateral, AP - anteroposterior. Data is in units of m/s^2.
- `StartHesitation`, `Turn`, `Walking`: The three types of FOG events, represented by a binary variable indicating the occurrence of each event type.

• **tdcsfog_metadata.csv** identifies each series in the `tdcsfog` dataset by a unique Subject, Visit, Test, and Medication condition.

- `Id`: The ID of a time series.
- `Subject`: The subject that performed the experiment referred to in `Id`.
- `Visit`: The number of the patient's visit.
- `Test`: Which of three test types was performed.
- `Medication`: Whether the subject is `off` or `on` anti-parkinsonian medication.

- **subjects.csv**: Metadata for each subject including `Age` and `Sex`.
- **events.csv**: Metadata for each FOG event in all data series.

The test set containing the time series used by Kaggle to score the performance of a model is hidden to ensure fair comparability among different competitors.

TABLE 5.1: First rows from a single data series from the `tdcsfog` dataset.

Time	AccV	AccML	AccAP	StartHesit	Turn	Walking
0	-9.6	0.9	0.9	0	0	0
1	-9.6	0.9	0.9	0	0	0
2	-9.6	0.8	0.9	0	0	0
3	-9.6	0.8	0.9	0	0	0
4	-9.6	0.8	0.9	0	0	0
5	-9.6	0.8	0.9	0	0	0

5.1.3 Exploration

The objective of this exploration is to gain a deeper understanding of the dataset, focusing on the key features and event types.

The `train` folder consists of 833 time series, each representing a lab experiment in the form of time series data. The length of these series ranges from 2,359 timesteps to 97,077 timesteps, representing experiments lasting from 18 to 758 seconds. By concatenating the different experiments and mapping their IDs to the corresponding experiment in **tdcsfog_metadata.csv**, we can associate each experiment with a subject. This approach reveals that the data series were collected from only sixty-two different subjects. This limited number of subjects is one of the main challenges introduced in the next section.

We now explore the three features (`AccV`, `AccML`, and `AccAP`) and the three corresponding events (`StartHesitation`, `Turn`, `Walking`) in greater detail.

To visualize the different features and events, we start by plotting the acceleration data over time with event overlays for a single experiment. The result, shown in Figure 5.1, helps us understand the structure of each time series. Experiments begin with the patients standing still for a few seconds before starting the FOG provoking protocol. During the experiment, events may occur for a specified number of timesteps. In this specific time series, only the `Turn` event occured for a single time window, represented in red.

It is also relevant to visualize the correlation between each acceleration feature and the event types. This is achieved in Figure 5.2. While no significant observation can be made from the vertical acceleration feature, we find a correlation between mean `AccML` or `AccAP` values closer to zero and the absence of events.

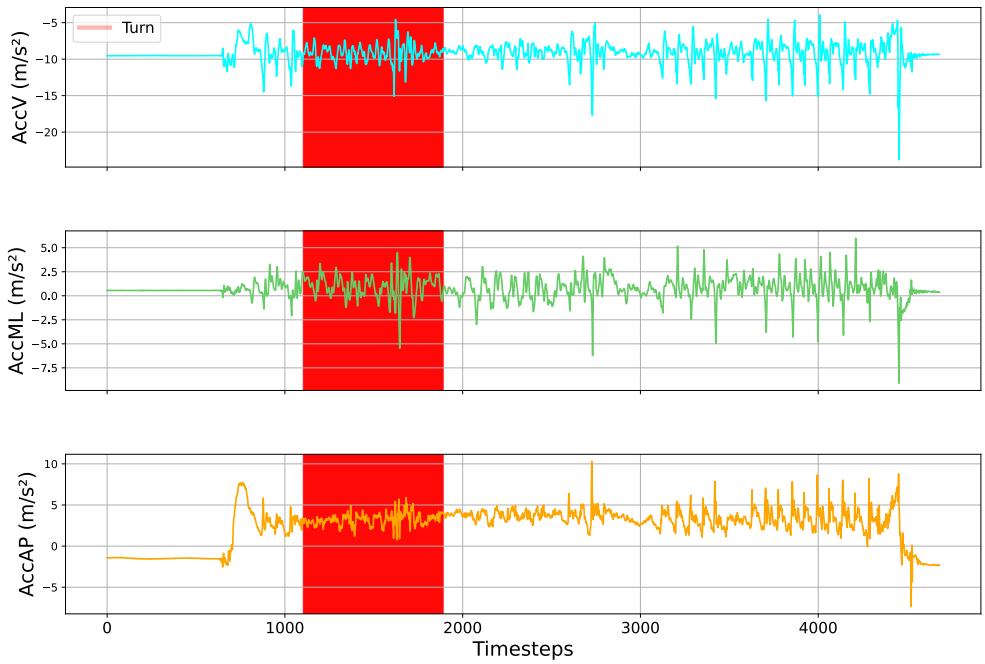


FIGURE 5.1: Acceleration data over time with event overlays. The red rectangle represents the time window during which the Turn event occurred

While the first two illustrations are useful for visualizing the data and identifying simple correlations, further exploration regarding event duration and frequency is crucial for understanding the challenges we face and designing an effective solution. The distribution of different events within the dataset varies significantly. As highlighted in Figure 5.3, 23% of timesteps are annotated as a `Turn` event, while less than 5% of timesteps are annotated as `StartHesitation` or `Walking` events. Considering all data series, `Turn` events occurred 958 times, while `StartHesitation` and `Walking` occurred 100 and 108 times, respectively.

Due to the sporadic nature of FOG, we observe significant discrepancies even within each event type as the duration of these events varies between experiments. The table 5.2 below illustrates this by highlighting the key metrics regarding the duration of each event type.

5.2 Challenges

Due to the nature of the data available and the current advances on FOG detection, two main challenges will have to be addressed during the implementation of a novel model. First, current datasets used for training and testing these models are

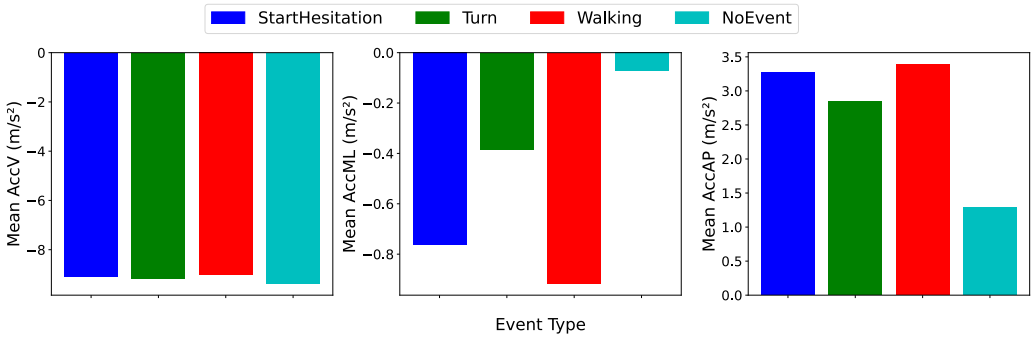


FIGURE 5.2: Correlation analysis between acceleration features and event targets

TABLE 5.2: Event duration statistics

	StartHesitation	Turn	Walking
Mean Duration	3,047	1,752	1,924
Shortest Duration	31	26	23
Median Duration	491	430	427
Longest Duration	18,773	74,493	21,018

limited in size and scope, which significantly affects the generalizability of the models. Additionally, while high levels of accuracy have been achieved, other critical metrics such as precision have largely been ignored.

5.2.1 Precision

The shortcomings of accuracy

The emphasis when designing models for HAR has often been on achieving high levels of accuracy, as shown by the models presented by Ignatov[11] and Dua, Singh, and Semwal[6]. These models focused on accuracy during development and highlighted this metric to assess their performance. Accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5.1}$$

"There is a downside to focusing on accuracy as a primary metric. The reason is that it treats all classes as equally important and looks at all correct predictions" [28]. It is important to consider the distribution of the event types presented in Section 5.1.3:

- 23% of data entries are of type `Turn`
- 4% of data entries are of type `StartHesitation`

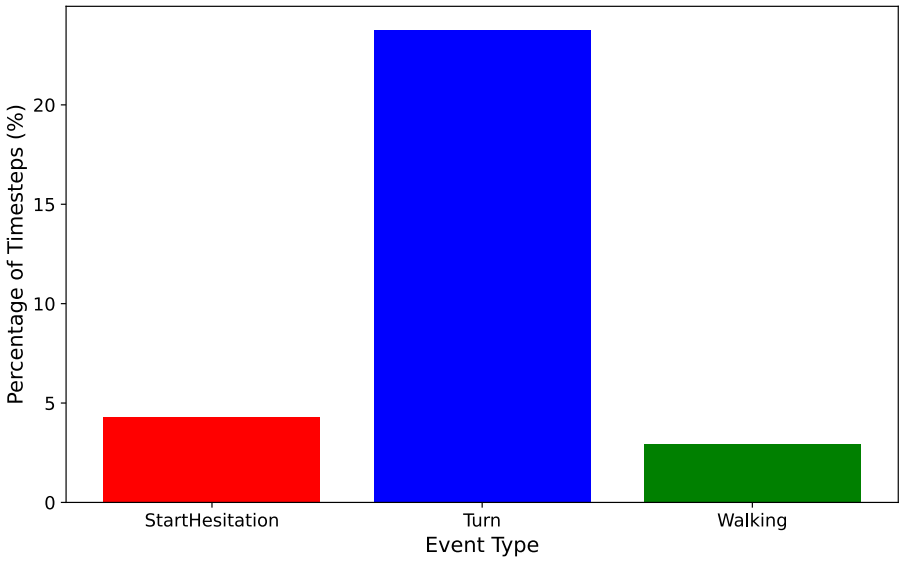


FIGURE 5.3: Percentage of each event type within the complete dataset

- 3% of data entries are of type `Walking`
- 70% of data entries are non-events

Given this distribution, it would be easy to achieve high accuracy by simply predicting the majority class for every time step. If a model always predicted "non-event," it would be correct 70% of the time, leading to a high accuracy despite not actually detecting any events. This is a common issue in imbalanced datasets and highlights the need for other metrics to ensure an effective model.

