"Decoding Brain Waves" A Comparative Study of CNN-RNN-based Classifier Architectures on EEG Data for Image Response Analysis

Alfredo Leon[†], Satria Bagus[‡]

***Abstract*—This study evaluates different Recurrent Neural Network (RNN) configurations within a Convolutional Neural Network-Recurrent Neural Network (CNN-RNN) framework for EEG signal analysis. Emphasizing data preprocessing, including smoothing and downsampling for augmentation, we aim to enhance model robustness against the complex temporal patterns in EEG data. Our experiments compare traditional RNN, GRU, and LSTM layers against Closed-Form Continuous-Time (CfC) networks combined with Neural Circuit Policies (NCP) to optimize model size and efficiency. We find that while a regular CfC model achieves best accuracy, CfC models equipped with NCP significantly reduce model size with only a moderate impact on performance.**

***Index Terms*—EEG Signal Classification, Convolutional Neural Networks, Recurrent Neural Networks, Neural Circuit Policies, Liquid Neural Networks**

I. Introduction

The quantification and scrutiny of cerebral data represent a multifaceted challenge, with the need of a comprehensive approach to encapsulate its principal dimensions: temporal dynamics and the measurement channels. The objective is to discern the electromagnetic responses elicited by distinct cerebral regions in reaction to specific stimuli. Given the substantial dimensionality and the extensive computational demands typically associated with the processing of such data, there is a pressing need for the development of rapid, small and precise algorithms to facilitate the analysis.

Recent endeavors, such as those discussed in [1], underscore the significance of neural network methodologies in the development of EEG classifiers. The adeptness of deep learning paradigms in discovering and highlighting interrelations within data underscores the pertinence of adopting these techniques. Notably, the employment of recurrent neural networks (RNNs) is deemed intuitive due to their proficiency in processing sequential datasets. Moreover, this study embarks on a comparative analysis of various RNN layers, including vanilla RNNs, Gated Recurrent Units (GRUs), Long Short-Term Memory (LSTM) networks, and Closed-Form Continous-Time networks (CfC) [2], in conjunction with the Neuronal Circuit Policies (NCP) wiring as proposed by [3]. Our objective is to rigorously assess the efficacy of various Recurrent Neural Network (RNN) layer configurations within our framework, with the intent of delineating scenarios where each specific model could be optimally employed.

In this study, we perform a benchmarking exercise to evaluate the efficacy and architectural dimensions of diverse iterations of a Convolutional Neural Network-Recurrent Neural Network (CNN-RNN) framework. This entails experimenting with assorted recurrent layer configurations and their respective parameters. The empirical foundation of this study is derived from the dataset documented in [4]. Specifically, our examination is concentrated on the Hand Leg Tongue (HaLT) paradigm, wherein participants, outfitted with 22 electrodes, were intermittently exposed to images depicting the left and right hands, feet, or tongue. The intricacies of our data preprocessing strategies will be discussed in subsequent sections.

Following a brief review of pertinent literature (II), we will furnish a comprehensive overview of our proposed architectural solution, thereafter delving into the details of our approach (III) and the description of the signals analysed in building our classifiers (IV). Next, we will describe in detail our learning framework (V). Subsequently, we will dissect and interpret the outcomes yielded by the models (VI). To finalize, we will culminate with a series of concluding remarks and reflections (VII).

II. Related Work

The work in [5] already highlights the range of applications within EEG data analysis and processing. In the context of classification challenges, traditional algorithms such as Support Vector Machines (SVM), Naïve Bayes (NB), decision trees (DT), and Logistic Regression (LR) have been explored. The advent of deep learning introduced Convolutional Neural Networks (CNNs) along with Recurrent Neural Networks (RNNs) and their advanced variant, Long Short-Term Memory (LSTM) networks, as common tools to the EEG analysis domain.

Furthermore, [1] delves into the utilization of neural networks for classifiers tailored to EEG data, underscoring the superior efficacy of CNNs and RNNs when compared to

[†]University of Padova, email: luisalfredo.leonvillapun@studenti.unipd.it
[‡]University of Padova, email: satriabagus.wicaksono@studenti.unipd.it
The code implementation for this study can be found at: https://github.com/Action52/HumanDataProject

traditional architectures like stacked autoencoders and multilayer perceptrons. The combination of these into CNN-RNN architectures provides a powerful framework for prediction of time-series data [6].

The study in [7] uses 1D CNNs to process EEG signals, applying convolutions to both raw and augmented data. A similar architecture is explored in [8], analizing the effect of multiple convolutional layers in building time-series classifiers. While this approach enhances feature extraction, it does not fully exploit the data's temporal dynamics, an area where 1D CNNs excel by maintaining temporal coherence [9].

