Dyadic Synchrony Detection with a Deep Neural Network

Carleano Libretto (4325125)[1]

2023-2024

[1]Universiteit Utrecht
[2]TU Eindhoven

Applied Cognitive Psychology
Master's Thesis 27.5 ECTS

Supervisors: *Samson Chota[1], Emilia Barakova[2] & Jing Li[2]*

Universiteit Utrecht

Abstract

Background

Human interaction affects emotional health, cognitive development, and social learning. Physiological synchrony, where individuals' physiological processes mirror each other during interactions, is especially vital in parent-child relationships for fostering emotional bonding and empathy. However, analyzing heart rate variability (HRV) to understand interaction states presents challenges, particularly with neurodivergent children who show atypical responses. This study utilizes one-dimensional convolutional neural networks (1D CNNs) to enhance the analysis of these complex signals.

Methods

This study employs 1D CNNs to classify interaction states using ECG data from neurodivergent child-caregiver dyads. The approach combines hand-crafted feature extraction with supervised learning, optimizing neural network architectures to process time-series data efficiently. The model's performance is evaluated using accuracy and F1 score metrics. The neural network architecture includes layers for convolution, ReLU activation, and dense layers, concluding with a softmax output layer for classification.

Results

The neural network achieved 97% for accuracy and F1 score in ternary classification of interaction states. The results highlight the effectiveness of using deep learning for HRV analysis in this context.

Conclusions

The study demonstrated the potential of deep neural networks to develop robust, continuous, and noninvasive methods for synchrony detection. The ultimate goal is to improve the quality of life for neurodivergent children and their caregivers through enhanced understanding.

Contents

1 Background **4**

2 Methods **6**
2.1 Participants and Data Collection 6
2.1.1 Participants 6
2.1.2 Data Collection 6
2.2 ECG Data Pre-processing 7
2.2.1 Categorical Synchrony Direction 7
2.2.2 Visual Representations 8
2.3 Neural Networks 8
2.3.1 Model Overview 8
2.3.2 Convolution by Filtering and Activation 9
2.4 Convolutional neural network for signals from chest-worn sensors 10
2.5 Neural Network Training 11

3 Results **12**

4 Discussion **13**
4.1 Summary of Key Findings 13
4.2 Comparison with Previous Work 13
4.3 Theoretical Implications 13
4.4 Practical Applications 13
4.5 Limitations 13
4.6 Alternate Explanations 14
4.7 Significance of the Results 14

5 Conclusions **15**

6 References **16**

7 Appendix A **17**

8 Appendix B **22**

9 Appendix C **26**

List of Figures

1 *Pre-processing workflow adopted from Reindl et al. (2022)* 8
2 *Exemplary illustration of high synchrony for a dyad* [Synthetic data] 9
3 *Exemplary illustration of the categorical synchrony distribution* [Synthetic data] . 9
4 *1D Convolutional Neural Network Architecture* 11

List of Tables

1 *1D Convolutional Neural Network Architecture* 10
2 *Performance metrics of the 1D CNN model* 12

1 Background

The complexity of human interaction significantly influences emotional health, social learning, and cognitive development. Within this network of social signals, physiological synchrony, where individuals' physiological processes mirror each other during interactions, has become a key area of scholarly interest. This phenomenon is especially relevant in parent-child relationships, where synchrony plays a role in emotional bonding, social learning, and empathy development (Feldman, 2007). However, analyzing complex physiological signals, such as Heart Rate Variability (HRV) as measured through Interbeat Intervals (IBI), to understand and classify interaction states, particularly in dyads involving neurodivergent children, remains a challenge.

HRV, which is derived from the variation in time intervals between consecutive heartbeats, is a non-invasive measure that reflects the autonomic nervous system's regulatory dynamics. It is widely recognized as a marker of emotional and physiological states, providing insight into an individual's emotional landscape and autonomic control (Thayer et al., 2012). In the context of parent-child interactions, HRV synchrony, as observed through aligned IBIs, is believed to indicate a harmonious connection, with significant implications for the child's social and emotional development (Porges, 2007). However, classifying interaction states—synchrony, desynchrony, and asynchrony—using ECG data in dyads involving neurodivergent children is complex. Children with conditions such as autism spectrum disorders (ASD) and attention deficit hyperactivity disorder (ADHD) often exhibit atypical physiological and emotional regulation patterns. These atypical patterns complicate the establishment and interpretation of synchrony, making traditional classification methods less effective and often unreliable (Porges et al., 1996).

The advent of computational and machine learning approaches has introduced new possibilities for analyzing physiological data within the health and behavioral sciences. However, there is still a significant gap in the application of deep learning models, particularly 1D CNNs, for the nuanced task of ternary synchrony detection in dyadic interactions involving neurodivergent children. Neural networks have the capacity to efficiently process and interpret time-series data like ECG signals, distinguishing complex physiological states. This capability represents a new frontier in this research domain (Kiranyaz et al., 2016).

This study aims to bridge this gap by developing a deep 1D CNN model to analyze HRV time-series data for synchrony detection in parent-neurodivergent child interactions. The deep learning approach, embodied by the 1D CNN, leverages its proven effectiveness in managing time-series data across various fields, including speech recognition and biomedical signal analysis, to improve the accuracy and reliability of interaction state classifications within these unique dyads (LeCun et al., 1998; Zheng et al., 2014). This effort is expected to fill a void in scientific understanding and contribute towards providing a scalable, nuanced tool for researchers and practitioners, facilitating more personalized and effective interventions for neurodivergent children and their families.

HRV's role extends beyond its function as a health indicator, encapsulating the dynamics of human emotions and social interactions. Studies suggest that HRV synchrony between individuals can serve as a non-verbal communication form, aligning with theories of emotional contagion and the mirror neuron system, highlighting the potential of HRV analysis in enhancing our understanding of human emotional and social exchange (Hatfield et al., 1993; Gallese et al., 1996). The significance of HRV in these areas is particularly pronounced in parent-child interactions, where it is considered a fundamental component of emotional bonding and attachment, critical to the child's capacity for emotional regulation and social development (Feldman, 2007).

Investigating interactions between caregivers and neurodivergent children presents distinct challenges due to their unique patterns of emotional and physiological regulation. The social communication difficulties and diverse emotional responses often associated with neurodivergence tempt a reassessment of traditional methods. These methods typically rely on observable behavioral cues and self-reported measures, which may not fully capture the complexities of communication

dynamics within these dyads (Schoen et al., 2009; Lord et al., 2012).