Average Precision

Due to the shortcomings of accuracy, the official metric used by Kaggle to score submissions and the one we will focus on when developing our novel model is the **Mean Average Precision** (mAP). We will compute the **average precision** (AP) on predicted confidence scores separately for each of the three event classes and take the average of these three scores to get the overall score.

Average Precision (AP) is a metric used to evaluate the performance of classification models, particularly with imbalanced datasets. Each prediction made by the model is associated with a confidence score, indicating the probability or confidence of the prediction being correct. By varying the confidence score threshold, we can compute precision and recall [1] at different levels, resulting in a precision-recall curve

[1]Precision is the ratio of correctly predicted positive observations to the total predicted positives, while recall is the ratio of correctly predicted positive observations to all actual positives.

[39].

AP is calculated as the area under the precision-recall curve. Mathematically, it is expressed as:

$$\text{AP} = \sum_{n=1}^{N} (R_n - R_{n-1}) P_n \tag{5.2}$$

where P_n and R_n are the precision and recall at the nth threshold, and N is the number of thresholds. Figure 5.4 illustrates the precision-recall curve for the `turn` event achieved by a benchmark model can help visualize the concept of AP. It would show how precision varies with recall at different thresholds, with the area under the curve representing the AP.

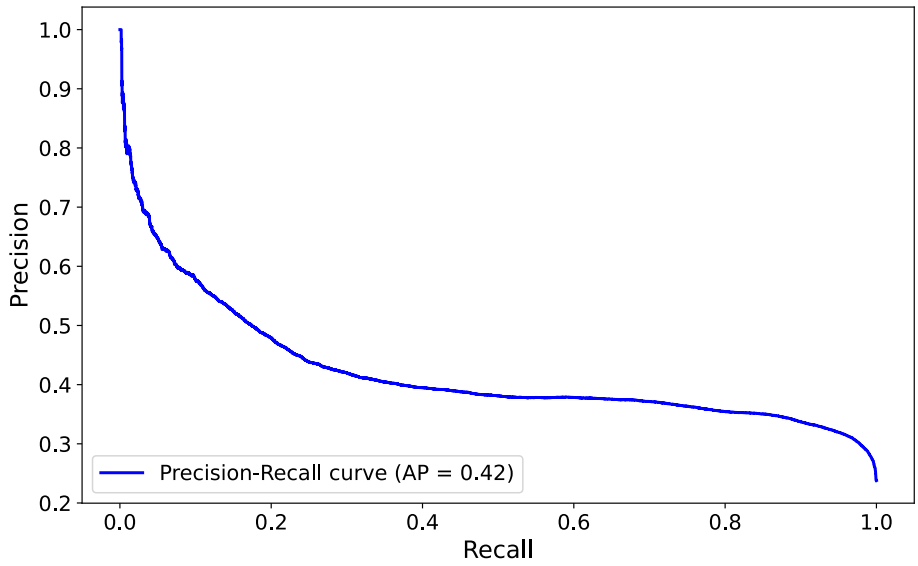


FIGURE 5.4: Example of a Precision-Recall Curve obtained during model testing

mAP is the average of the AP values for each class, providing a single metric that summarizes the overall performance of the model across all classes:

$$\text{mAP} = \frac{1}{C} \sum_{c=1}^{C} \text{AP}_c \tag{5.3}$$

where C is the number of classes, and AP_c is the Average Precision for class c.

Working with Mean Average Precision (mAP) is challenging as it requires balancing the model's performance across all classes, including rare events. Detection of

infrequent events such as `StartHesitation` and `Walking` is critical. The mAP scores obtained by the participants to the competition are presented on the figure 5.5 below for future comparisons.

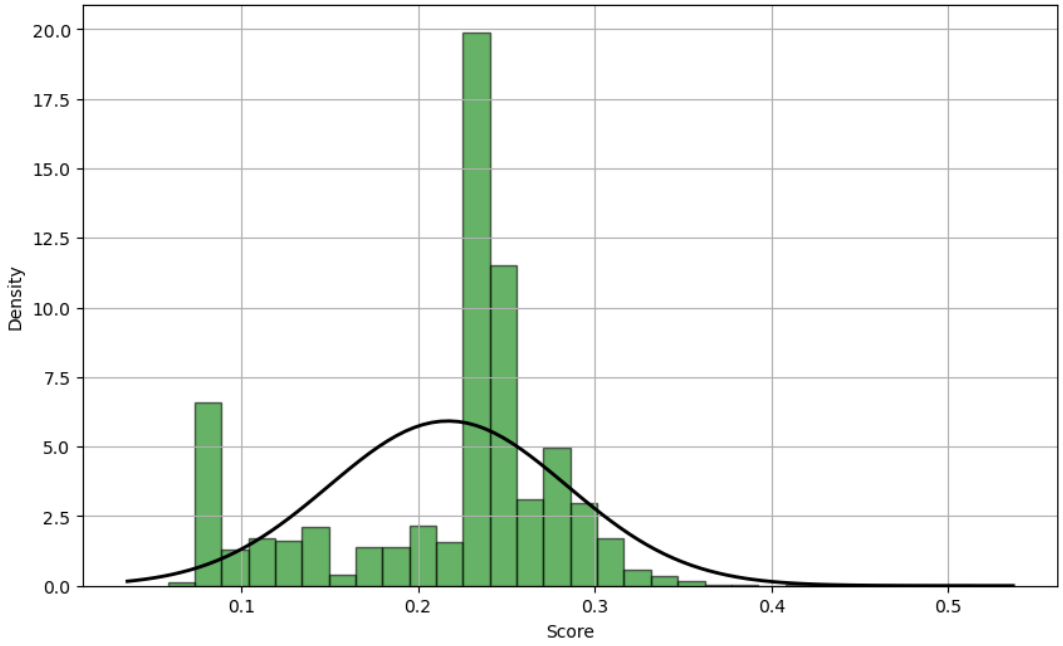


FIGURE 5.5: Scores from the leaderboard for the FOG detection competition.

5.2.2 Generalizability

In the article "Deep Learning in Human Activity Recognition with Wearable Sensors: A Review on Advances", Zhang et al. discusses the difficulties of acquiring a sufficiently large corpus of quality data for HAR. "When training a DL model with a limited dataset size, the model will be prone to overfitting, and the generalizability will be sacrificed... The quality of models is highly dependent on the quality of the training data" [38]. Thanks to the EDA performed earlier, two factors complicating the generalizability of models have been identified:

1. **Lack of data for certain events**: Events such as `walking` only contain 108 windows in the training data. Coupled with the sporadic nature of FOG, each event varies significantly, making it difficult for the model to recognize patterns and predict accurately on new data.

2. **Limited diversity of patients in the training data**: The training data includes only sixty-two different patients. Each patient has a unique gait and different reactions to freezing of gait, making it challenging for the model to generalize patterns to unseen patients.

The Table 5.3 illustrates the difficulty of generalizing to new subjects by comparing the scores of two benchmarks models detecting events on patients both present and absent from the training set.

TABLE 5.3: Performance comparison using mAP for benchmark CNN and GRU models detecting events on patients both present and absent from the training set

Model used	**CNN**	**GRU**
absent subject	0.29	0.32
present subject	0.41	0.43

5.2.3 Objectives

After outlining the challenges in FOG (Freezing of Gait) detection and reviewing related works, we can define the requirements for our implementation more clearly.

Model Learning Requirements: The model must learn complex patterns from sequential data and generalize these patterns effectively. Zhang et al. (2022) [38] list and compare several models that are well-suited for this task.

Handling Low-Quality Data: The model must handle low-quality data due to the small corpus and the noisy nature of sensor data. Insights from Ignatov et al. (2018) [11] and the solution proposed by Moro (2023) [18] suggest that CNNs can address this through dimensional reduction. Additionally, RNNs like LSTMs and GRUs are a good fit due to their robustness to noise, as demonstrated by the performance of Zinxhira's model [40].

Performance: Given the promising results demonstrated by hybrid models [31], the proposed model should explore a new hybrid architecture. This approach is expected to improve the overall performance. The results will be compared to other submissions.

