EEG 2 Class ML PROJECT

EEG Dataset of choice

Using a published dataset has the advantages of ease of comparison with other published research and benchmarks, but it limits the complexity of our Motor Imagery protocol. Ideally we would start with a 4 class motor imagery, that would provide us 4 bits of information to control the prosthetic device to be added to the proportional control from the EMG.

The quantity of online datasets that used strict protocols and a standard data reporting system for MI-EEG are limited, the best compromise we could find was a dataset with 2 classes of Motor Imagery from the dominant arm, that uses the EEG-BIDS data format convention [1], with raw EMG and EEG data, good labeling, processed data and a benchmarked accuracy classification algorithm that we will compare with our framework.

It uses the same 10-20 EEG electrode layout, but with 62 channels of EEG data, as shown in Fig. 27.

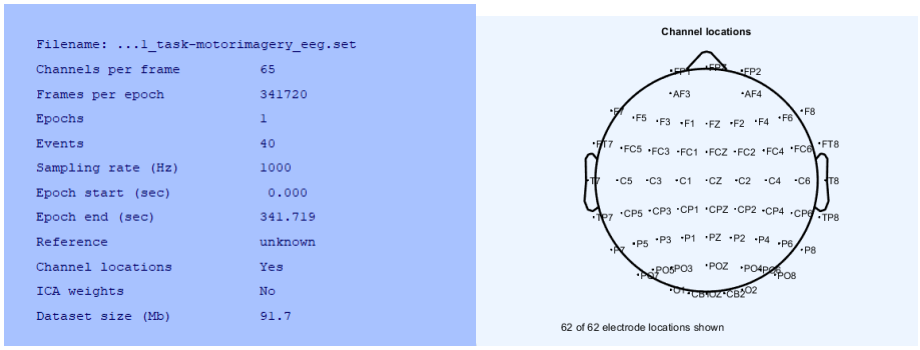


Fig. 1. *EEGLAB's [2] interface with the dataset parameters for one trial of one subject and the view of the used electrodes positioning on the scalp, using the 10-20 system. Channels HEO, M2, VEO, EMG1 and EMG2, were separated from the EEG data for this analysis, resulting in 62 EEG data channels.*

The MI training protocol used is similar to the one described on Section 3.1.4,, but with several subjects across several trials. For specific details of the EEG data acquisition and MI training protocol used on this dataset and please refer to [3].

EEG Data Processing

We used the EEGLAB's MATLAB extension [2] to preprocess the data and improve data visualization. A script to load the dataset and pipeline the data preprocessing was developed.

First, we loaded the EEG data from the first session of the first subject. Then each column of the data is labeled accordingly, to assign accurate spatial coordinates to each EEG channel.

Next, the script separates the EMG and Electrooculography (EOG) channels, used to measure muscle and eye movements that might be used on other part of the process. The non-EEG channels are removed, to focus on the brain activity.

For task-based sessions, we identify and retain data segments corresponding to task performance, eliminating any data recorded before or after the task. Then the data is re-referenced using the common average reference (CAR) method that reduces the noise that is common for all the channels. CAR was calculated using Eq. 3.10, where N is the number of EEG channels.

$$\tilde{s}_i = s_i - \frac{1}{N}\sum_{i=1}^{N} s_i \qquad (3.1)$$

A band-pass filter is used in order to restrict the EEG signals to a certain frequency range. High pass filter edge at 7 Hz to and low pass filter edge at 35 Hz, capturing the most relevant brainwave activity for the MI signal detection [4], [5]. Baseline correction is then performed to remove any steady-state offsets that could distort the signal. To reduce ocular artifacts, the script detects and removes EOG interference using Automatic Artifact Removal (AAR) method from EEGLAB, enhancing signal clarity. After this

correction, the data is re-referenced again using CAR method to ensure accuracy. The data before and after pre-processing is shown on Fig. 35.