To fully leverage the temporal attributes of the dataset, the employment of layers designed for sequential data, such as Recurrent Neural Networks (RNNs), becomes imperative. [9] examines the functionality of RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) cells, noting that in a basic RNN structure, each cell incorporates the context of its predecessor to inform its output. However, this model is susceptible to the "vanishing/exploding gradient", which compromises the network's ability to maintain long-term temporal dependencies. GRU and LSTM architectures are devised as enhancements to RNNs, aiming to mitigate gradient-related challenges.

An innovative variant within the RNN paradigm is the Liquid Time-Constant (LTC) network, introduced by [10]. LTC networks conceptualize the input signal as a continuous temporal function, diverging from the discrete treatment observed in conventional approaches, and address this through neural differential equations. Comparative analyses involving LTC, LSTM, GRU, and traditional RNN architectures on elementary time-series datasets demonstrated equal or superior performance for LTC. Subsequent explorations into LTC's capabilities have included the implementation of a topology algorithm inspired by the neural circuitry of the *C. elegans* nematode, as delineated in [3]. This innovative topology has been instrumental in diminishing the required number of cells and recurrent connections for precise time-series prediction. Moreover, a LTC improvement was designed in [2]. This new type of cell approximates a closed-form solution to the LTC, this is why this type of layer was named Closed-form Contious Time Network (CfC).

III. Processing Pipeline

Our methodology unfolds through a four-phase process, as depicted in 1. Initially, the EEG signals undergo a comprehensive preprocessing routine, starting with loading, segmentation, and normalization based on their respective labels. Subsequently, to enrich the dataset and enhance the model's robustness, we implement data augmentation techniques. This includes generating smoothed variations of the time series via moving averages to capture the underlying trends across varying temporal windows, alongside downsampling to foster the model's invariance to signal duration.

The ensuing phase involves the application of a localized one-dimensional convolutional layer, designed to extract salient features from each channel independently across all augmented signal iterations. This layer serves as the cornerstone for initial feature identification, setting the stage for more complex pattern recognition.

Advancing to the third stage, our architecture integrates a recurrent neural network layer, where we explore the efficacy of various RNN configurations, including the conventional Vanilla RNN, the more advanced GRU and LSTM, and the innovative CFC, the latter employing the Neural Circuit Policy algorithm as delineated in [3]. This phase is instrumental in assessing the comparative performance of distinct recurrent architectures and advocating for a bespoke recurrent layer wiring strategy aimed at minimizing the model's parameter footprint, thereby significantly enhancing computational efficiency.

In the fourth stage, the processed data is funneled through a dense layer, culminating in the final classification where the probabilities for each label are computed and presented.

Our architecture draws significant inspiration from the seminal work of [7] and [8], with a pivotal modification being the introduction of a recurrent layer subsequent to the one-dimensional convolution. This adjustment stems from our observation that global convolutions may not fully exploit the temporal correlations within the data. Given that 1D convolutions inherently preserve temporal relationships as noted by [9], it is logical to further process the extracted features through an RNN layer to capitalize on these temporal dynamics.

IV. Signal Pre-processing

Before inputting data into our neural networks, we pre-processed the raw EEG signals to refine the data, focusing on four main steps: *normalization*, *smoothing*, *segmenting*, and *down-sampling*. This section will also detail our approach to dividing the dataset into training, validation, and testing sets for the training and testing the models.

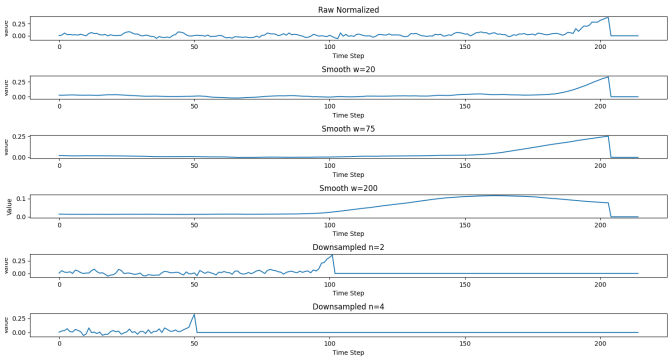

Fig. 2: Data augmentation steps for one channel, for one observation.

Our process follows the guidelines provided in reference [6], with a special focus on *smoothing* and *down-sampling* to improve signal quality. We begin by normalizing the signals across all channels. Then, we segment the data based on the labels information, preparing each segment through *smoothing*