Building on the promising results achieved by both CNNs and GRUs in detecting Freezing of Gait (FOG), this master's thesis aims to explore the potential of a hybrid model that combines the strengths of both approaches. The primary objective is to develop and implement a novel CNN-GRU model specifically tailored for FOG detection. Inspired by the model introduced in the article 'Multi-input CNN-GRU based human activity recognition using wearable sensors' [6], this thesis will adapt and extend the architecture to suit the FOG detection context, with the goal of designing a new model and benchmarking its performance against existing solutions.

The proposed model aims to:

- **Utilize CNNs for Feature Extraction Reduction**: Leverage the ability of Convolutional Neural Networks to detect local features, enabling efficient feature extraction and dimensionality reduction.

- **Enhance Temporal Context Understanding**: Address the limited temporal context understanding of CNNs by incorporating Gated Recurrent Units (GRUs) to capture long-term dependencies.

- **Performance and Efficiency**: Leveraging the computational speed of both CNNs and GRUs, the hybrid CNN-GRU model is designed for efficient performance without compromising accuracy. Our goal is to balance accuracy with computational efficiency. We will assess the proposed model's efficiency and compare it to the Transformer-biLSTM approach [31].

By achieving these objectives, the thesis seeks to advance the state-of-the-art in FOG detection by providing new insights into CNN-GRU models for human activity recognition and introducing a novel approach for effective and efficient FOG detection.

Chapter 6

Solution

This chapter focuses on the implementation choices and architectures developed for this master's thesis.

After discussing data preparation, we will detail the implementation of two benchmark models: a CNN and a GRU. These models are essential for gaining insight into their ability to assess FOG detection and for ensuring that the new hybrid CNN-GRU model surpasses both in performance. Finally, we will present the implementation of the novel hybrid model.

6.1 Preparing the Data

The first step of the implementation is to prepare the data to enable effective training. This process includes pre-processing and sequencing the data.

6.1.1 Pre-processing

The pre-processing step of our implementation ensures the data is clean and in a suitable format for the model. This part consists of three different stages.

Grouping the time series

As the data is distributed across more than eight hundred different time series, we group them into a single DataFrame and add a column to reference the corresponding time-series ID. This consolidation facilitates easier manipulation of the data, allowing us to manage the time series as a cohesive dataset rather than dealing with multiple fragmented series.

Mapping the subjects

Our model aims to detect events in data from unknown subjects. Since these subjects are not present in the original time series, we perform a mapping between the time-series IDs in our DataFrame and the corresponding subjects in `tdcsfog_metadata.csv`. This mapping allows us to add a new column that captures the specific subject on which the measurements were taken. This step is crucial for ensuring that our model

can generalize well to new, unseen subjects by properly structuring the data during training.

Standardization

An example of feature measurements was illustrated in Figure 5.1. We can observe that the features do not have the same scales. To prevent the model from being biased towards features with larger scales, regularization or normalization of the data is essential. For this implementation, we apply standardization to the acceleration features of each experiment individually.

Motivation:

- **Why Standardization:** Standardization transforms the data to have a mean of zero and a standard deviation of one. This is crucial for many machine learning algorithms as it ensures that each feature contributes equally to the model's performance and helps in achieving faster convergence during training[12].
- **Why Not Normalization:** Normalization scales the data to a specific range, typically [0, 1]. While this can be useful for some applications, it is not appropriate here because it does not account for the variance in the data, which can be significant in accelerometer measurements. Standardization, on the other hand, maintains the distribution of the data, which is important for preserving the characteristics of the original signals [35].
- **Why Individually on Each Experiment:** Each experiment may have different characteristics and scales due to varying conditions and subjects. As the patients perform different tasks and have different gaits, the resulting measurements vary and have different scales. Standardizing each experiment individually ensures that the scaling reflects the specific context of each experiment. This prevents the introduction of biases that could arise if a global standardization was applied, which could give too much weight to specific experiments.

Standardization is performed using the `StandardScaler` from the `sklearn` library. Applying the `StandardScaler` as described resulted in a relative mAP score improvement of up to 50%. The first lines of the DataFrame can be observed in Table 6.1 below.

TABLE 6.1: First five lines of the DataFrame after pre-processing. Each row represents a timestep with standardized accelerometer measurements, corresponding labels and identifications.

Time	AccV	AccML	AccAP	SHes	Turn	Walk	Id	Subject
0	-0.222604	-0.116734	-0.459735	0	0	0	690...	2a39f8
1	-0.218296	-0.119831	-0.459735	0	0	0	690...	2a39f8
2	-0.215475	-0.118364	-0.463046	0	0	0	690...	2a39f8
3	-0.221249	-0.116820	-0.461988	0	0	0	690...	2a39f8
4	-0.219619	-0.121241	-0.468508	0	0	0	690...	2a39f8

6.1.2 Sequencing

Sequencing organizes the data into the format required by the model, preparing the features and labels in separate arrays with specific sequence sizes for effective use. Due to the size of our data and the limited RAM available, it is impractical to provide the entire sequenced dataset to the model in one go. Therefore, we need a method that can yield sequences dynamically during model training. To address this, we implemented the `TimeseriesGenerator` utility class from the `Keras` library for generating batches of temporal data.

The `TimeseriesGenerator` operates through two main functions. The initialization function, executed once, sets up variables such as the features, labels, sequence length, and batch size. The second function creates sequences based on the information provided during initialization, groups these sequences into batches, and supplies them to the model during training.

While the model training effectively utilizes the computation capabilities of the GPU, the function used by the `TimeseriesGenerator` to prepare and send the batches operates on the CPU. To avoid slowing down the training process, we precompute information such as the starting position of each sequence during initialization, thus minimizing the computational load on the CPU during batching.

Implementation choices specific to each model will be detailed in Section 6.

6.2 Benchmark CNN

This section will detail the implementation of the benchmark CNN model.

6.2.1 Sequencing

During the sequencing for the CNN model, we chose to assign only a single label per window instead of a label per timestep as initially provided by the dataset. While this means that some timesteps might be mislabeled, this effect is mitigated by the fact that the events typically span long windows of time. Empirical experiments confirm that the loss in the mAP score due to the mislabeling of some timesteps is negligible. On the bright side, this approach allows us to reduce the dimension of the output and capture the overall context of the sequence within that window, which is beneficial for CNNs that are effective at recognizing spatial patterns

6.2.2 Architecture

This specific architecture, which will be used for our benchmark CNN model, was developed thanks to the insights from an existing Kaggle solution [18] and related literature [23].

- **Input Layer**: Takes input sequences with shape (`window_size`, `n_features`), where `window_size` is the length of the sequence and `n_features` is the number of features per timestep.

- **Convolutional Block**: This block is applied three times in a row.
  - **Conv1D Layer**: Applies 1D convolution to extract `n_filters` filters, each of size `k_size`. "Same" padding is used to preserve the input shape.
  - **BatchNormalization Layer**: To mitigate overfitting and speed up training. Normalizes the output of the previous layer to have zero mean and unit variance.
  - **ReLU Activation Layer**: Applies the ReLU activation function to introduce non-linearity.
  - **Dropout Layer**: Applies dropout with a rate of 0.2 to prevent overfitting.
- **GlobalAveragePooling1D Layer**: Reduces each feature map to a single value by averaging all values in the sequence, effectively reducing the spatial dimensions.
- **Dense (Output) Layer**: A fully connected layer with `n_labels` units, used to enable classification. Produces the final output.

The resulting architecture, along with example output dimensions, can be observed in Figure 6.1.

6.2.3 Discussion

The results and hyperparameters will be detailed in a later chapter. However, a short analysis is still relevant to gain insights for building our hybrid model.

Average pooling is great for generalizing because it considers all activations, reducing sensitivity to specific dominant features. This approach outperformed "max pooling" as it helps in maintaining a more balanced representation of the features, leading to better generalization and robustness against overfitting [37].

The CNN model performed the best with a small window size (shorter than a second), indicating that it is most effective at extracting local features when confined to short time periods.

6.3 Benchmark GRU

This section will detail the implementation of the benchmark GRU model.