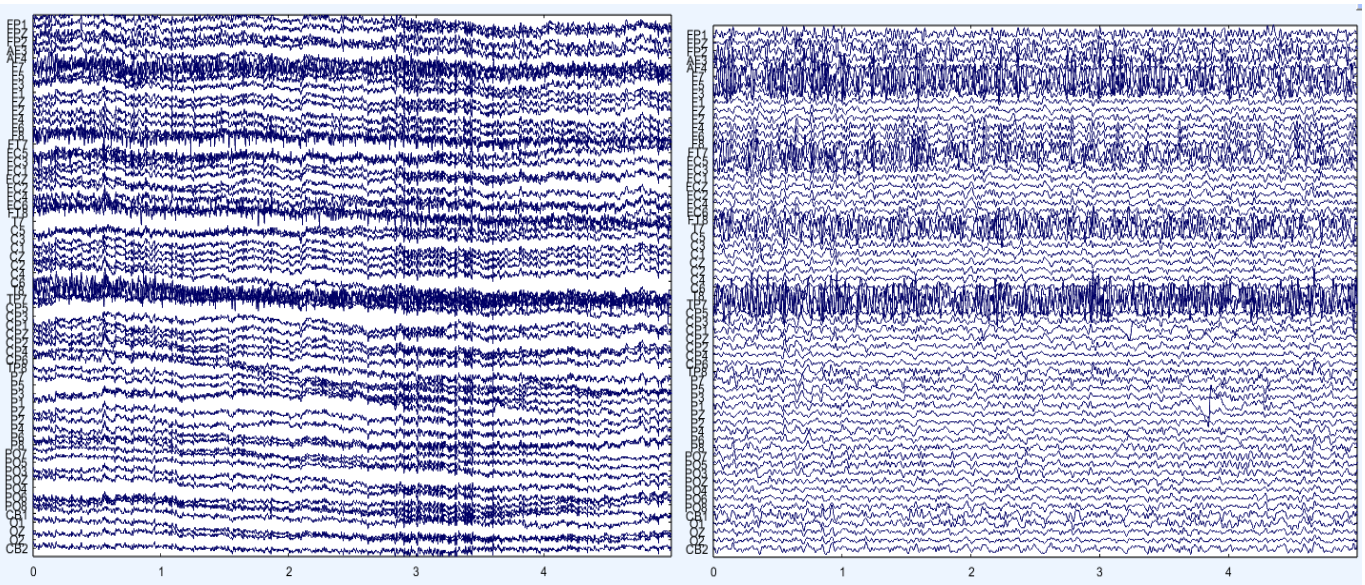


Fig. 1. *EEG data before (left) and after preprocessing (right): 62 Channels listed, their relative amplitude variation is displayed, the abscissa is the time in seconds.*

Finally, the script performs a consistency check to confirm that all channels and segments are processed correctly, channels with non-removable artifacts close to the MI intervention time range are removed from the data and an interpolation with the neighboring channels is carried out to substitute the missing electrode. After that the script iterates over all the sessions saving the preprocessed EEG data to be later formatted during the data loading process for the NNs.

The preprocessed EEG-MI log power spectra are plotted on Fig. 36 with the frequency domain shown for all electrodes. Two hotspots can be noticed on the scalp plot, close to the primary sensorimotor cortex, with a high intensity of 12Hz. According to the literature the μ and β rhythms (i.e., 8-13Hz and 13-30Hz respectively) are the ones that present variation of amplitude during motor imagery [4].