Deep learning models, especially 1D CNNs, offer a promising approach for analyzing physiological signals, including HRV. These models are well-suited, for this task because they can handle the complex, non-linear nature of physiological data over time. 1D CNNs can automatically learn hierarchical patterns within the data, allowing them to distinguish subtle and intricate features (Goodfellow et al., 2016). In the context of HRV analysis for neurodivergent child-parent pairs, deep learning models have the potential to reveal subtle physiological signs of synchrony that may not be identified using traditional analysis methods.

Analyzing HRV synchrony can provide valuable insights into the physiological mechanisms underlying interactions between individuals. This knowledge has the potential to significantly impact therapeutic and educational approaches for neurodivergent children and their families. By understanding the physiological underpinnings of these interactions, researchers can develop personalized interventions tailored to the specific needs of each child and family. These interventions could target improved communication and emotional regulation. Furthermore, the findings of this research can inform the creation of tools and resources for parents, educators, and therapists. These resources could promote the development of supportive and understanding environments that nurture positive interactions and emotional connections between neurodivergent children and their caregivers (Rogers et al., 2014).

This exploration clarifies the role of HRV in studying human interactions, the challenges of interactions within neurodivergent child-parent dyads, the potential of deep learning models in this research, and the implications for developing targeted interventions. Advancing a deep 1D CNN model for HRV synchrony detection aims to address a gap in our understanding of these interactions and contribute to the well-being of neurodivergent children and their families. This effort, grounded in a multidisciplinary approach, demonstrates the potential of combining computational methods with psychological and physiological research to enhance our understanding of human interactions.

2 Methods

The study employed an approach that combined hand-crafted feature extraction with supervised learning using 1D CNNs.

The performance of the model was evaluated based on metrics: Accuracy and the F1 Score. Accuracy measures the overall correctness of the model across all classes, while the F1 Score provides a balance between precision and recall. The formulas for these metrics are provided below:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

2.1 Participants and Data Collection

2.1.1 Participants

The study recruited 10 parent-child dyads, including children 6-12 years old with autism spectrum disorders (ASD), to examine physiological synchrony during homework sessions. The inclusion criteria for children included a formal diagnosis of ASD for the neurodivergent group. Parents were required to be the primary caregivers, fluent in the language of the study, and without any severe mental health conditions that could impact the interaction dynamics.

2.1.2 Data Collection

Data collection involved recording electrocardiogram (ECG) signals from both the child and the parent using Polar H10 heart rate monitors. These devices were chosen for their reliability and accuracy in capturing heart activity. The procedure began by moistening the electrode area of the strap with room-temperature water to ensure good conductivity. The strap was then fastened around the participant's chest, ensuring that the electrode area was centered and the strap was tight enough to stay in place throughout the session. Adjustments were made for smaller children by tying a knot or using an extra clip to secure the strap properly.

Once the straps were fitted, the Polar H10 devices were connected to a mobile application, ECG Logger (ECG Logger for Polar H10, ECG Logger, Finland), via Bluetooth. This application was used to start and monitor the ECG recordings. Both the child and the parent were seated comfortably in chairs to minimize movement artifacts during the recording.

The data collection session comprised a 5-minute benchmarking period followed by a 30-minute homework session. The benchmarking period allowed for the stabilization of physiological signals, ensuring that subsequent measurements were not influenced by initial adjustments or environmental changes. During the homework session, both the ASD child and the parent engaged in various assignments such as cutting paper, color painting, or solving math questions. This design aimed to simulate real-life interactive tasks that require cooperation and communication, thus providing a naturalistic setting for studying physiological synchrony.

Throughout the 30-minute session, the interaction between the child and the parent was unstructured, allowing for natural interaction patterns to emerge. This approach was intended to

capture a wide range of synchrony dynamics, reflecting the variability in everyday parent-child interactions.

Upon completion of the session, the recordings were saved as CSV files with a sampling frequency of 130 Hertz (Hz) in the ECG Logger app. These files were then transferred to a computer for further analysis. The raw ECG data were processed using Python (Appendix A).

2.2 ECG Data Pre-processing

An important step in this study focused on establishing a target variable, the categorical synchrony direction. This variable was derived through a sequence of processing steps designed to capture the dynamics of physiological synchrony in dyadic interactions. The methodology employed was based on established scientific practices, in attempt to ensure reliability in the classification of interaction states.

2.2.1 Categorical Synchrony Direction

The derivation of the categorical synchrony direction target variable, which classified interactions into synchrony, desynchrony, or asynchrony, involved several precise steps:

1. Detection of R Peaks: The process began with the detection of R peaks in the ECG signal. R peaks represent the apex of the QRS complex in an ECG signal, and their accurate detection is crucial for deriving HRV measures such as IBI. This was performed using an automated algorithm capable of handling the noise and artifacts typically present in physiological data, as detailed by Reindl et al. (2022). Manual correction of R peaks and removal of artifacts were performed as necessary to ensure accuracy.

2. Epoch Segmentation: Once R peaks were detected, the ECG signal was segmented into 2-second epochs. Epoch segmentation divides the continuous ECG signal into shorter, manageable segments, allowing for detailed and localized analysis of interaction intervals. The epoch size of 2 seconds was selected based on the minimal amount of time required to reliably approximate the heart rate (Helm et al., 2018). This technique, facilitates the isolation of specific intervals where synchrony, desynchrony, or asynchrony can be analyzed effectively.

3. Extraction of IBIs: The next step involved calculating the average IBI for each epoch across all participants. This created a series of average IBIs for each participant over time. Since cleaning the initial IBI data removed some data points, these gaps (missing values) were filled in using cubic spline interpolation. However, if a participant (either adult or child) had more than 5% of their data missing within a single recording session, the entire recording (including both participants) was excluded from that particular experimental condition. The resulting IBI series data was then prepared for further processing.

4. Detrending: The IBI series data was detrended using second-order polynomial regression to remove any linear or quadratic trends that could obscure the true variability and synchrony patterns. This detrending step ensures that the analysis focuses on the genuine fluctuations in HRV rather than any underlying trends that might distort the results.