6.3.1 Sequencing

Due to the high frequency of the sensor measurements, the features exhibit only slight variations between consecutive timesteps. For this reason, despite the GRU's inherent robustness to noise, the model showed improvement when sampling only

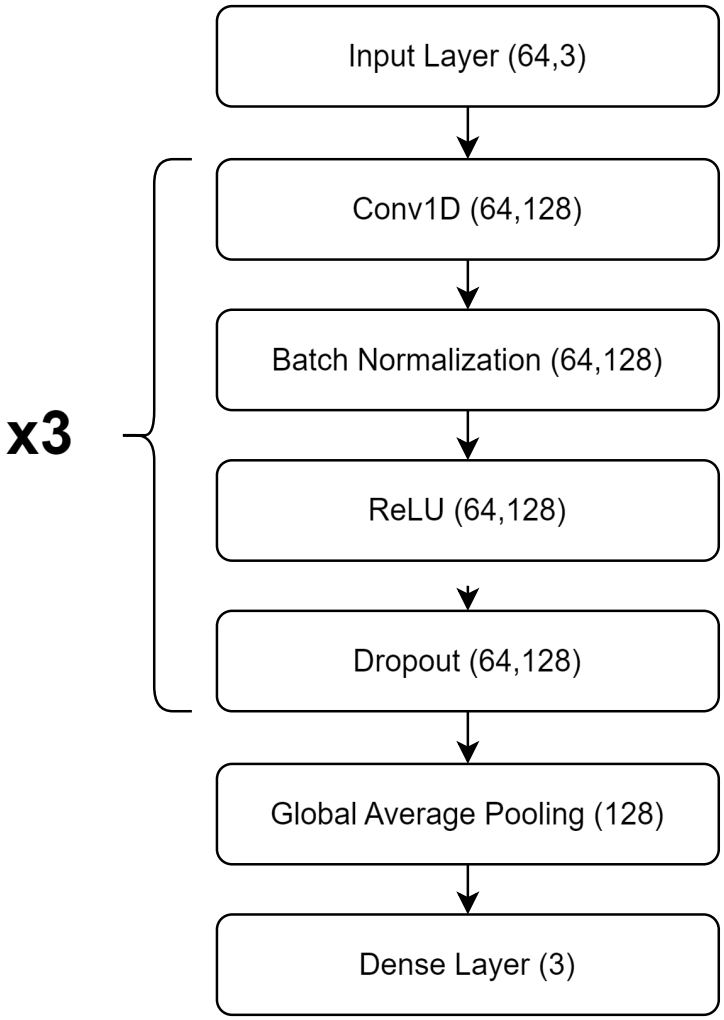


FIGURE 6.1: Architecture of the benchmark CNN with example output dimensions

one timestep out of every two. The sampling rate can be specified when using the `TimeseriesGenerator`.

6.3.2 Architecture

The specific architecture of the benchmark GRU is detailed below. This architecture took inspiration from a Kaggle solution [40].

- **Input Layer**: Takes input sequences with shape (`window_size`, `n_features`), where `window_size` is the length of the sequence and `n_features` is the number of features per timestep.
- **Dense Layer**: Applies a dense layer to align dimensions with the future GRU layer.

- **Batch Normalization Layer**: Normalizes the output of the dense layer to have zero mean and unit variance, mitigating overfitting and speeding up training.
- **ReLU Activation Layer**: Applies the ReLU activation function to introduce non-linearity.
- **Bi-directional GRU Block with Residual Connection**: This block is applied three times in a row.
  - **Bi-directional GRU Layer**: Applies a Bi-directional GRU with `n_units` units, returning sequences. This layer captures dependencies from both forward and backward directions.
  - **BatchNormalization Layer**: Normalizes the output of the GRU layer to have zero mean and unit variance, mitigating overfitting and speeding up training.
  - **Dropout Layer**: Applies dropout with a rate of 0.2 to prevent overfitting.
  - **Residual Connection**: Adds the input to the output of the GRU layer to form a residual connection. Mitigates the vanishing gradient problem in deep networks.
- **Time-distributed Dense Layer**: A fully connected layer with `n_labels` units, used to enable classification. Produces the final output.

The resulting architecture, along with example output dimensions, can be observed in Figure 6.2.

6.3.3 Discussion

Similar to the CNN, the results and hyperparameters for the GRU model will be detailed in a later chapter. However, a brief discussion remains relevant to gain insights for building our hybrid model.

Applying a bi-directional wrapper to the GRU layers enabled the model to use both past and future context, as we have access to the entire data sequence. This bi-directional approach improved the model's ability to generalize by capturing dependencies from both directions.

The residual connections significantly mitigated the vanishing gradient problem, enabling the use of larger window sizes and improving precision. This component proved crucial in achieving convincing performance with the GRU model.

The model demonstrated consistent improvement as the window size increased to a certain point, after which the performance plateaued. This suggests that the GRU is most effective at capturing temporal dependencies over larger time scales.

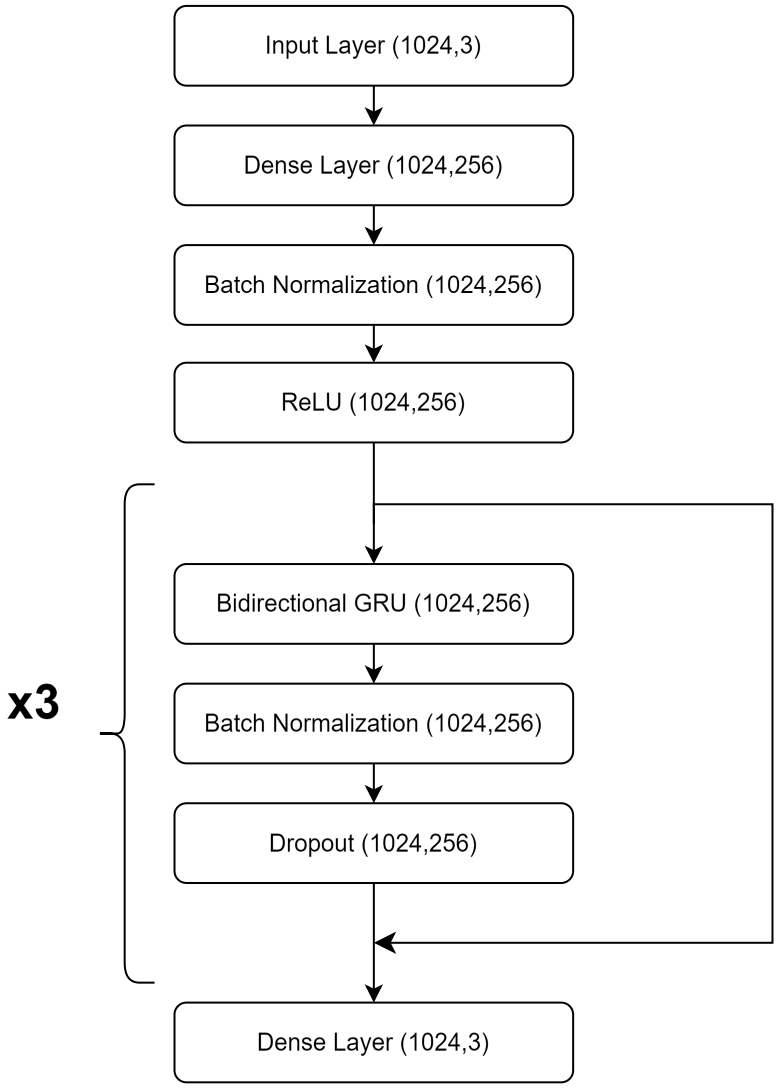


FIGURE 6.2: Architecture of the benchmark GRU with output dimensions

6.4 CNN-GRU architecture

This section details the implementation of the CNN-GRU model. We start with the basic CNN-GRU architecture, followed by modifications tailored for FOG detection.

6.4.1 Basic CNN-GRU

The basic CNN-GRU architecture combines the two previously defined models by stacking a CNN on top of a GRU. This combination allows the model to effectively handle both the spatial and temporal aspects of the input data. The CNN extracts local features within the specified window, while the GRU captures the sequential dependencies of these features.

However, simply merging the CNN and GRU models, as defined earlier for the benchmark, did not yield significant improvements in results. This suggests that further tailoring and optimization of the model are necessary for effective FOG detection.

6.4.2 Simplified Network

Each additional convolutional or recurrent layer enhances the model's ability to recognize more complex patterns. However, deeper networks are more prone to overfitting, where the model learns noise and details specific to the training data rather than generalizing well to new data [1].

Given the characteristics of the FOG dataset and our need to generalize detection to new subjects, the model showed better performance with a shallower architecture. Consequently, we will utilize an architecture with only two convolutional blocks and two GRU blocks, as proposed by Dua, Singh, and Semwal [6].

6.4.3 Time-distributed CNN + GRU

As of now, the model is able to demonstrate similar results to the two benchmark models, but is still unable to surpass them. We have to take a closer look at the insights of each model to propose further upgrades.

On one hand, the GRU requires large windows to be able to understand the temporal context effectively. According to the insights from Section 6.3.3, the GRU is unable to perform optimally if the window does not capture several seconds of continuous measurements. This is because GRUs rely on capturing long-term dependencies and patterns within the data, which are better represented in longer windows. Shorter windows fail to provide the GRU with sufficient context, resulting in suboptimal performance.

On the other hand, according to the insights from Section 6.2.3, the CNN performed optimally when extracting features from periods shorter than a second. For FOG detection using CNNs, it is crucial to keep the window size small to allow convolutional filters to focus on very localized features and enhance generalization by learning patterns across many small segments. Larger windows deteriorated the CNN's performance by introducing irrelevant information.

Due to this conflict, we apply a `Time-distributed` wrapper to the layers of the convolutional part of the architecture. This allows the CNN and the GRU to operate on different time scales. The CNN can effectively stay confined to smaller windows to extract localized features, while the GRU can function on a larger window, leveraging the required temporal context.

The new model functions as follows: each sequence is divided into multiple subsequences. A CNN is applied to each subsequence independently to extract features. The extracted features from all subsequences are then flattened into 1D vectors. These vectors are fed sequentially into a GRU, which captures the temporal

dependencies across the entire sequence.

This new architecture clearly enhances the model's ability to detect FOG events and now outperforms both benchmark models. By using the time-distributed wrapper, the model effectively leverages the strengths of both the CNN and the GRU. This approach has already been used successfully for HAR [6] or tasks such as speech emotion recognition [19].