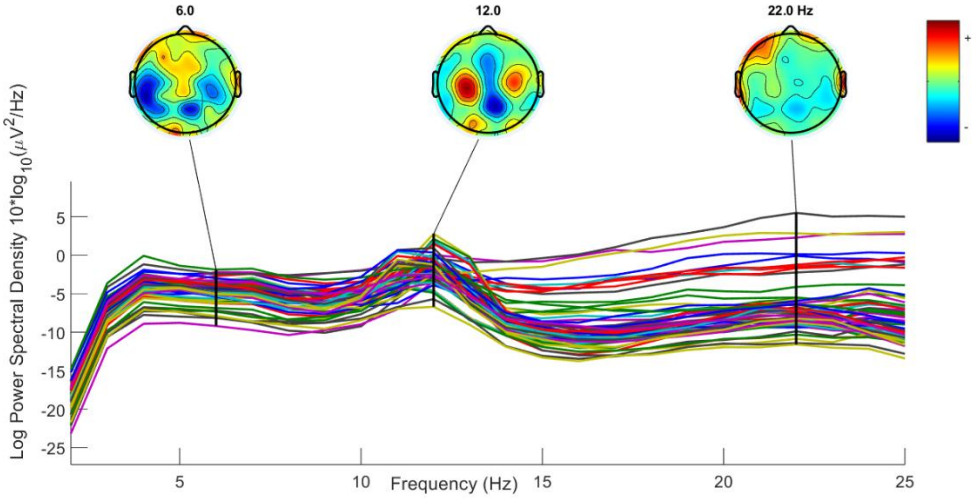


Fig. 2. *Log power spectral density for a given subject of all sessions of MI for each one of the 62 EEG channels. The scalp plot shows which areas of the head were more active for a given frequency.*

We also computed some time domain features to evaluate if using them as the input data would meaningfully change the classification accuracy. Hjorth in 1970 developed three-time domain statistical parameters [6], that are still widely used [4].

$$A(x(n)) = \frac{\Sigma^{N}_{n=1}\,[x(n) - \mu]^2}{t} \tag{3.2}$$

$$M(x(n)) = \sqrt{\frac{\sigma(x\prime(n))}{\sigma(x(n))}} \tag{3.3}$$

$$C(x(n)) = \frac{M(x\prime(n))}{M(x(n))} \tag{3.4}$$

The Hjorth features are Activity (3.11), Mobility (3.12), and Complexity (3.13). Activity is mathematically represented as the square of the standard deviation of the signal, measure of the overall level of signal fluctuations within a given set of time samples. x(n) is the data point at the *n-th* point of the dataset.

Mobility quantifies the dynamic characteristics of the EEG signal. It is computed as the square root of the division of the derivative of the signal's variance to its variance. In this context, *x'(n)* denotes the differential of the EEG signal *x(n)*, and σ symbolizes the variance of the data [11].

Complexity is a metric that offers insights into the signal's structure by providing the derivative of mobility divided by the mobility of the signal. The value of complexity stays on the one to zero range, with a value of one meaning a signal akin to a sine wave.

CNN Classification Algorithm

A key consideration when applying deep learning to EEG classification, and a principle guiding our methodology, is the nature of the input data. It has been observed that feeding deep learning models preprocessed EEG data directly, as opposed to relying on hand-extracted features, often yields comparable or even superior classification accuracy [7]. This allows the network itself to learn the most salient features from the data. Therefore, we feed all channels' preprocessed data to the input layer of the CNNs and SNNs during the training and inference (test) phases. (We can do something different, maybe check the newer papers how do they input eeg data)

Data Loader

All neural networks require a well-defined and labeled input data for training and usage. The MI-EEG data, preprocessed as described in Section 3.2.2, was first structured into epochs using MATLAB and then loaded into the Google Colab Python environment.

Input data format: After preprocessing the data on MATLAB we had a EEG object with continuous EEG recordings for each subject, they were segmented into 4-second trials. For motor imagery tasks namely elbow flexion (MI 1) or hand grasp (MI 2), these 4-second epochs were extracted starting from the onset of the visual cue. For resting-state recordings, consecutive 4-second segments were extracted. Each trial, sampled at 90 Hz after resampling, comprised 360 data points per channel.

The processed data for each subject was saved into a "*SubjectXX_eeg.mat*" file. This file structure encapsulates two primary data structures: the first is a three-dimensional numerical array containing the EEG epochs, with dimensions corresponding to the number of trials, the number of EEG channels (62 channels after the removal of EOG and EMG channels), and the number of samples per trial (360 samples). The second is a corresponding array of class labels, with each entry indicating the motor imagery task (Rest, Elbow, or Hand) associated with each trial.