5. ARIMA Modeling: To refine the data further, ARIMA (AutoRegressive Integrated Moving Average) modeling was applied to the detrended IBI values with one autoregressive term, one moving average term, and integrated noise. ARIMA modeling is used to analyze and forecast time-series data by removing any remaining autocorrelations in the ECG data. This results in residuals that better represent the true synchrony between dyad members. The choice of ARIMA model parameters was based on the partial autocorrelation functions (PACF) of the detrended IBI series, which indicated a strong autocorrelative component at lag 1.

6. Cross-Correlation Calculation: The next step involved computing the cross-correlation between the ARIMA residuals of the two individuals in the dyads. Cross-correlation measures the similarity between two time series as a function of the lag of one relative to the other. This

technique quantifies the degree of synchrony between the dyad members by identifying how well their HRV signals align over time. Higher cross-correlation values indicate higher synchrony, whereas lower values indicate desynchrony or asynchrony.

7. Classification of Epochs: The correlated values for each epoch were extracted, and thresholds were set at the 33rd and 66th percentiles of these values. Epochs with cross-correlation values above the 66th percentile were classified as synchrony, those below the 33rd percentile as desynchrony, and those in between as asynchrony. These thresholds were chosen based on their ability to optimally distinguish between the three interaction states. This method provides a ternary classification that captures the nuanced dynamics of the interactions (Appendix A & C).

2.2.2 Visual Representations

The workflow described above is visualized in several figures:

1. Pre-processing Workflow:

Figure 1 illustrates the overall pre-processing workflow adapted from Reindl et al. (2022).

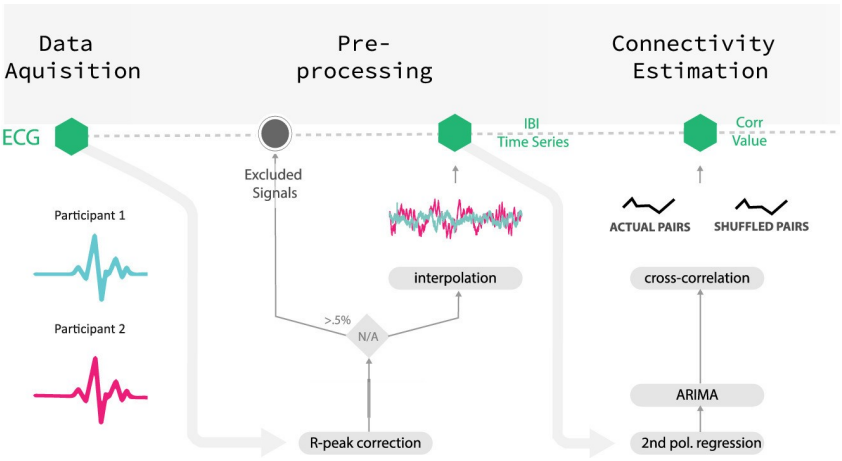

Figure 1: *Pre-processing workflow adopted from Reindl et al. (2022)*

2. Post-Detrending Signal:

Figure 2 shows an exemplary illustration of the IBI signal after detrending.

3. Categorical Synchrony Distribution:

Figure 3 depicts the distribution of the categorical synchrony classification, illustrating the thresholds used for classification.

2.3 Neural Networks

The architecture of 1D CNNs is designed to mimic the human visual cortex, enabling the effective processing of time-series data through several layers of convolution and activation. These layers facilitate the extraction of significant features for pattern recognition in physiological data (Goodfellow et al., 2016).

2.3.1 Model Overview

The neural network model utilized in this study consisted primarily of convolutional layers, which are the building blocks of CNNs. These layers applied filters to the input data, which allowed the network to perform convolution operations that capture temporal dependencies and patterns within the ECG data. Each convolution layer was followed by an activation function, typically the Rectified Linear Unit (ReLU), which introduced non-linearity into the model.

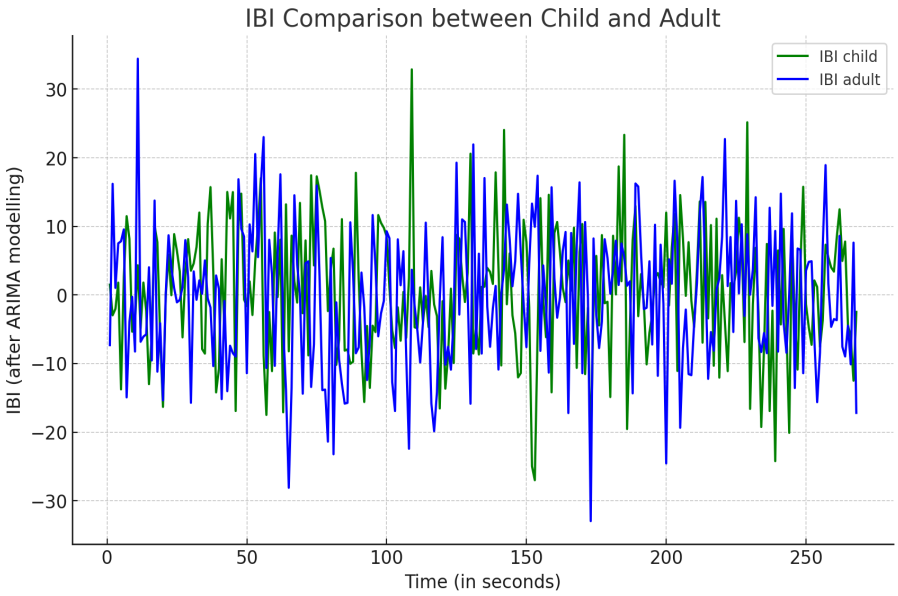

Figure 2: *Exemplary illustration of high synchrony for a dyad* [Synthetic data]