6.4.4 Improved Sequencing

To accommodate the new model's requirements, each sequence is further divided into multiple subsequences of a specified length during batching. Similar to the approach used for the benchmark CNN, we assign a single label to each subsequence, which reduces the dimensionality of the output. The CNN effectively reduces the dimensionality of the input through pooling layers, eliminating the need for resampling the input data.

Additionally, the bi-directional GRUs ability to use temporal context is limited at the edges of the sequences, as it only has context from one direction. To address this issue, it is possible to apply an overlap to the sequences. This overlap ensures that the edge of one sequence is in the center of another, providing the GRU with sufficient context from both directions across the entire time series. This strategy improves the GRU's ability to capture temporal dependencies. The tradeoff of this overlap is an increased computational load as it duplicates the data.

A first time-distributed CNN-GRU model will be trained without overlap and a second one with 50% overlapping sequences.

An example of the sequencing applied to the data is shown in Figure 6.3.

6.4.5 Final Architecture

This section showcases the hybrid architecture developed for FOG detection. It is inspired by both architectures defined for our benchmark models and literature on HAR using CNN-GRU models [6].

- **Input Layer**: Takes input sequences with shape (`n_subsequence`, `subsequence_length`, `n_features`), where `n_subsequence` is the number of subsequences, `subsequence_length` is the length of each subsequence, and `n_features` is the number of features per timestep.
- **Time-distributed Convolutional Block**: This block is applied twice in a row.
  - **Time-distributed Conv1D Layer**: Applies 1D convolution independently to each subsequence to extract `n_filters` filters, each of size `k_size`. "Same" padding is used to preserve the input shape.

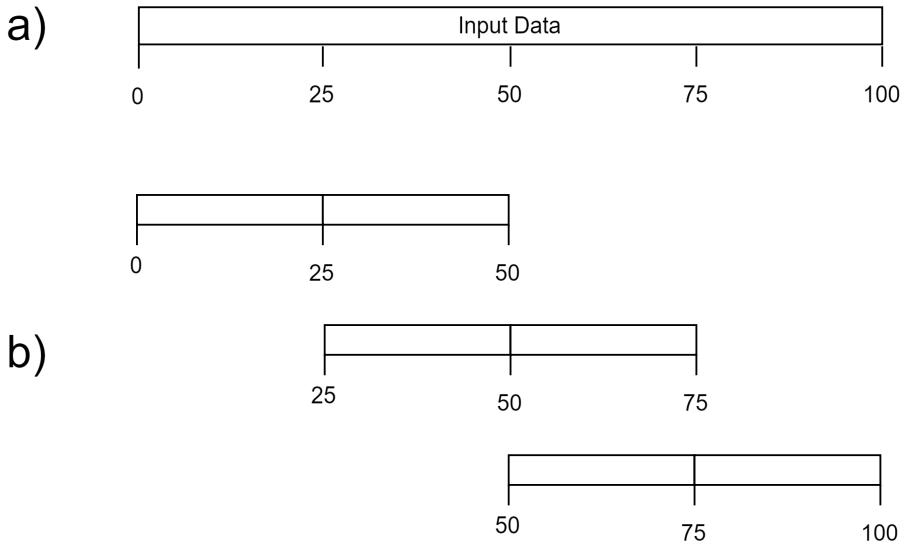


FIGURE 6.3: Sequencing of 100 timesteps using sequences of size 50 and subsequences of size 25 with a 50% overlap.
a) Input data b) Resulting sequences

  - **Time-distributed BatchNormalization Layer**: Normalizes the output of the previous layer to have zero mean and unit variance, mitigating overfitting and speeding up training.
  - **Time-distributed ReLU Activation Layer**: Applies the ReLU activation function to introduce non-linearity.
  - **Time-distributed Dropout Layer**: Applies dropout with a rate of 0.2 to prevent overfitting.

- **Time-distributed AveragePooling1D Layer**: Applies average pooling with a specified pool size independently to each subsequence to reduce the spatial dimensions.
- **Time-distributed Flatten Layer**: Flattens the pooled feature maps independently for each subsequence.
- **Dense Layer**: Aligns dimensions for the future GRU layers by applying a dense layer
- **BatchNormalization Layer**: Normalizes the output of the dense layer to have zero mean and unit variance, mitigating overfitting and speeding up training.
- **ReLU Activation Layer**: Applies the ReLU activation function to introduce non-linearity.
- **Bi-directional GRU Block with Residual Connection**: This block is applied twice in a row.

  - **Bi-directional GRU Layer**: Applies a Bi-directional GRU with `units` units, returning sequences. This layer captures dependencies from both forward and backward directions.
  - **BatchNormalization Layer**: Normalizes the output of the GRU layer to have zero mean and unit variance, mitigating overfitting and speeding up training.
  - **Dropout Layer**: Applies dropout with a rate of 0.2 to prevent overfitting.
  - **Residual Connection**: Adds the input to the output of the GRU layer to form a residual connection. Mitigates the vanishing gradient problem in deep networks.

- **Time-distributed Dense Layer**: A fully connected layer with `n_labels` units, used to enable classification. Produces the final output.

This architecture is illustrated on Figure 6.4.

To the best of the author's knowledge, a hybrid time-distributed CNN-GRU architecture has not yet been successfully implemented for detecting FOG events. Therefore, this model represents a novel approach to the task of Freezing of Gait detection using deep learning.

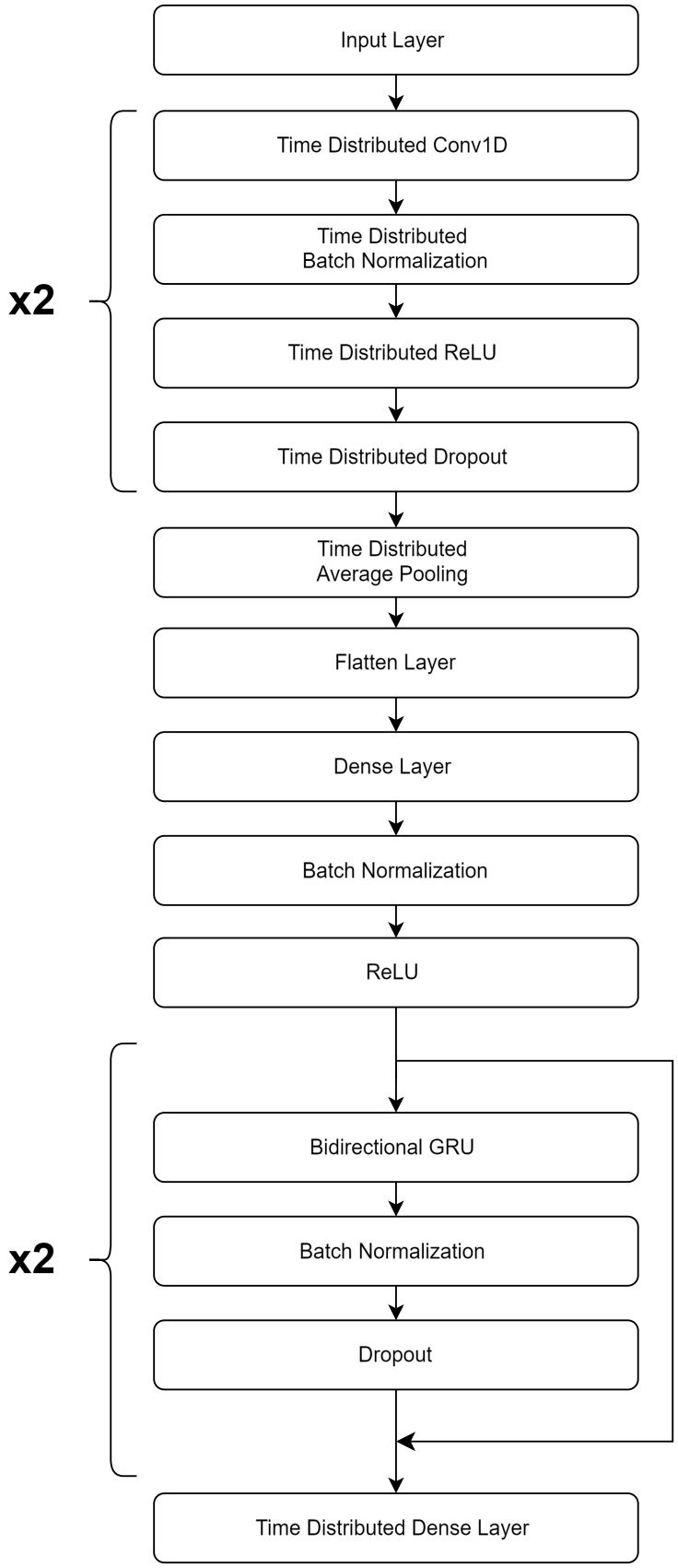


FIGURE 6.4: Time-distributed CNN-GRU architecture tailored for FOG event detection

Chapter 7

Validation

This chapter will outline the methodology and experiments undertaken to demonstrate the effectiveness of a time-distributed CNN-GRU model in a specific HAR task, the detection of FOG events.