Loading the data: Within the Python environment, these MATLAB-formatted files were loaded, and the contained EEG epoch data and label arrays were extracted and subsequently converted into PyTorch tensors. Before feeding the data into the neural networks, a class balancing step was performed, primarily by adjusting the number of

'Rest' class samples to mitigate potential biases from class imbalance. The balanced data and labels were then split into training and testing sets (70% training, 30% Test).

A modified version of the data loading function used by [8] was implemented to prepare these datasets for the neural network models. Firstly, the class labels are converted to a numerical tensor format suitable for loss calculation, flattened into a one-dimensional array, and allocated to the designated computation device (CPU or GPU). Secondly, the input EEG data tensor, initially structured with dimensions for samples, channels, and time points, is processed. While a check exists to reorder dimensions if channels exceed time points, this is typically not the case for EEG data (e.g., 62 channels versus 360 time points per epoch). Thirdly, an additional singleton dimension is introduced to the data tensor, resulting in a two-dimensional structure (sample number, 1, EEG channel number, EEG data). This specific data format ($N_{number\ of\ samples} \times 1 \times C_{number\ of\ channels} \times L_{lenght\ per\ sample}$) is required by the two-dimensional convolutional layers within the LENet CNN architecture, treating the multi-channel EEG epoch as a single-channel input image where EEG channels and time points form the spatial dimensions. The data tensor is also converted to a floating-point numerical type and moved to the selected computation device (The Tesla T4 GPUs using CUDA acceleration).

The processed data and label tensors are encapsulated into a PyTorch compatible dataset structure. This structure is then utilized by a data loader utility. The data loader function facilitates efficient data handling by enabling batching of samples, shuffling of the training data to ensure robust learning, and optionally discarding any final, incomplete batch to maintain consistency during training iterations. The output of this data preparation pipeline is an iterable data loader object, which efficiently feeds data in specified batch sizes to the CNN during both the training and evaluation phases.

Core Components: The CNN implementation relies on a set of specialized functions to handle data preparation, model training, evaluation, and the ANN-to-SNN conversion process. These functions work in conjunction with two primary Convolutional Neural Network (CNN) architectures, $LENet_{CCB}$ and $LENet_{FCL}$, which serve as the basis for the subsequent SNN conversion. Several helper functions were defined to streamline the experimental workflow. The data loading mechanism, detailed previously, prepares and

batches the EEG data for PyTorch. For evaluating Spiking Neural Networks, a specific validation function was implemented. This function calculates the SNN's accuracy on a test dataset, crucially incorporating the temporal dynamics of SNNs by accumulating outputs over a defined number of time steps (T) and resetting neuron states between input samples to ensure independent processing.

The conversion from a trained Artificial Neural Network (ANN) to an SNN is handled by another function. It takes a pre-trained CNN, along with training and testing data, and employs the SpikingJelly library's ANN2SNN Converter function, using a 'max normalization' mode, which scales weights based on the maximum activation observed during a pass over the training data, to transform the ANN into an SNN. The accuracy of the resulting SNN is then evaluated over the specified time steps.

To promote stable and effective training, a weight initialization function was used. This function applies Kaiming Normal initialization to the weights of convolutional layers, a common practice for layers followed by ReLU-like activations and initializes weights and biases of batch normalization layers to standard values (1 for weights, 0 for biases).

CNN Model Architectures

Two variants of a lightweight CNN architecture, inspired by the LENet concept proposed by [8] for efficient EEG classification and SNN conversion, were implemented:

Firstly, the LENet with the Classification Convolution Block (CCB) Variant is designed to capture spatiotemporal features from EEG signals while maintaining a relatively small number of parameters, which is beneficial for ANN-to-SNN conversion. Its structure comprises several specialized blocks that can be visualized on Fig. 38.