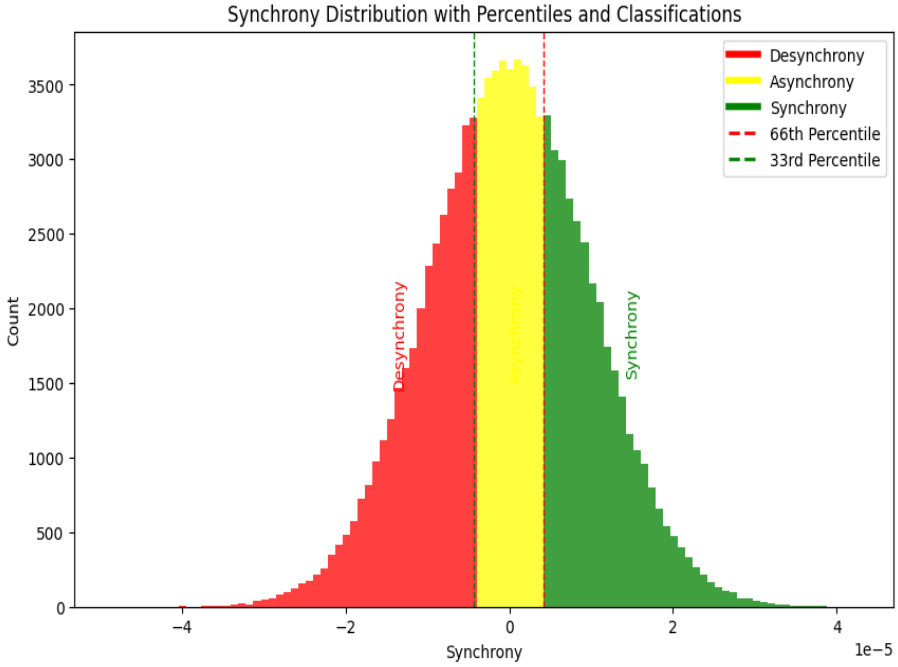

Figure 3: *Exemplary illustration of the categorical synchrony distribution* [Synthetic data]

2.3.2 Convolution by Filtering and Activation

Mathematically, a convolution operation in a CNN can be represented as:

$$z^{[l]} = w^{[l]} \cdot a^{[l]} + b^{[l]} \tag{5}$$

Symbol	Description
$z^{[l]}$	Output of the current layer (l)
$w^{[l]}$	Filter weights of the current layer
$a^{[l]}$	Activation of the previous layer
$b^{[l]}$	Bias term of the current layer

The activation function, often the ReLU function, was then applied to the output of the convolution:

$$z^{[l]} = w^{[l]} \cdot a^{[l]} + b^{[l]} \tag{6}$$

$$a^{[l]} = g\left(z^{[l]}\right) \tag{7}$$

$$g(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{8}$$

Symbol	Description
$g(z)_j$	Output probability for class j
z_j	Output of the j-th unit in the output layer
K	Total number of classes in the classification task

The softmax function provided a probabilistic interpretation of the model's output, which made it easier to predict the category with the highest probability.

2.4 Convolutional neural network for signals from chest-worn sensors

The CNN was built to take in and analyze handcrafted features from physiological signals from a human chest-mounted electrocardiogram (ECG) sensor. With the input divided into segments of 2 seconds, the network performs ternary synchrony detection in parent-child dyads, categorizing three distinct states: asynchrony, desynchrony, and synchrony.

Layer Type	**Details**
Conv1D	Filters: 64, Filter Size: 3, Activation: ReLU
Conv1D	Filters: 64, Filter Size: 3, Activation: ReLU
MaxPool1D	Pool Size: 2, Strides: 2
Flatten	Converts multi-dimensional input into a single dimension
Dense (Layer 1)	Units: 128, Activation: ReLU
Dense (Layer 2)	Units: 64, Activation: ReLU
Output Dense	Units: 3, Activation: Softmax

Table 1: *1D Convolutional Neural Network Architecture*

As illustrated in Table 1, the initial layer of the model was a 1D convolutional layer, which extracted features from the input data using 64 filters, each with a size of 3. The ReLU (Rectified Linear Unit) activation function was applied to the output of each filter, adding non-linearity to the model. This was followed by another 1D convolutional layer, also with 64 filters of size 3 and ReLU activation, and further refined the features extracted by the first convolutional layer.

Next, a max pooling layer with a pool size of 2 and strides of 2 was applied. This layer reduced the dimensionality of the feature maps, and retained the most significant information while it reduced computational complexity.

The subsequent layer was a flattening layer, which converted the multi-dimensional output from the previous layers into a one-dimensional vector. This flattened output served as the input for the dense layers that followed.

The model then included two dense layers, also known as fully connected layers. The first dense layer comprised 128 units, and the second contained 64 units. Both layers used the ReLU activation function to enable the model to learn complex patterns from the data.

The final layer was an output dense layer with 3 units, corresponding to the three predicted categories. This layer utilized the softmax activation function, ideal for multi-class classification

problems. The softmax function converted the raw output scores into probabilities for each class, and allowed the model to predict the most likely category for a given input.

The model was compiled using the Adam optimizer, with categorical cross-entropy as the loss function and accuracy as the performance metric (see Appendix B).

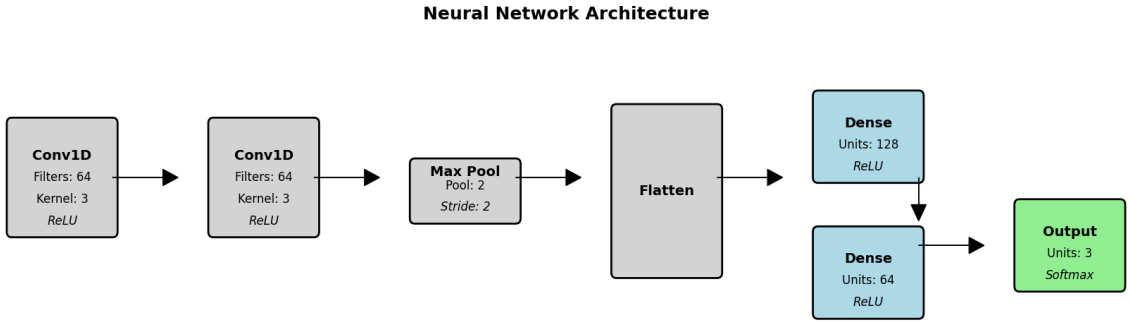

Figure 4: *1D Convolutional Neural Network Architecture*

Figure 4 illustrates the overall architecture of the proposed deep 1D convolutional neural network.

2.5 Neural Network Training

Training Element	Description
Optimizer	Adam
Loss Function	Categorical Cross Entropy
Epochs	100
Batch Size	40
Validation	Tenfold Cross Validation
Software	Keras

3 Results

The efficacy of the 1D CNN for classifying interaction states of synchrony, desynchrony, and asynchrony using ECG data was evaluated using accuracy, precision, recall, and the F1 score. Table 2 presents the performance metrics of the model:

Metric	**Value**
Median Accuracy	0.966126
Median F1 Score	0.965733
Median Recall	0.965395
Median Precision	0.966210

Table 2: *Performance metrics of the 1D CNN model*

The median accuracy of 0.966 indicates the model's high overall effectiveness in classifying interaction states. Precision, recall, and F1 score all hover around 0.966, reflecting the model's balanced performance across these critical measures. These metrics collectively suggest that the model reliably distinguishes between synchrony, desynchrony, and asynchrony with minimal errors.