7.1 Implementation Environment

The entire implementation was carried out in Python using a Jupyter notebook on Kaggle, leveraging the `Keras` and `sklearn` libraries. Keras, built on TensorFlow, was employed for deep learning model development. To expedite the training process, a GPU P100 was utilized along with efficient multi-processing. This setup allowed for faster model training and more efficient handling of computationally intensive tasks.

The specific versions used in the implementation are as follows:

- Python version: 3.10.13
- Keras version: 2.15.0
- TensorFlow version: 2.15.0

7.2 Benchmark Models

To evaluate the performance of the time-distributed CNN-GRU model, two benchmark models were developed for this master's thesis: a standalone CNN and a standalone GRU. Both benchmark models are detailed further in Section 6.

These benchmarks serve to assess the hybrid model's ability to leverage the strengths of both individual models. To ensure a fair comparison, all models were tuned and tested using identical procedures. Additionally, the hybrid model was compared under the same conditions, including the following optimizations:

- **Residual Connections:** Residual connections, which are employed to enhance the recurrent part of the hybrid model, were also incorporated into the benchmark GRU model.

- **Overlap:** Since the benchmark models were not tested with overlapping windows, the performance of the hybrid model should first be evaluated without this optimization.

7.3 Model Tuning

Hyperparameter tuning is a critical process in optimizing neural network performance. The hyperparameters considered for tuning in this study include:

– **n_filters**: The number of kernels in the convolutional layers.
– **k_size**: The size of kernel used in the convolutional layers.
– **units**: The number of units in the GRU layer.
– **subsequence_length**: The length of each subsequence.
– **n_subsequence**: The number of subsequences in a sequence.
– **gru_drop**: The dropout rate applied to the GRU layers.
– **cnn_drop**: The dropout rate applied to the CNN layers.
– **pool_size**: The size of the pooling window.
– **lr**: The learning rate, which controls the step size during gradient descent.
– **batch_size**: The number of samples processed before the model's internal parameters are updated.
– **n_epochs**: The number of times the entire dataset is passed through the network during training.
– **overlap**: The percentage of the sequence that overlaps with the next sequence.
– **padding**: The technique used to handle borders during convolution.
– **activations**: The activation functions (ReLU, tanh, sigmoid) applied to introduce non-linearity into the model.
– **optimizer**: Optimization algorithm used to update the model.
– **loss function**: Function that measures the error between predicted and actual outputs.

Exploring three possible values for each hyperparameter would require training millions of models, which is computationally infeasible. To efficiently narrow down the search space, we divided the tuning process into four stages: *Initial Sensitivity Analysis*, *Random Search*, *Grouping Hyperparameters*, and *Grid Search*.

Additionally, overlap will only be introduced and fixed at the very end for the reason described in Section 7.2.

Initial Sensitivity Analysis:
Initial Sensitivity Analysis is a preliminary step where we vary one or two hyperparameters at a time to identify which ones have the most significant impact on the model's performance. Based on these observations, we fixed certain hyperparameters (Table 7.1) to reduce the search space.

TABLE 7.1: Hyperparameters fixed based on Initial Sensitivity Analysis

Hyperparameter	**Fixed Value**
`Activations CNN`	ReLU
`Activations GRU`	tanH + sigmoid
`Optimizer`	Adam
`Loss Function`	BinaryCrossentropy
`Padding`	same

Random Search:
The next step involves a Random Search to identify promising hyperparameter combinations. We defined three to five reasonable values for each hyperparameter and trained a hundred models using random combinations. This approach allowed us to explore a broad range of potential configurations efficiently, identifying those that yield the best performance.

Grouping Parameters:
Based on the Random Search results, we grouped the hyperparameters into different categories, as shown in Table 7.2

TABLE 7.2: Hyperparameter Grouping for Targeted Tuning

Group Name	**Hyperparameters**
Model Complexity Parameters	n_filters, k_size, units
Regularization Parameters	cnn_drop, gru_drop
Sequence Parameters	subsequence_length, n_subsequence, pool_size
Training Parameters	lr, batch_size, n_epochs

Grid Search:
Finally, Grid Search is conducted within each parameter group. This systematic method tests all possible combinations of specified parameter values to identify the best-performing model configuration. For each parameter group, the other groups' parameters were fixed at their baseline values to isolate their impact on model performance.

Once the best combination for each parameter group is identified, the model is considered optimized. The configuration of each model is provided in Table 7.3 for reproducibility.

7.4 Validation

7.4.1 FOG Detection

The models were initially validated on the comprehensive FOG dataset, which was created by combining the `tdcsfog` and `defog` datasets. We employed a custom

TABLE 7.3: Hyperparameters Configuration for Each Model

Hyperparameter	**CNN**	**GRU**	**CNN-GRU**
subsequence_length	64	1024	32
n_subsequence	-	-	64
n_filters	128	-	256
k_size	15	-	7
pool_size	64	-	1
units	-	128	128
cnn_drop	0.2	-	0.5
gru_drop	-	0.2	0.2
lr	2×10^{-5}	1×10^{-3}	1.5×10^{-4}
batch_size	128	64	32
n_epochs	10	20	15

cross-validation strategy that was specifically tailored to the unique characteristics of this dataset.

To achieve balanced and representative splits, we utilized the `StratifiedKFold` function from the `sklearn` library. This method produced five distinct splits, with each split containing unique subject IDs to ensure that no subject appears in more than one split. Furthermore, we ensured that each event label was equally represented across the splits, a critical step in addressing the inherent class imbalance within our dataset.

Following this, we conducted a five-fold cross-validation using the computed splits. In each iteration, the subjects from four splits were used for training, while the remaining subjects were used for validation. This process was repeated five times, with each split serving as the validation set once. By averaging the results across these folds, we obtained a more reliable estimate of the model's performance, reducing the risk of overfitting and ensuring that our evaluation was not biased by any particular subset of the data.

Typically, robust model evaluation would include a separate test set to verify that the selected hyperparameters do not lead to overfitting. However, given the nature of our dataset and the complexities involved in detecting FOG, this approach posed several challenges. A small test set could result in highly variable and unreliable outcomes, whereas allocating a large portion of the data to testing would leave insufficient data for effective model training.

Therefore, we chose to rely on cross-validation performance as our primary evaluation metric. This approach allows for the maximum utilization of available data while providing a stable and reliable assessment, ensuring that our model generalizes effectively to unseen data.

Although not the primary focus of this thesis, a comparison of the training speeds between the proposed model and a Transformer model [31] was conducted. To ensure a fair comparison, both models were trained under similar conditions, including the use of the same GPU and identical training and validation splits.

7.4.2 Validation on UCI HAR Dataset

To further evaluate the generalizability of our CNN-GRU model, we validated it on the UCI HAR dataset, a widely recognized benchmark for HAR tasks [24]. This dataset includes six distinct features, specifically 3D accelerometer and 3D gyroscope data, and is associated with six activity labels.

For this evaluation, we used the predefined training and test splits, as well as the predefined window segments, provided with the UCI HAR dataset. This ensured consistency and allowed us to compare our results with those from existing studies. By validating on this dataset, we aimed to demonstrate the model's ability to generalize across different datasets and tasks.

7.5 Results

In this section, we present the performance of our proposed model, the time-distributed CNN-GRU with residual connections, on the FOG dataset and UCI HAR dataset. We compare the results with baseline models, highlighting the effectiveness of our approach in detecting FOG events and recognizing activities from the UCI HAR dataset.

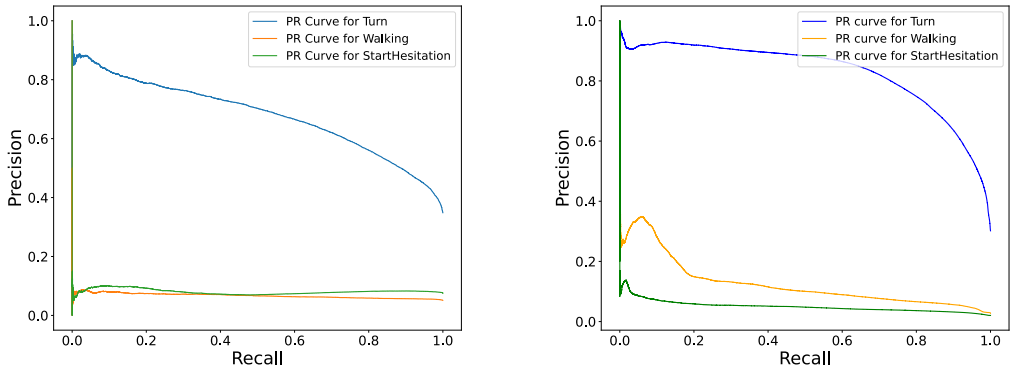
7.5.1 FOG Dataset

The performance of the models on the FOG dataset is summarized in Table 7.4, which presents mAP and AP scores for detecting specific events: turning, walking, and hesitation. The mAP score is the main metric used to evaluate model performance, while the AP scores give insight into how well each model detects individual events. It's important to note that the scores achieved by the Kaggle submissions were computed on a different test set. Therefore, these scores are presented here for reference only and should not be directly compared.

Figures 7.1a and 7.1b show the precision-recall curves for the CNN and GRU benchmark models, respectively. These curves help visualize the trade-off between precision and recall for different decision thresholds, providing a deeper understanding of how each model performs across different tasks.