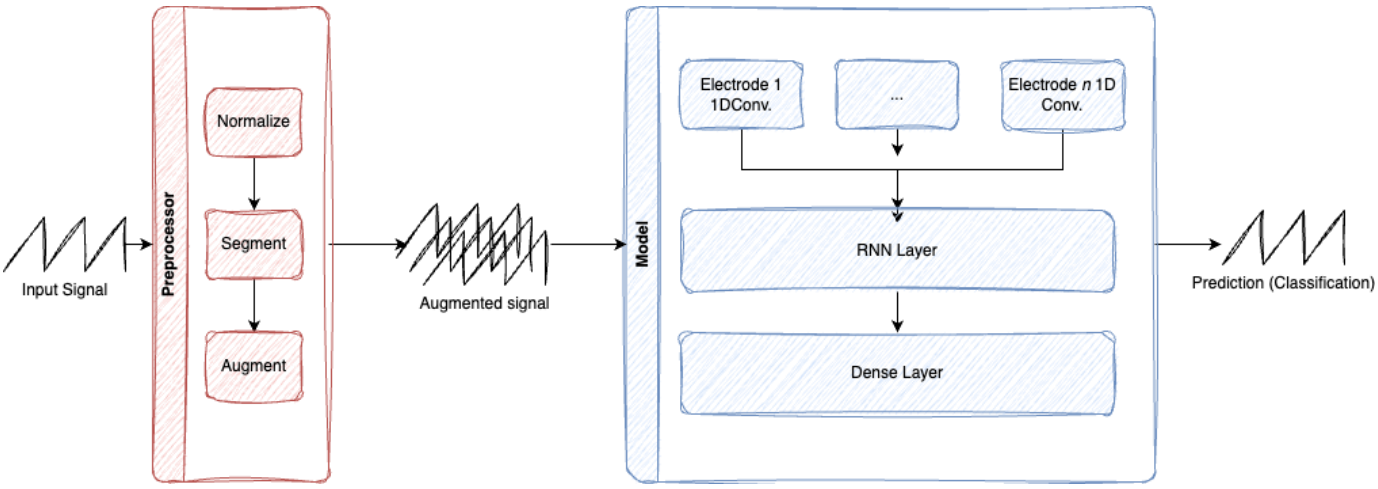

Fig. 1: General overview of our architecture.

and *down-sampling* for our neural network input. Figure 2 shows an example augmentation to enrich the signal data. Finally, we split the dataset into training, validation, and testing datasets.

A. Raw Signals

The raw signals, as defined by [4], are obtained with a 200 Hz sampling rate. Consequently, we chose not to remove artifacts from our EEG signals, following the approach mentioned in [4]. It is assumed that the EEG BCI-Data processing subsystem is sufficiently equipped to handle any noise introduced during recording. This understanding allows us to maintain a streamlined preprocessing workflow, ensuring the data is reliable for analysis. The action signal appears in the raw data for 1 second and involves 21 input channels. The 22$^{\text{nd}}$ channel does not contain EEG data and is only used for verification; thus, we remove it.

B. Normalization

Normalization plays a crucial role in EEG data processing, addressing the non-stationary nature and variability across sessions and subjects. Z-score normalization, as detailed by [11], is particularly effective for managing these variations. We apply this technique to each channel, using the formula:

$$z_{i,c} = \frac{x_{i,c} - \mu_c}{\sigma_c}$$

where $z_{i,c}$ is the normalized data point for the i^{th} observation in channel c, μ_c denotes the mean, and σ_c the standard deviation of the observations in channel c. This approach ensures data is uniformly processed, providing a consistent baseline across different sessions and subjects for subsequent segmentation and analysis.

C. Multi-frequency Smoothing

For the multi-frequency smoothing phase, we focused on smoothing the signals within each channel to mitigate signal variance. This step was inspired by a method utilizing a moving average across multiple window sizes [8]. Given a EEG signal channel represented as a sequence $x[n]$, where $n = 0, 1, \ldots, N-1$, and a moving average filter of window size W, the smoothed signal $y[n]$ can be obtained as follows:

$$y[n] = \frac{1}{W} \sum_{k=0}^{W-1} x[n-k]$$

To ensure that the smoothing process is uniformly applied across the entire signal, we employ edge padding. This technique extends $x[n]$ by replicating its edge values at both the beginning and the end.

We calculated the moving average for each segment using three predetermined window sizes: 20, 75, and 200. The choice of window sizes was informed by initial tests conducted with sample datasets, which allowed us to refine our method for achieving optimal smoothing outcomes within our experimental constraints.

D. Segmenting

After normalizing and applying multi-frequency smoothing to the dataset, we proceed to the segmentation phase, utilizing label information as outlined in [4]. Suppose we have the time series data:

$$T = \{t_1, t_2, \ldots, t_n\}$$

the segment S_i is defined as

$$S_i = \{s_1, s_2, \ldots, s_n\}$$

where n is the size of the segment, and each segment contains data with the same label. This process entails segmenting the data according to label information, specifically focusing on labels 1 to 6. Labels such as 0, 91, 92, and 99 are excluded due to their irrelevance to the experimental setup.

The segment size is set at 215, reflecting the typical duration of 1 second in experiments conducted at a 200 Hz sampling rate. For segments shorter than 215 samples, padding with zeros is applied to reach the requisite length. Conversely,

segments exceeding 215 samples in length are cropped to maintain uniformity. This ensures that each segment is of equal length, facilitating a consistent analysis framework.