4 Discussion

4.1 Summary of Key Findings

The primary findings indicate that the 1D CNN model is highly effective in classifying the states of interaction in dyadic HRV data. The consistency across accuracy, precision, recall, and F1 score suggests that the model performs well in both identifying true positive instances and minimizing false negatives.

4.2 Comparison with Previous Work

Our results align with previous research that has demonstrated the potential of deep learning models in handling time-series physiological data. For instance, Kiranyaz et al. (2016) highlighted the efficacy of convolutional neural networks in processing and classifying ECG signals at an individual level with the proposed approach achieving median 98.1% accuracy. The high accuracy and robustness of our model corroborate these findings, extending the applicability of deep learning techniques to the analysis of dyadic interactions involving neurodivergent children.

4.3 Theoretical Implications

The findings of this study have a couple of theoretical implications. First, they support the notion that physiological synchrony can be effectively quantified and classified using advanced computational models. This provides a robust framework for investigating the underpinnings of social and emotional interactions, particularly in parent-child dyads involving neurodivergent children.

Second, the success of the 1D CNN model underscores the potential of deep learning in uncovering subtle and complex patterns in dyadic physiological data. This aligns with theories of emotional contagion and the mirror neuron system, which suggest that physiological synchrony reflects deeper emotional and cognitive connections between individuals (Hatfield et al., 1993; Gallese et al., 1996). The ability to accurately classify synchrony states offers new avenues for exploring these theories in more detail.

4.4 Practical Applications

From a practical perspective, the model's high performance has implications for therapeutic and educational settings. Accurate classification of interaction states can inform the development of personalized intervention strategies aimed at enhancing communication and emotional regulation in neurodivergent children. For example, therapists and educators can use real-time feedback from the model to adjust their approaches, ensuring that interventions are tailored to the unique needs of each child.

Moreover, the model can be integrated into wearable devices, providing continuous monitoring of physiological synchrony during daily interactions. This can offer valuable insights into the child's social and emotional development, enabling parents and caregivers to respond more effectively to their needs.

4.5 Limitations

Despite the promising results, several limitations need to be addressed. One significant limitation is the potential impact of noise and artifacts in ECG data. Although preprocessing steps were employed to mitigate these issues, residual noise could still affect the model's performance. Future

research should explore more sophisticated noise reduction techniques to further enhance the model's accuracy.

Another limitation is the model's sensitivity to epoch segmentation size. The choice of epoch segment size influences the granularity and sensitivity of synchrony detection. Shorter segments may capture transient changes but include more noise, while longer segments provide a more stable signal but may miss rapid changes. Finding the optimal balance between these trade-offs is key for improving model performance.

Additionally, the study sample consisted of dyads with 10 high-functioning children with Autism Spectrum Disorder (ASD) aged 6-12. While this provided a representative sample of the target population, the findings may not generalize to other age groups, levels of neurodivergence, or different cultural and socio-economic backgrounds. Future studies should include a more diverse sample to enhance the generalizability of the results.

4.6 Alternate Explanations

While the 1D CNN model demonstrated high performance, it is important to consider possible alternate explanations for the results. The model's success could be attributed to overfitting to the inherent characteristics of the dataset, such as the specific patterns of ECG data in the studied population.

Further research is needed to validate the model across larger datasets and different settings to confirm its robustness and applicability. Exploring the impact of different preprocessing techniques and model architectures will also enable optimization the model's performance.

4.7 Significance of the Results

The significance of these results lies in the advancement of our understanding of dyadic interactions involving neurodivergent children. By leveraging deep learning techniques, this study provides a novel approach to analyzing dyadic ECG data, offering more accurate and reliable classifications of interaction states. This contributes to the field of psychology broadly and opens new avenues for research and practical applications.

The ability to accurately classify synchrony, desynchrony, and asynchrony states has profound implications for understanding the dynamics of parent-child interactions. It allows researchers to explore the impact of physiological synchrony on emotional bonding, social learning, and cognitive development. Furthermore, the insights gained from this study can inform the design of interventions aimed at enhancing the quality of life for neurodivergent children and their families.

5 Conclusions

The 1D CNN model developed in this study demonstrated high accuracy, precision, recall, and F1 score in classifying interaction states using ECG data. The results confirm the model's robustness and reliability, highlighting its potential for practical applications in therapeutic and educational settings. Despite some limitations, the findings provide a foundation for future research and underscore the potential of deep learning in understanding physiological synchrony in dyadic interactions.

This study contributes to the existing body of knowledge by offering a novel approach to analyzing dyadic ECG data, addressing a significant gap in the current methodologies. The integration of advanced computational techniques with psychological and physiological research enhances our ability to comprehend and support the unique needs of neurodivergent children and their families. The implications of this work extend beyond the immediate context, providing insights for applications in health, psychology, and education.

6 References

1. Feldman, R. (2007). Parent-infant synchrony and the construction of shared timing: Physiological precursors, developmental outcomes, and risk conditions. *Journal of Child Psychology and Psychiatry, 48*(3-4), 329-354.

2. Gallese, V., Fadiga, L., Fogassi, L., & Rizzolatti, G. (2004). Action recognition in the premotor cortex. *Brain, 119*(2), 593-609.

3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* (Vol. 1). MIT Press.

4. Hatfield, E., Cacioppo, J. T., & Rapson, R. L. (1993). Emotional contagion. *Current Directions in Psychological Science, 2*(3), 96-100.

5. Helm, J. L., Miller, J. G., Kahle, S., Troxel, N. R., & Hastings, P. D. (2018). On measuring and modeling physiological synchrony in dyads. *Multivariate Behavioral Research, 53*, 521-543. https://doi.org/10.1080/00273171.2018.1459292

6. Kiranyaz, S., Ince, T., & Gabbouj, M. (2016). Real-time patient-specific ECG classification by 1-D convolutional neural networks. *IEEE Transactions on Biomedical Engineering, 63*(3), 664-675.

7. LeCun, Y., Bengio, Y., & Hinton, G. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

8. Lord, C., Rutter, M., DiLavore, P. C., Risi, S., Gotham, K., & Bishop, S. L. (2012). *Autism diagnostic observation schedule, (ADOS-2) manual (Part I): Modules 1-4*. Torrance, CA: Western Psychological Services.