TABLE 7.4: Model Performance on Different Tasks. The first rows present the scores achieved during validation by our models, while the later rows report scores from the competition leaderboard [13].

Model	mAP	AP Turn	AP Walking	AP Hesitation
CNN	0.2964	0.6603	0.0707	0.1582
GRU	0.3218	0.7845	0.1258	0.0551
CNN-GRU (No Overlap)	0.3828	0.8057	0.1189	0.2240
CNN-GRU (With Overlap)	0.4017	0.8410	0.1175	0.2467
1st place	0.5140	-	-	-
5th place	0.3898	-	-	-
10th place	0.3477	-	-	-
50th place	0.3060	-	-	-



(A) (B)

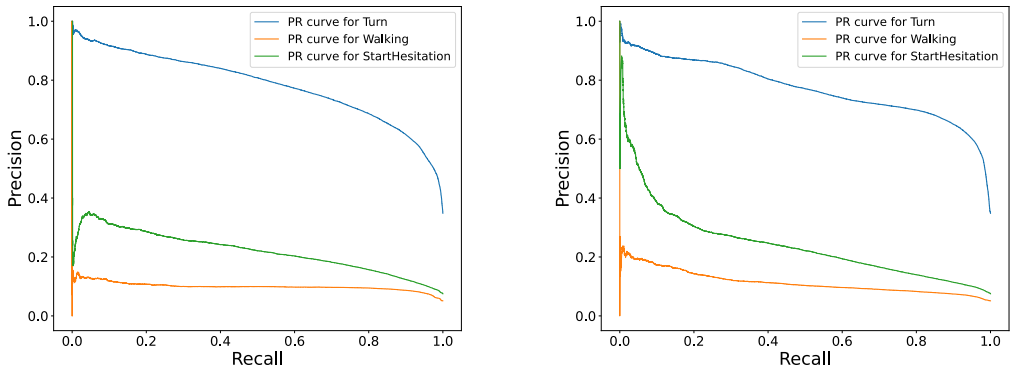
FIGURE 7.1: PR curves of the benchmark models on the FOG dataset. **A** CNN model. **B** GRU model.

Table 7.5 compares the training times of the proposed CNN-GRU variant (with overlapping sequences) and the Transformer-LSTM model that reached first place in the competition. Similar hardware, training and validation splits have been used for the measures to ensure fair comparison.

TABLE 7.5: Training Time Comparison between our best performing model and the first place submission

Model	Training Time (s)	Epochs	Time per Epoch (s)
CNN-GRU + overlap	315	15	21
Transformer-LSTM [31]	2400	20	120

Figures 7.2a and 7.2b illustrate the PR curves for the proposed CNN-GRU model without and with overlapping sequences, respectively.



(A) (B)

FIGURE 7.2: PR curves of the novel model on the FOG dataset. **A** CNN-GRU without overlap. **B** CNN-GRU with overlap.

7.5.2 UCI HAR Dataset

Table 7.6 presents a comparison of the accuracy and F1-score[1] of our model against state-of-the-art models. The scores were achieved on the official test set provided by the UCI-HAR dataset.

TABLE 7.6: Model Performance Comparison on the UCI HAR dataset

Model	Accuracy (%)	F1-score (%)
Proposed CNN-GRU	89.3	89.3
Multi-Input CNN-GRU [6]	96.20	96.19
Transformer [26]	98.6	-

7.6 Discussion

FOG detection

The results presented in Table 7.4 highlight the inherent complexity of detecting FOG events. These events require the model to extract features that are complex enough to differentiate between various types of movements but also generalizable across different patients. This balance is particularly challenging due to the class imbalance in the dataset, where models tend to be biased towards the majority classes. This difficulty was evident across all models tested during this thesis and aligns with observations from the Kaggle competition, where no solution achieved high AP scores on the minority classes, reflecting a broader challenge in this domain.

The CNN model demonstrated moderate precision in detecting `Turn` and `Hesitation` events. However, its local feature extraction approach was insufficient for learning the broader temporal context required for detecting `Walking` events, as depicted in Figure 7.1a. This limitation underscores the importance of temporal context in

[1]he F1 score is a metric that balances precision and recall by calculating their harmonic mean.

identifying this specific type of movement, which a purely spatial model like CNN cannot adequately capture.

In contrast, the GRU model, which is designed to handle temporal dependencies, showed improved performance in detecting `Walking` and `Turn` events. However, Figure 7.1b shows it struggled with detecting `Hesitation` events, suggesting that these events might rely on highly localized features that GRU alone cannot capture effectively. This finding points to the need for a model that can learn both localized and sequential patterns.

The CNN-GRU model, highlighted in Figure 7.2a, outperformed standalone CNN and GRU models by effectively combining spatial feature extraction with temporal sequence modeling. This hybrid approach improved the detection of `Hesitation` and `Walking` events, leveraging CNNs for localized features and GRUs for temporal context. However, while it showed notable gains in mAP scores, the ability of the model to detect minority events is not fully satisfactory.

Introducing sequence overlap in the CNN-GRU model further enhanced the overall mAP score. However, this enhancement did not translate into significantly better detection of `Walking` and `Hesitation` events. This indicates that while overlap improves general performance, it does not fully address the specific challenges posed by these events, likely because it fails to introduce new discriminative features. Figure 7.2b depicts the ability of the model to detect each individual event.

With a final mAP score of 0.4017, the time-distributed CNN-GRU model with residual connections would hypothetically place fifth on the official Kaggle leaderboard if it could replicate these results on the hidden test set. This performance is notable given the competition but highlights the ongoing challenge of effectively handling minority class events in FOG detection.

Given the performance boost from residual connections in the GRU layers, this approach was tested in the CNN block, but it did not yield any improvement. Similarly, experiments with class weights to counteract the dataset imbalance were also ineffective.

The computational efficiency of the CNN-GRU model, as shown in Table 7.5, remains its main advantage when compared to the Transformer model. While it may not be the top performer in terms of mAP, its faster training times and simplicity suggest that it could also offer faster inference, making it a viable candidate for real-time applications, where quick predictions are crucial.

Finally, despite its improvements over standalone CNN and GRU models and its strong precision in detecting `Turn` events, the proposed CNN-GRU model was not as effective at detecting minority events compared to the top-performing models. The superior performance of models incorporating Transformer architecture, particularly those utilizing attention mechanisms [32], suggests that these models are better equipped to capture the nuanced and diverse features required for robust FOG detection. The attention mechanism's ability to focus on relevant parts of the

input sequence allows for better generalization, which could explain the outstanding performance of such models in this context.

UCI HAR
The results for the UCI HAR dataset, as shown in Table 7.6, confirm the robustness of the proposed model. Despite being tailored for FOG detection, the CNN-GRU model achieved reasonable accuracy and F1-scores on this dataset, demonstrating its ability to generalize to different tasks and datasets without modifications to its architecture. This robustness is a strong indicator of the model's potential applicability across various domains of activity recognition.

However, it is important to note that achieving state-of-the-art results on the UCI HAR dataset was not the primary objective of this thesis. The CNN-GRU variant was heavily regularized to prevent overfitting in the FOG detection task, which may have led to suboptimal performance on this new problem. Additionally, the training techniques and hyperparameter tuning were not as advanced as those employed by the top-performing models in this field, further contributing to the performance gap.

Despite these limitations, the model's reasonable performance on the UCI HAR dataset suggests its relevance in the domain of HAR.

7.7 Threats to validity

While the proposed model demonstrates promising results, it has not been tested on the hidden Kaggle test set used for final competition scoring. As a result, the performance metrics reported in this thesis are based solely on the training and validation sets. This limitation raises the possibility that the model's performance could decrease when evaluated on the unseen test set.

Additionally, while the CNN-GRU model benefits from quicker training and inference times, it is important to acknowledge the superior performance of transformer-based models in both FOG detection and broader HAR tasks. Transformers have demonstrated a strong ability to capture complex temporal patterns and generalize across different datasets, which may give them an edge over the CNN-GRU architecture. This could limit the applicability of the CNN-GRU model, especially in scenarios where top-tier accuracy is prioritized over computational efficiency.

7.8 Conclusion

This thesis presented a novel time-distributed CNN-GRU model with residual connections and extensive regularization, designed for the detection of Freezing of Gait events in Parkinson's patients. The model aimed to leverage the strengths of Convolutional Neural Networks for spatial feature extraction and Gated Recurrent Units for temporal sequence modeling, addressing the complexity of FOG detection.

The proposed model demonstrated notable improvements over standalone CNN and GRU models, achieving clearly superior mAP score. However, the model faced

challenges in effectively detecting minority events and generalizing across different patients, particularly due to the inherent class imbalance in the dataset.

Despite these challenges, the model's computational efficiency stands out, making it a viable option for real-time applications where quick inference is essential. However, the superior performance of transformer-based models, particularly those utilizing attention mechanisms, suggests that these models may be better suited for tasks requiring top-tier accuracy.

In conclusion, while the CNN-GRU model offers a compelling balance between accuracy and computational efficiency, further refinements are necessary to fully address the challenges of FOG detection and to compete with state-of-the-art transformer models in this domain. Nonetheless, the model's robustness on the UCI HAR dataset highlights its potential applicability across various activity recognition tasks.