E. Multi-scale Down-sampling

We also down-sampled the data to capture temporal patterns from different time scales [8]. The process of down-sampling is described as follows. Suppose that we have a segment:

$$S = \{s_1, s_2, \ldots, s_n\}$$

Given a down-sampling degree k, the down-sampled segment K is obtained by retaining only every k-th element from S, which can be mathematically represented as:

$$K = \{s_1, s_{1+k}, s_{1+2k}, \ldots\}$$

The sequence is continued by including every k-th element until reaching the end of the segment S. After down-sampling, we pad K with zeros to ensure that each down-sampled segment retains the same dimensionality as the original. We used degrees of 2 and 4, considering our segment size of 215, to preserve sufficient data for analysis. Post down-sampling, zero-padding ensures each segment maintains original dimensionality, simplifying analysis.

F. Splitting the Dataset

After pre-processing, we split our dataset into training, validation, and testing sets with a 63-27-10 split, following the approach used in the baseline method [4]. This strategy balances extensive training with adequate validation and testing, allowing for effective model tuning and performance evaluation. By adopting this proven split ratio, we facilitate meaningful comparisons with established benchmarks and ensure our models are well-validated against diverse data scenarios. We maintain the consistency of our dataset by using the same *seeds* for all experiments to ensure the integrity of the datasets.

V. Learning Framework

Once we have preprocessed the data, we have a tensor with a shape of (b, v, t, c) where b represents the batch size, v is the number of versions we augmented the data with, t time steps, and c channels.

A. Local Convolution Layer

In our architecture, the *WindowsConvolutionLayer* processes the EEG signals at a granular level, corresponding to individual diodes or channels. This custom layer, is designed to manage inputs from multiple diodes, with each diode undergoing a series of one-dimensional convolutions independently. The layer is initialized with a configuration specifying the number of diodes and the convolutional parameters for each diode, facilitating tailored feature extraction from each channel. The 1D convolution step formula is given by

$$(f * g)(i) = \sum_{j=1}^{m} = g_j \cdot f(i - j + m/2), \text{[1]} \quad (1)$$

where we multiply the signal f against a kernel g in the hopes of highlighting relevant features in the signal.

During the forward pass, the layer slices the input tensor to isolate data corresponding to each diode. These slices are then sequentially processed through their respective convolutional blocks. Post-convolution, the output for each diode is reshaped to introduce an additional dimension, ensuring that the feature maps remain distinguishable across diodes.

The culmination of this process involves aggregating the convolved outputs from all diodes, stacked to preserve the integrity of the extracted features. This approach enables the layer to maintain a high degree of specificity in feature extraction, enhancing the model's ability to discern nuanced patterns within the EEG data, crucial for accurate classification tasks. Therefore, the resulting tensor before running the layer has a shape of (b, v, t, c, f) where f are the feature maps learned.

B. Recurrent Neural Network Layer

This layer serves as the integration point for various Recurrent Neural Network (RNN) architectures, enabling a comprehensive evaluation of their respective impacts on our classification task. To accommodate the specific requirements of these RNN cells, we initially reshape the input tensor to a compatible format, denoted as $(b, t, v * c * f)$, where b represents the batch size, t the time steps, and $v * c * f$ the vectorized feature dimensions. In the subsequent subsections, we delineate the distinct RNN layers deployed, with each model incorporating a unique RNN variant.

After going through the recurrent layer, the tensor has a shape of (b, t, r), where r represents the number of units in the recurrent layer. In the case of the model using CfC with NCP, the shape is given by (b, t, r_o), where r_o are the number of *motor* neurons in the neural circuit.

1) Vanilla RNN: A fully connected simple RNN layer, we recall the cell state of a RNN layer is given by

$$c^{(t)} = tanh(W_{cc}c^{(t-1)} + W_{cx}x^{(t)} + b), \quad (2)$$

where we only multiply weight matrices $W_{cc} and W_{cx}$ against the cell state at the previous time step $c^{(t-1)}$ and against the current input $x^{(t)}$. This approach can entail two problems. On one hand, the full connection of the layer ensures a high computational demand, while on the other we can experience the vanishing gradient problem.

We implement a model with a Vanilla RNN layer, in order to measure the effects of normal recurrency compared to other more complex types of RNNs.

2) LSTM: LSTMs address vanishing and exploding gradients in RNNs through three gates: *forget gate*, *input gate*, and

[1]Taken from http://tinyurl.com/bde2534a

output gate. The forget gate regulates information retention as per the equation [9]:

$$f(t) = \sigma(W_{fh}h(t-1) + W_{fx}x(t) + b_f), \tag{3}$$

where W_{fh} and W_{fx} are weight matrices, b_f is the bias, $h(t-1)$ represents the previous state, and $x(t)$ the current input. The sigmoid function σ scales the output between 0 and 1, denoting the necessity of information.