9. Porges, S. W. (2007). The polyvagal perspective. *Biological Psychology, 74*(2), 116-143.

10. Porges, S. W., Doussard-Roosevelt, J. A., Portales, A. L., & Greenspan, S. I. (1996). Infant regulation of the vagal "brake" predicts child behavior problems: A psychobiological model of social behavior. Developmental Psychobiology, 29(8), 697-712.

11. Reindl, V., et al. (2022). Multimodal hyperscanning reveals that synchrony of body and mind are distinct in mother-child dyads. *NeuroImage, 251*, 118982. https://doi.org/10.1016/j.neuroimage.20

12. Rogers, S. J., Vismara, L. A., Wagner, A. L., McCormick, C., Young, G., & Ozonoff, S. (2014). Autism treatment in the first year of life: A pilot study of Infant Start, a parent-implemented intervention for symptomatic infants. *Journal of Autism and Developmental Disorders, 44*(12), 2981-2995.

13. Schoen, S. A., Miller, L. J., Brett-Green, B., & Nielsen, D. M. (2009). Physiological and behavioral differences in sensory processing: A comparison of children with Autism Spectrum Disorder and Sensory Modulation Disorder. *Frontiers in Integrative Neuroscience, 3*, 29.

14. Thayer, J. F., Åhs, F., Fredrikson, M., Sollers, J. J., & Wager, T. D. (2012). A meta-analysis of heart rate variability and neuroimaging studies: Implications for heart rate variability as a marker of stress and health. *Neuroscience & Biobehavioral Reviews, 36*(2), 747-756.

15. Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classification using multi-channels deep convolutional neural networks. In F. Li, G. Li, S. Hwang, B. Yao, & Z. Zhang (Eds.), *Web-Age Information Management. WAIM 2014. Lecture Notes in Computer Science* (Vol. 8485). Springer, Cham.

7 Appendix A

Python code that encapsulates the data preprocessing and produces the dataset the model learned on.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import heartpy as hp
import neurokit2 as nk
from scipy.signal import resample, correlate, correlation_lags
from scipy.interpolate import CubicSpline
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import datetime
import os
import re

def parse_datetime_from_filenames(ecg_path, benchmark_path):
    """Parse date from ECG filename and time from benchmark filename to form a
                                            complete datetime object."""
    # Extract the date from the ECG filename
    date_match = re.search(r"(\d{4}-\d{2}-\d{2})", ecg_path)
    if not date_match:
        raise ValueError("Date not found in the ECG file path")
    date_str = date_match.group(1)

    # Extract the time from the benchmark filename
    time_match = re.search(r"(\d{2})_(\d{2})_(\d{2})", benchmark_path)
    if not time_match:
        raise ValueError("Time not found in the benchmark file path")
    time_str = f"{time_match.group(1)}:{time_match.group(2)}:{time_match.group(
                                            3)}"

    # Combine date and time into a single datetime object
    datetime_str = f"{date_str} {time_str}"
    datetime_obj = datetime.datetime.strptime(datetime_str, "%Y-%m-%d %H:%M:%S"
                                            )

    # Convert datetime object to Unix epoch nanoseconds
    epoch_ns = int(datetime_obj.timestamp() * 1e9)  # 1 second = 1e9
                                            nanoseconds
    return datetime_obj, epoch_ns

def prep_and_get_peaks(data, sample_rate):
    '''Preprocess and detect peaks with NeuroKit2.'''
    # Clean ECG signal
    cleaned = nk.ecg_clean(data, sampling_rate=sample_rate, method='vg')
    # Detect peaks
    signals, info = nk.ecg_peaks(cleaned, sampling_rate=sample_rate, method="vg
                                            ")
    return signals, info

def epoch(data, samples_per_epoch):
    """Divide the data into epochs."""
    num_epochs = len(data) // samples_per_epoch
    epochs = [data[i * samples_per_epoch:(i + 1) * samples_per_epoch] for i in
                                            range(num_epochs)]
    return epochs
```

```
def calculate_epoch_means(epochs):
    '''Calculate mean IBI for each epoch.'''
    return [epoch.mean() for epoch in epochs if not epoch.isnull().all()]

def interpolate_missing_values(ibi_series):
    '''Interpolate missing values in IBI series with cubic spline interpolation
                                                .'''
    # Create a series with an index that spans the full range
    full_index_series = pd.Series(index=np.arange(ibi_series.index[0],
                                                  ibi_series.index[-1] + 1))
    combined_series = pd.concat([ibi_series, full_index_series], axis=1).iloc[:
                                                  , 0]

    # Interpolate missing values
    non_missing_index = combined_series.notnull()
    if sum(non_missing_index) > 2:  # Cubic spline requires at least 4 points
        cs = CubicSpline(combined_series.index[non_missing_index],
                                                    combined_series[
                                                    non_missing_index])
        interpolated_series = pd.Series(cs(combined_series.index), index=
                                                    combined_series.index)
        return interpolated_series
    else:
        return combined_series  # If not enough points for cubic spline, return
                                                     original

def exclude_excessive_missing_data(participant_epochs, threshold=0.05):
    '''Exclude epochs with excessive missing data.'''
    valid_epochs = []
    for epoch in participant_epochs:
        if isinstance(epoch, np.ndarray):
          epoch = pd.Series(epoch)
        missing_percentage = epoch.isnull().mean()
        if missing_percentage <= threshold:
            valid_epochs.append(epoch)
    return valid_epochs

def detrend_data(epoch_means):
    """Detrend epoch means time series by removing a second order polynomial
                                                fit."""
    # Ensure the data is in a proper format (removing any NaNs and resetting
                                                indices)
    y = np.array(epoch_means)
    x = np.arange(len(y))

    # Fit a second-order polynomial (Polynomial of degree 2)
    coeffs = np.polyfit(x, y, 2)
    trend = np.polyval(coeffs, x)
    detrended = y - trend
    return detrended

def fit_arima(series, order):
    """Fit an ARIMA model to the series and return the residuals."""
    model = ARIMA(series, order=order)
    model_fit = model.fit()
    return model_fit.resid

def compute_epoch_cross_correlation(series1, series2):
    if len(series1) != len(series2):
```