Chapter 8

Future Works

Although the CNN-GRU model demonstrated promising results on the training and validation sets, its performance on the hidden Kaggle test set remains unverified. This untested scenario raises concerns about the generalizability of the findings. To address this, future work should prioritize submitting the model for official evaluation on the unseen test set. This step is crucial for assessing its true performance and ensuring its robustness in real-world applications.

One promising avenue for improving the model's ability to generalize is the exploration of multi-head CNNs with different kernel sizes. By incorporating multiple convolutional layers with varying kernel sizes, the model could capture a broader range of spatial features, enhancing its ability to detect FOG events across different patients and movement types. This approach could address the limitations observed with the current CNN-GRU model, particularly in detecting minority events like `Hesitation`.

Additionally, the integration of attention mechanisms should be a key focus of future research. Attention mechanisms, which have proven effective in transformer models, allow the model to focus on the most relevant parts of the input sequence, potentially improving the detection of subtle and complex patterns associated with FOG events. Implementing such mechanisms could significantly enhance the model's ability to generalize and accurately detect FOG across diverse datasets and patient populations.

In conclusion, future work should aim to test the CNN-GRU model on the Kaggle test set, explore advanced CNN architectures with varied kernel sizes, and integrate attention mechanisms to overcome the current limitations of FOG detection.

Bibliography

[1] M.M. Bejani and M. Ghatee. "A systematic review on overfitting control in shallow and deep neural networks". In: *Artificial Intelligence Review* 54 (2021), pp. 6391–6438. DOI: 10.1007/s10462-021-09975-1.

[2] Francois Chollet and J. J. Allaire. *Deep Learning with R*. 1st. USA: Manning Publications Co., 2018. ISBN: 161729554X.

[3] D2L.ai. *Forward Propagation, Backward Propagation, and Computational Graphs*. https://d2l.ai/chapter_multilayer-perceptrons/backprop.html. Accessed: 2024-08-14. 2024.

[4] DeepAI. *Gated Recurrent Unit (GRU)*. https://deepai.org/machine-learning-glossary-and-terms/gated-recurrent-unit. Accessed: 2024-08-14. 2024.

[5] I. Dirgová Luptáková, M. Kubovčík, and J. Pospíchal. "Wearable Sensor-Based Human Activity Recognition with Transformer Model". In: vol. 22. 5. 2022, p. 1911. DOI: 10.3390/s22051911. URL: https://doi.org/10.3390/s22051911.

[6] Nidhi Dua, Shiva Singh, and Vijay Semwal. "Multi-input CNN-GRU based human activity recognition using wearable sensors". In: vol. 103. July 2021, pp. 1–18. DOI: 10.1007/s00607-021-00928-8.

[7] Weijiang Feng et al. "Audio visual speech recognition with multimodal recurrent neural networks". In: May 2017, pp. 681–688. DOI: 10.1109/IJCNN.2017.7965918.

[8] Chao Gao et al. "Freezing of gait in Parkinson's disease: pathophysiology, risk factors and treatments". In: *Translational Neurodegeneration*. Vol. 9. 1. Springer, Apr. 2020, p. 12. DOI: 10.1186/s40035-020-00191-5.

[9] D. Gupta. *Fundamentals of Deep Learning - Activation Functions and When to Use Them*. https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/. Accessed: 2024-08-14. Jan. 2020.

[10] Elke Heremans, Alice Nieuwboer, and Sarah Vercruysse. "Freezing of gait in Parkinson's disease: where are we now?" en. In: *Curr Neurol Neurosci Rep* 13.6 (June 2013), p. 350.

[11] Andrey Ignatov. "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks". In: vol. 62. Elsevier, 2018, pp. 915–922. DOI: 10.1016/j.asoc.2017.09.027. URL: https://www.sciencedirect.com/science/article/pii/S1568494617305665.

[12] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Vol. 112. Springer, 2013.

[13] Kaggle. *Leaderboard of the Kaggle Competition: Parkinson's Freezing of Gait Prediction*. https://www.kaggle.com/competitions/tlvmc-parkinsons-freezing-gait-prediction/leaderboard. Accessed: 2024-08-14. 2024.

[14] Kaggle. *Parkinson's Freezing of Gait Prediction*. https://www.kaggle.com/competitions/tlvmc-parkinsons-freezing-gait-prediction/overview. Accessed: 2024-08-14. 2024.

[15] Tianyi Liu et al. "Towards Understanding the Importance of Shortcut Connections in Residual Networks". In: *ArXiv* abs/1909.04653 (2019). URL: https://api.semanticscholar.org/CorpusID:202542887.

[16] Chandrahas Mishra and Dharmendra Lal Gupta. "Deep machine learning and neural networks: an overview". In: *International Journal of Hybrid Information Technology* 9.11 (2016), pp. 401–414.

[17] Saeed Mohsen. "Recognition of human activity using GRU deep learning algorithm". In: *Multimedia Tools and Applications* 82 (May 2023). DOI: 10.1007/s11042-023-15571-y.

[18] Moro. *20th Place Solution*. https://www.kaggle.com/competitions/tlvmc-parkinsons-freezing-gait-prediction/discussion/416106. Accessed: 2024-07-25. 2023.

[19] Alaa Nfissi et al. "CNN-n-GRU: end-to-end speech emotion recognition from raw waveform signal using CNNs and gated recurrent unit networks". In: *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2022, pp. 699–702. DOI: 10.1109/ICMLA55696.2022.00116.

[20] OpenAI. *ChatGPT: A Large Language Model*. https://chat.openai.com/. Accessed: August 14, 2024. 2024.

[21] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[22] Ravindra Parmar. "Training Deep Neural Networks". In: *Towards Data Science* (2018). Published on Sep 11, 2018. URL: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964.

[23] Mohammed Ragab et al. "A Novel One-Dimensional CNN with Exponential Adaptive Gradients for Air Pollution Index Prediction". In: vol. 12. Dec. 2020, p. 10090. DOI: 10.3390/su122310090.

[24] Anguita Davide Ghio Alessandro Oneto Luca Reyes-Ortiz Jorge and Xavier Parra. *Human Activity Recognition Using Smartphones*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C54S4K. 2012.

[25] Lamyaa Sadouk. "CNN approaches for time series classification". In: *Time series analysis-data, methods, and applications*. Vol. 5. IntechOpen, 2019, pp. 57–78.

[26] Oumaima Saidani et al. "An Efficient Human Activity Recognition Using Hybrid Features and Transformer Model". In: *IEEE Access* 11 (2023), pp. 101373–101386. DOI: 10.1109/ACCESS.2023.3314492.

[27] Hui Xing Tan et al. "Time series classification using a modified LSTM approach from accelerometer-based data: A comparative study for gait cycle detection". In: vol. 74. Elsevier, 2019, pp. 128–134. DOI: 10.1016/j.gaitpost.2019.09.007. URL: https://www.sciencedirect.com/science/article/pii/S0966636219306952.

[28] Evidently AI Team. *Accuracy vs. Precision vs. Recall in Machine Learning: What's the Difference?* https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall. Accessed: 2024-08-14. 2024.

[29] Yingjie Tian and Yuqi Zhang. "A comprehensive survey on regularization strategies in machine learning". In: *Information Fusion* 80 (2022), pp. 146–166. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2021.11.005. URL: https://www.sciencedirect.com/science/article/pii/S156625352100230X.

[30] Unknown. *Activation Function*. Accessed: 2024-08-02. n.d. URL: https://machine-learning.paperspace.com/wiki/activation-function.

[31] Baurzhan Urazalinov. *1st Place Solution*. https://www.kaggle.com/competitions/tlvmc-parkinsons-freezing-gait-prediction/discussion/416026. Accessed: 2024-07-25. 2023.

[32] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[33] Shiva Verma. "Understanding 1D and 3D Convolution Neural Network | Keras". In: *Towards Data Science* (2019). Published on Sep 20, 2019. URL: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610.

[34] Wikipedia contributors. *Bidirectional recurrent neural networks — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-July-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Bidirectional_recurrent_neural_networks&oldid=1166971172.

[35] Ian H. Witten et al. *Data Mining: Practical Machine Learning Tools and Techniques*. 4th. Morgan Kaufmann, 2016.

[36] Wanshun Wong. *What is Residual Connection?* https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55. Accessed: 2024-07-25. 2021.

[37] A. Zafar et al. "A Comparison of Pooling Methods for Convolutional Neural Networks". In: vol. 12. 17. 2022, p. 8643. DOI: 10.3390/app12178643. URL: https://doi.org/10.3390/app12178643.

[38] S Zhang et al. "Deep Learning in Human Activity Recognition with Wearable Sensors: A Review on Advances". In: vol. 22. 4. MDPI, 2022, p. 1476. DOI: 10.3390/s22041476. URL: https://doi.org/10.3390/s22041476.

[39] Mu Zhu. *Recall, precision and average precision*. Tech. rep. 2.30. Department of Statistics and Actuarial Science, University of Waterloo, 2004, p. 6.

[40] Zinxira. *4th Place Solution*. https://www.kaggle.com/competitions/tlvmc-parkinsons-freezing-gait-prediction/discussion/416410. Accessed: 2024-07-25. 2023.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**