The input gate's role in updating the cell state is highlighted by [9]:

$$c(t) = (f(t) \cdot c(t-1)) + (i(t) \cdot \hat{c}(t)), \tag{4}$$

with $c(t)$ as the updated cell state, $i(t)$ indicating the current input's significance, and $\hat{c}(t)$ as the candidate state.

The impact of the output gate on the hidden state is detailed by [9]:

$$y(t) = o(t) \cdot \tanh(c(t)). \tag{5}$$

LSTMs enhance the modeling of long-term dependencies, potentially improving EEG classifier accuracy by introducing complex mechanisms compared to standard RNNs [9].

3) GRU: GRUs address the vanishing and exploding gradient challenges of RNNs with a simpler design, featuring two gates: *reset gate* and *update gate*, and excluding the cell state present in LSTMs. The reset gate's function in determining the amount of past information to discard is specified by [9]:

$$r^{(t)} = \sigma(W_{rh}h^{(t-1)} + W_{rx}x^{(t)} + b_r), \tag{6}$$

where $r^{(t)}$ is computed via the sigmoid function σ. The update gate's role in deciding the relevance of information to retain is expressed by [9]:

$$z^{(t)} = \sigma(W_{zh}h^{(t-1)} + W_{zx}x^{(t)} + b_z), \tag{7}$$

with $z^{(t)}$ managing the information transition from the previous state $h^{(t-1)}$ to the current state $h^{(t)}$.

Our model incorporates a GRU layer to facilitate a direct comparison with LSTMs, aiming to assess GRUs' advantages and performance in the same experimental context [9].

4) CfC: [10] introduced a new framework for solving time-dependent data. A Liquid Time Constant (LTC) cell is a type of Continuous-Time Recurrent Neural Network (CT-RNN) that is solved through the use of differential equations. In [2], we are presented with Closed-Form Continuous-Time Constant cells, a closed-form approximation of the LTC. Let's assume the synapse between two cells over time as the interaction between a pre-synaptic stimuli $I(t)$ and the difference between synaptic reversal potential A and the post-synaptic neuron potential $x(t)$

$$S(t) = f(I(t))(A - x(t)). \tag{8}$$

LTCs solve numerically

$$\frac{dx(t)}{dt} = -\frac{x(t)}{\tau} + S(t), \tag{9}$$

which, in terms of training, proves to be a slow process. CfCs approximate the closed-form solution as

$$x(t) = (x(0) - A)e^{[-\frac{1}{\tau} + f(I(t))]t} f(-I(t)) + A. \tag{10}$$

In this context, τ represents the time-constant of $x(t)$, imbuing the architecture with the capacity for precise temporal approximations of time-dependent data. By leveraging such cells, characterized by their reduced parameter set, our aim is to obtain performance on par with more conventional RNN architectures, at a reduced computational cost.

C. Implementing NCP Wiring

The work of [3] introduces the Neural Circuit Policy (NCP) wiring algorithm, drawing inspiration from the anatomy of the *C. Elegans* nematode. This algorithm adopts a biologically inspired architecture, structured into four neuron layers: the *sensory* layer for input reception, two hidden layers (*inter* and *command*) which embody the network's recurrent components, and the *motor* layer responsible for output generation. The process begins with the sensory layer receiving stimuli, which are then relayed through the intricate network of the inter and command layers, culminating in the motor layer's output production. A conceptual overview of this wiring algorithm is provided in Fig. 3, with a more detailed exposition available in [3].

The incorporation of NCP wiring into our models facilitates the construction of highly efficient and compact topologies without compromising accuracy. Our experimental framework integrates a Closed-Form Continuous-Time (CfC) layer, configured with a sparsely connected NCP, to validate the potential of such architectures. This approach underscores the viability of deploying streamlined models in resource-limited settings, emphasizing the balance between model simplicity and performance efficiency.

1) *Layer insertion*: Insert four layers N_s, N_i. N_c, N_o.
2) Synapse insertion: Between each consecutive layer, insert k synapses with a predefined sparsity. In our experiments, we used a sparsity of 75% to show that even highly sparse Liquid RNNs can achieve competitive results. The number of synapses per cell is given by the distribution of the target layer.
3) Ensure connectivity: For every target neuron that did not receive a synapse, insert m synapses randomly.
4) Recurrent connections: For every command neuron, insert l synapses randomly to other command neurons.

Fig. 3: The NCP creation algorithm, from [3].