```
        raise ValueError("Both series must have the same length.")
    num_epochs = len(series1)
    max_correlations = []
    # Iterate over each residual value corresponding to an epoch
    for i in range(num_epochs):
        residual1 = np.array([series1[i]])
        residual2 = np.array([series2[i]])
        corr = correlate(residual1, residual2, mode='full')
        max_idx = np.argmax(corr)
        max_corr = corr[max_idx]
        max_correlations.append(max_corr)
    return max_correlations

def calculate_epoch_features(data, samples_per_epoch):
    """Calculate RMSSD (Root Mean Square of Successive Differences) for each
                                                epoch."""
    num_epochs = len(data) // samples_per_epoch
    rmssd_values = []
    for i in range(num_epochs):
        # Extract epoch data
        start_index = i * samples_per_epoch
        end_index = start_index + samples_per_epoch
        epoch_data = data[start_index:end_index]
        # Ensure there are enough points to compute successive differences
        if len(epoch_data) > 1:
            # Calculate the successive differences
            successive_diffs = np.diff(epoch_data)
            # Compute RMSSD
            rmssd = np.sqrt(np.mean(successive_diffs ** 2))
        else:
            rmssd = np.nan  # If insufficient data points, return NaN
        rmssd_values.append(rmssd)
    return rmssd_values

def find_largest_file(directory, keyword):
    """Find the largest file in the directory containing a specific keyword in
                                                its name."""
    files = [f for f in os.listdir(directory) if keyword.lower() in f.lower()]
    if not files:
        raise FileNotFoundError(f"No file found in the directory containing the
                                                    keyword '{keyword}'")
    largest_file = max(files, key=lambda f: os.path.getsize(os.path.join(
                                                directory, f)))
    return os.path.join(directory, largest_file)

def process_group_data(asd_group_path, sample_rate, epoch_ms,
                                            more_synchrony_threshold_pct,
                                            less_synchrony_threshold_pct):
    """Process data for a specific ASD group."""
    child_ecg_path = find_largest_file(asd_group_path, 'child')
    parent_ecg_path = find_largest_file(asd_group_path, 'parent')
    benchmark_file = [f for f in os.listdir(asd_group_path) if f.startswith("
                                                start")][0]
    benchmark_path = os.path.join(asd_group_path, benchmark_file)


    benchmark_start_obj, benchmark_start_time_ns =
                                                parse_datetime_from_filenames(
                                                child_ecg_path, benchmark_path)
```

```
child_df = pd.read_csv(child_ecg_path)
parent_df = pd.read_csv(parent_ecg_path)

child_start_index = (child_df['time'] >= benchmark_start_time_ns).idxmax()
parent_start_index = (parent_df['time'] >= benchmark_start_time_ns).idxmax
                                          ()

child_data_synced = child_df[child_start_index:]
parent_data_synced = parent_df[parent_start_index:]

min_length = min(len(child_data_synced), len(parent_data_synced))
child_data_synced = child_data_synced.iloc[:min_length]
parent_data_synced = parent_data_synced.iloc[:min_length]

child_ecg = child_data_synced['ecg'].values
parent_ecg = parent_data_synced['ecg'].values

samples_per_epoch = int(sample_rate * (epoch_ms / 1000))

child_features = calculate_epoch_features(child_ecg, samples_per_epoch)
parent_features = calculate_epoch_features(parent_ecg, samples_per_epoch)

child_signals, _ = prep_and_get_peaks(child_ecg, sample_rate)
parent_signals, _ = prep_and_get_peaks(parent_ecg, sample_rate)

child_ibi = child_signals['ECG_R_Peaks']
parent_ibi = parent_signals['ECG_R_Peaks']

child_ibi_interpolated = interpolate_missing_values(child_ibi)
parent_ibi_interpolated = interpolate_missing_values(parent_ibi)

child_epochs = epoch(child_ibi_interpolated, samples_per_epoch)
parent_epochs = epoch(parent_ibi_interpolated, samples_per_epoch)

child_valid_epochs = exclude_excessive_missing_data(child_epochs)
parent_valid_epochs = exclude_excessive_missing_data(parent_epochs)

child_epoch_means = calculate_epoch_means(child_valid_epochs)
parent_epoch_means = calculate_epoch_means(parent_valid_epochs)

child_epoch_means_detrended = detrend_data(child_epoch_means)
parent_epoch_means_detrended = detrend_data(parent_epoch_means)

child_residuals = fit_arima(child_epoch_means_detrended, order=(1, 0, 1))
parent_residuals = fit_arima(parent_epoch_means_detrended, order=(1, 0, 1))

max_correlations = compute_epoch_cross_correlation(parent_residuals,
                                          child_residuals)
more_synchrony_threshold = np.percentile(max_correlations,
                                          more_synchrony_threshold_pct)
less_synchrony_threshold = np.percentile(max_correlations,
                                          less_synchrony_threshold_pct)
num_epochs = min(len(child_features), len(max_correlations))

data = pd.DataFrame({
    "Time": pd.date_range(start=benchmark_start_obj, periods=num_epochs,
                                              freq='2S'),
    "Child RMSSD": child_features[:num_epochs],
    "Parent RMSSD": parent_features[:num_epochs],
    "Child IBI": [np.nanmean(epoch) if len(epoch) > 0 else np.nan for epoch
```

```
                                                in child_valid_epochs][:
                                               num_epochs],
            "Parent IBI": [np.nanmean(epoch) if len(epoch) > 0 else np.nan for
                                               epoch in parent_valid_epochs][:
                                               num_epochs],
            "Child Detrended IBI": child_epoch_means_detrended[:num_epochs],
            "Parent Detrended IBI": parent_epoch_means_detrended[:num_epochs],
            "Child ARIMA Residuals": child_residuals[:num_epochs],
            "Parent ARIMA Residuals": parent_residuals[:num_epochs],
            "Max Correlations": max_correlations[:num_epochs],
            "Sync Direction": ["Synchrony" if corr >= more_synchrony_threshold else
                                               "Desynchrony" if corr <=
                                               less_synchrony_threshold else "
                                               Asynchrony" for corr in
                                               max_correlations[:num_epochs]],
        })
    return data

# Aggregating results from multiple ASD groups
base_directory = "/content/Sample"
asd_group_directories = [os.path.join(base_directory, d) for d in os.listdir(
                                        base_directory) if os.path.isdir(os.
                                        path.join(base_directory, d))]

all_data = []

sample_rate = 130
epoch_ms = 2000
more_synchrony_threshold_pct = 66
less_synchrony_threshold_pct = 33

for asd_group in asd_group_directories:
    group_data = process_group_data(asd_group, sample_rate, epoch_ms,
                                            more_synchrony_threshold_pct,
                                            less_synchrony_threshold_pct)
    all_data.append(group_data)

data = pd.concat(all_data, ignore_index=True)

data.to_excel("Dyadic_dataset_iter2.xlsx")
```

8 Appendix B

Python code for creating 1D CNN, running the model on the data, and displaying the training results.