D. Dense Layer

Following the processing by preceding layers, the data is subsequently directed through a final dense layer. Here, we receive a tensor of shape (b, t, r) that we flatten and run through a dropout layer to help the model better generalize, and then through the dense layer. This culminates in the generation of a tensor of shape (b, n), where n are the output

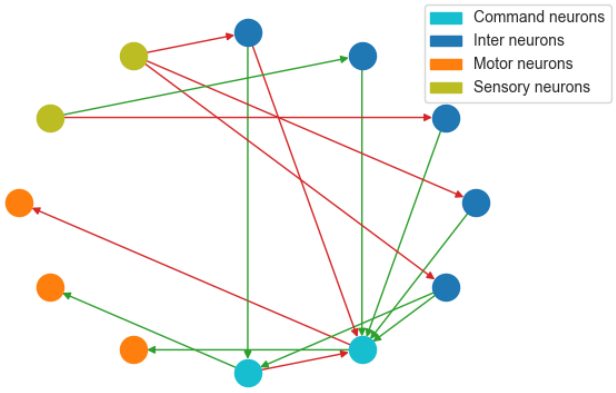

Fig. 4: Example NCP Wiring as used in our experiments.

neurons, each corresponding to the likelihood of a given label being the true classification. The softmax activation function is employed to normalize these outputs into a probabilistic distribution, facilitating the direct interpretation of the model's predictions.

VI. Results

A. Setup

The experimental setup was conducted on a Google Colab notebook, leveraging the platform's native TensorFlow environment. This environment was powered by an NVIDIA A100 GPU, supplemented by 80GB of system RAM and 40GB of GPU RAM, and a 166GB hard disk capacity. A total of six distinct experiments were orchestrated to evaluate the performance of various model architectures. To ensure fairness in our comparative analysis, each model was equipped with a standardized *Local Convolution Layer* and a *Dense Layer*. The specific hyperparameters employed for these components are delineated in Table 1.

TABLE 1: Shared Model Layers Parameters

Layer	**Parameters**
Conv1D (1)	filters=4, kernel_size=10, activation='relu'
Conv1D (2)	filters=4, kernel_size=10, activation='relu'
Conv1D (3)	filters=4, kernel_size=10, activation='relu'
Flatten	-
Dropout	0.5
Dense (1)	units=32, activation='relu'
Dense (2)	units=6, activation='softmax'

In each experiment, a distinct Recurrent Neural Network (RNN) architecture was integrated within the *Recurrent Layer*. The configurations for these architectures are detailed in Table 2. It is noteworthy that for the implementation involving Neuronal Circuit Policies (NCP), a high sparsity parameter was deliberately chosen. This approach was aimed at refining the model to be more lightweight while reducing, or potentially maintaining, its performance efficacy.

B. Training

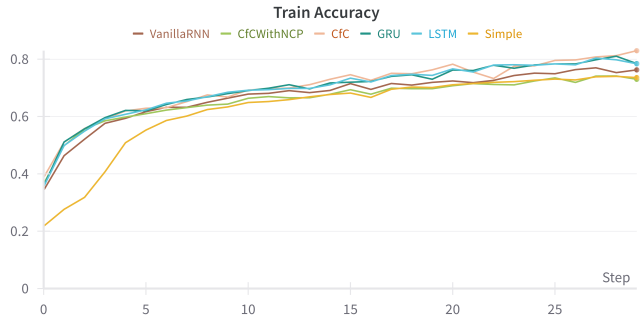

Fig. 5

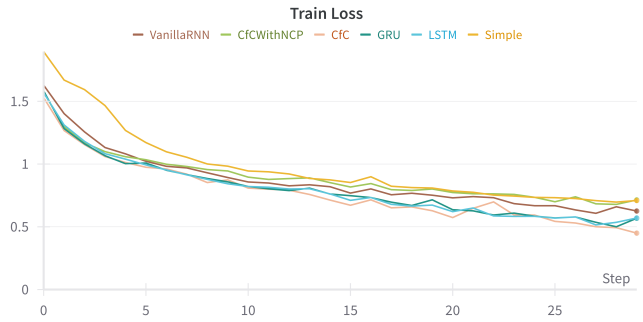

Fig. 6

In the training phase of our models, we employ the *Adam* optimizer, chosen for its efficiency in handling sparse gradients and adapting learning rates. For the loss function, we utilize *sparse categorical crossentropy*, which is particularly effective for multi-class classification problems where each class is mutually exclusive. The training process is conducted over 30 epochs in batches of 600 observations, allowing for sufficient iterations to converge to an optimal solution.

Figures 5 and 6 show the training process of the experiments. We can observe how all models converge to an optimum in the predefined 30 epochs.

C. Experimentation outcomes

The experimental results, with a focus on memory utilization, are presented in Table 3, while the performance metrics for each model are outlined in Table 4. An intriguing observation from Figure 7 is the relatively high memory footprint of the Simple Model, which lacks any RNN layer. This phenomenon is attributed to the direct connection of neurons from the convolutional phase to the dense layer, resulting in a significantly larger parameter count compared to configurations where data is first processed through a Recurrent Layer. Another disadvantage of this approach is, that we are not accounting for the temporal relationships within the model, thereby translating into less accuracy.