```
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import Callback
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Flatten, Dense, MaxPool1D
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, f1_score, recall_score,
                                        precision_score
import numpy as np

# Load the data
file_path = '/content/Dyadic_dataset_iter2.xlsx'
df = pd.read_excel(file_path)
df_clean = df.drop(columns=['Unnamed: 0'])

# Encode the target variable and apply one-hot encoding
label_encoder = LabelEncoder()
targets = label_encoder.fit_transform(df_clean['Sync Direction'])
targets = to_categorical(targets)

# Prepare features
features = df_clean[['Child RMSSD', 'Parent RMSSD', 'Child IBI', 'Parent IBI',
                                        'Child Detrended IBI', 'Parent
                                        Detrended IBI', 'Child ARIMA Residuals'
                                        , 'Parent ARIMA Residuals', 'Max
                                        Correlations']].values

# Normalize the feature data
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features)

# Reshape data for 1D CNN
features_reshaped = features_normalized.reshape((features_normalized.shape[0],
                                        features_normalized.shape[1], 1))

# Custom callback to capture metrics during training
class MetricsHistory(Callback):
    def __init__(self, validation_data):
        super(MetricsHistory, self).__init__()
        self.validation_data = validation_data

    def on_train_begin(self, logs=None):
        self.epoch_metrics = {
            'loss': [],
            'accuracy': [],
            'val_loss': [],
            'val_accuracy': [],
            'val_f1': []
        }

    def on_epoch_end(self, epoch, logs=None):
        self.epoch_metrics['loss'].append(logs.get('loss'))
        self.epoch_metrics['accuracy'].append(logs.get('accuracy'))
```

```
        self.epoch_metrics['val_loss'].append(logs.get('val_loss'))
        self.epoch_metrics['val_accuracy'].append(logs.get('val_accuracy'))

        # Calculate F1 score using logs
        val_predict = np.argmax(self.model.predict(self.validation_data[0]),
                                                    axis=1)
        val_targ = np.argmax(self.validation_data[1], axis=1)
        _val_f1 = f1_score(val_targ, val_predict, average='macro')
        self.epoch_metrics['val_f1'].append(_val_f1)

def create_model4_v1(input_shape):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=
                                                    input_shape),
        Conv1D(filters=64, kernel_size=3, activation='relu'),
        MaxPool1D(pool_size=2, strides=2),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
                                            accuracy'])
    return model

# Function to run each model
def run_model_with_metrics(model_func, input_shape, fold_data):
    model = model_func(input_shape)
    metric_values = []
    all_epoch_metrics = []

    for train_index, test_index in fold_data:
        metrics_history = MetricsHistory(validation_data=(features_reshaped[
                                                    test_index], targets[test_index
                                                    ]))
        model.fit(features_reshaped[train_index], targets[train_index],
                                                    batch_size=40, epochs=100,
                                                    verbose=0,
                  validation_data=(features_reshaped[test_index], targets[
                                                            test_index]),
                  callbacks=[metrics_history])
        predictions = model.predict(features_reshaped[test_index])
        predictions = np.argmax(predictions, axis=1)
        true_classes = np.argmax(targets[test_index], axis=1)

        # Calculate metrics
        accuracy = accuracy_score(true_classes, predictions)
        f1 = f1_score(true_classes, predictions, average='macro')
        recall = recall_score(true_classes, predictions, average='macro')
        precision = precision_score(true_classes, predictions, average='macro')
        metric_values.append([accuracy, f1, recall, precision])
        all_epoch_metrics.append(metrics_history.epoch_metrics)

    metrics_df = pd.DataFrame(metric_values, columns=['Accuracy', 'F1 Score', '
                                            Recall', 'Precision']).median().
                                            tolist()
    return metrics_df, all_epoch_metrics

# Prepare for K-fold validation
num_folds = 10
```

```
kfold = KFold(n_splits=num_folds, shuffle=True)
fold_data = list(kfold.split(features_reshaped))

# Run the model and collect results and metrics history
results = {}
all_metrics_histories = {}
model_functions = [create_model4_v1]
for i, model_func in enumerate(model_functions, start=1):
    model_name = f"Model{i}"
    results[model_name], all_metrics_histories[model_name] =
                                            run_model_with_metrics(model_func,
                                            (features_reshaped.shape[1], 1),
                                            fold_data)

# Convert results to DataFrame and display
results_df = pd.DataFrame(results, index=['Median Accuracy', 'Median F1 Score',
                                          'Median Recall', 'Median Precision'])
print(results_df)

# Plot metrics
for model_name, metrics_histories in all_metrics_histories.items():
    # Initialize lists for aggregated metrics
    avg_metrics = {
        'loss': np.mean([history['loss'] for history in metrics_histories],
                                            axis=0),
        'accuracy': np.mean([history['accuracy'] for history in
                                            metrics_histories], axis=0),
        'val_loss': np.mean([history['val_loss'] for history in
                                            metrics_histories], axis=0),
        'val_accuracy': np.mean([history['val_accuracy'] for history in
                                            metrics_histories], axis=0),
        'val_f1': np.mean([history['val_f1'] for history in metrics_histories],
                                             axis=0),
    }

    plt.figure(figsize=(14, 6))

    # Plot Loss
    plt.subplot(1, 3, 1)
    plt.plot(avg_metrics['loss'], label='Train Loss')
    plt.plot(avg_metrics['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Plot Accuracy
    plt.subplot(1, 3, 2)
    plt.plot(avg_metrics['accuracy'], label='Train Accuracy')
    plt.plot(avg_metrics['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot F1 Score
    plt.subplot(1, 3, 3)
    plt.plot(avg_metrics['val_f1'], label='Validation F1 Score')
    plt.title(f'{model_name} F1 Score')
    plt.xlabel('Epoch')
```

```
    plt.ylabel('F1 Score')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

9 Appendix C

A weblink providing access to the dataset used to train the 1D CNN.
Click for data