TABLE 2: RNN Layer Parameters

Model	**Parameters**
Simple Model	No RNN included
Vanilla RNN	units=128, activation='relu'
GRU	units=128
LSTM	units=128, activation='tanh'
CfC	units=128
CfC with NCP	units=128, AutoNCP(output size=48, sparsity level=0.75),

TABLE 3: Performance Metrics of Various Models

Model	**Training Time (sec)**	**Inference Time (sec)**	**Weight Size (MB)**	**Parameter Count**
Simple	**253.183**	0.002	12.292	3.04M
Vanilla RNN	342.798	0.002	3.572	0.85M
LSTM	263.433	0.002	4.544	1.10M
GRU	262.935	0.002	4.222	1.02M
CfC	449.470	0.003	3.841	0.92M
CfC w/ NCP	748.656	0.005	**1.871**	**0.42M**

TABLE 4: Evaluation Metrics of Various Models (Test)

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Simple	0.773	0.790	0.773	0.776
Vanilla RNN	0.819	0.819	0.819	0.819
LSTM	0.842	0.842	0.842	0.842
GRU	0.832	0.838	0.832	0.833
CfC	**0.885**	**0.886**	**0.885**	**0.885**
CfC w/ NCP	0.785	0.795	0.785	0.786

Among the tested models, the Vanilla RNN, LSTM and GRU variants achieved similar results. Despite being similar in size, the LSTM and GRU achieve slightly better accuracies, but still far from the best performant model.

Finally, the CfC model exhibited the best performance, 88.5%. In its standard configuration, it achieved 4% and 5% better results compared to LSTM and GRU, with a slightly reduced memory footprint.

Incorporating NCP wiring introduced a balance between model compactness and accuracy, with the model attaining an 78.5% accuracy rate, still outperforming the Simple model, while being nearly half the size of its closest competitor (CfC) in terms of size. This outcome underscores the efficacy of NCP wiring in developing models that are both competitive and resource-efficient. The deployment of a 0.75 sparsity level accentuates the strategic significance of architectural planning, surpassing mere layer and cell adjustments, in crafting cost-effective solutions. Figure 9 contains the confusion matrix for this cost-effective model. Figure 8 displays the confusion matrix for CfC, the most accurate model.

Conversely, the training durations for both CfC variants exceeded expectations. It is imperative to acknowledge that the implementation detailed in [3] are at an experimental stage, lacking specialized optimizations for TensorFlow that are already available for LSTM and GRU architectures. This aspect is crucial in understanding the extended training times observed.

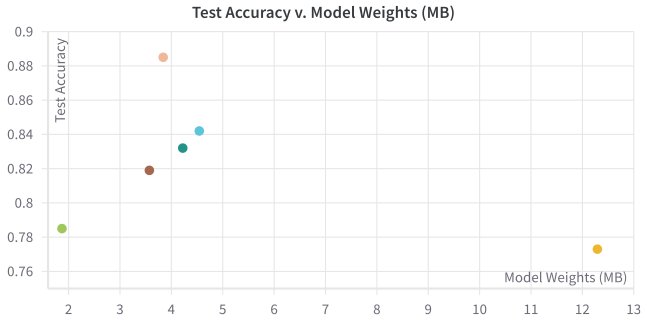

Fig. 7: Using NCP (light-green) can heavily reduce model size by sacrificing accuracy. A normal CfC layer (pink) outperforms models incorporating LSTM (blue) and GRU (green).

VII. Concluding Remarks

In alignment with the pioneering studies of [7] and [8], our research introduces a refined architectural model that features a versatile Recurrent Neural Network (RNN) layer. This layer is designed to be both adaptable and customizable, enabling tailored parameterization to meet diverse analytical requirements. Central to our methodology is an advanced

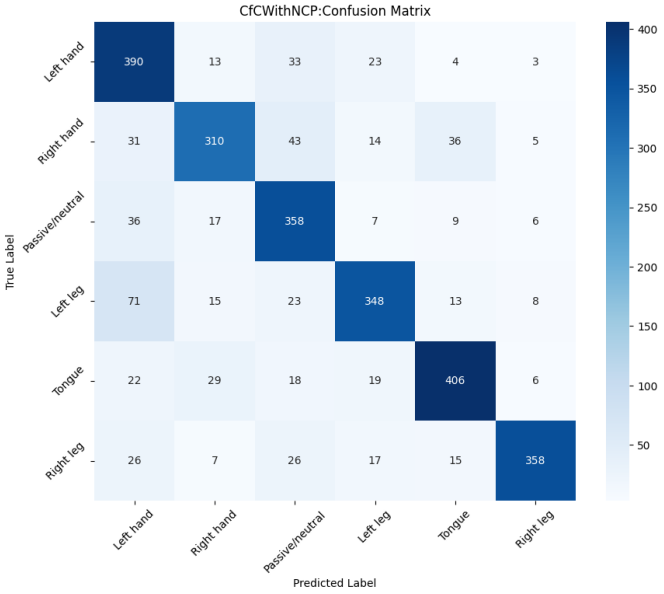

Fig. 8: We can observe the trade-off in accuracy by using a NCP model.

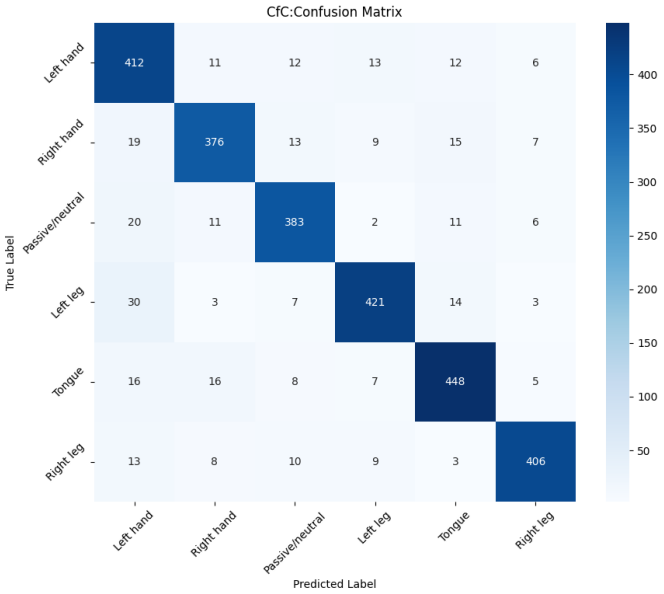

Fig. 9: Regular CfC was the best performing model in terms of accuracy.

data pre-processing strategy, which we define as the cornerstone of our architectural efficacy. This approach not only ensures the provision of meticulously normalized data but also incorporates strategic data augmentation techniques such as smoothing and downsampling. These techniques are not merely preparatory steps; they are integral in bolstering model robustness by exposing the network to both macroscopic trends and granular temporal hierarchies, thereby enhancing its generalization capabilities.

Moreover, our framework adopts an algorithmic approach through the integration of the Neuronal Circuit Policies (NCP) algorithm, as detailed in [3]. This inclusion is not just an augmentation but a transformative strategy that enables the construction of lightweight yet highly predictive models. By leveraging NCP, our models achieve an optimal balance between performance and efficiency, maintaining high predictive accuracy while significantly reducing their computational footprint.

REFERENCES

[1] Y. H. Alexander Craik and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (eeg) classification tasks: a review," *Journal of Neural Engineering*, vol. 16, Apr. 2019.

[2] R. Hasani, M. Lechner, A. Amini, L. Liebenwein, A. Ray, M. Tschaikowski, G. Teschl, and D. Rus, "Closed-form continuous-time neural networks," *Nature Machine Intelligence*, vol. 4, pp. 992–1003, Nov 2022.

[3] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, pp. 642–652, Oct 2020.

[4] M. Kaya, M. K. Binli, E. Ozbay, H. Yanar, and Y. Mishchenko, "A large electroencephalographic motor imagery dataset for electroencephalographic brain computer interfaces," *Scientific Data*, vol. 5, no. 1, p. 180211, 2018.

[5] A. Khosla, P. Khandnor, and T. Chand, "A comparative analysis of signal processing and classification methods for different applications based on eeg signals," *Biocybernetics and Biomedical Engineering*, vol. 40, no. 2, pp. 649–690, 2020.

[6] X. Jin, X. Yu, X. Wang, Y. Bai, T. Su, and J. Kong, "Prediction for time series with cnn and lstm," in *Proceedings of the 11th International Conference on Modelling, Identification and Control (ICMIC2019)* (R. Wang, Z. Chen, W. Zhang, and Q. Zhu, eds.), (Singapore), pp. 631–641, Springer Singapore, 2020.

[7] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "Eegnet: a compact convolutional neural network for eeg-based brainâ€“computer interfaces," *Journal of Neural Engineering*, vol. 15, p. 056013, jul 2018.

[8] W. Chen and K. Shi, "Multi-scale attention convolutional neural network for time series classification," *Neural Networks*, vol. 136, pp. 126–140, 2021.

[9] D. Walther, J. Viehweg, J. Haueisen, and P. MÃ¤der, "A systematic comparison of deep learning methods for eeg time series analysis," *Frontiers in Neuroinformatics*, vol. 17, 2023.

[10] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "Liquid time-constant networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7657–7666, May 2021.

[11] A. Apicella, F. Isgrò, A. Pollastro, and R. Prevete, "On the effects of data normalisation for domain adaptation on eeg data," *Engineering Applications of Artificial Intelligence*, 2023.