Temporal Convolution Blocks (TCB): It receives the initial EEG data as an input, initial stage consists of three parallel TCBs, each employing a 2D convolution with a kernel that spans only the temporal dimension (e.g., $(1 \times 64), (1 \times 32), (1 \times 16)$). This allows the network to capture temporal patterns at different scales from the EEG channels input. Each TCB is followed by batch normalization.

TCB Fusion: The outputs from the three parallel TCBs are concatenated along the feature channel dimension and then passed through a (1×1) convolution, also followed by batch normalization, to fuse these multi-scale temporal features.

Spatial Convolution Block (SCB): This block applies a depth wise-like convolution where the kernel height matches the number of input channels (derived from the TCB fusion). This is designed to learn spatial filters across the EEG channels. It includes batch normalization, a ReLU activation, average pooling to reduce temporal dimensionality, and dropout for regularization.

Feature Fusion Convolution Block (FFCB): This block further processes the features using a depth wise convolution along the temporal dimension, followed by a pointwise (1×1) convolution to mix features across channels. It also includes batch normalization, ReLU activation, average pooling, and dropout.

Classification Convolution Block (CCB): Instead of traditional fully connected layers, this model uses a (1×1) convolutional layer to map the features to the number of output classes. An adaptive average pooling layer then reduces each feature map to a single value, followed by a flatten operation to produce the final class scores.

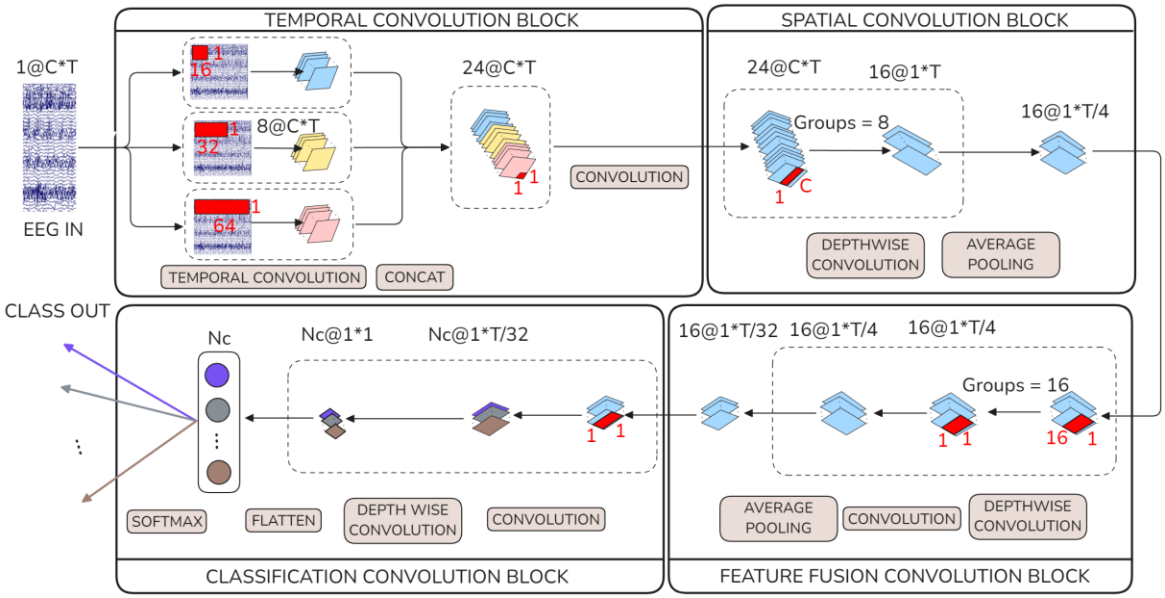


Fig. 3. *The LENet architecture [8]: a visual representation of each convolution block, with their input and output dimensions. C is the number of channels, T is the number of time points, N_C is the number of classes.S*

Secondly, for comparison, we tested the LENet Fully Connected Layer Variant FCL, it shares the same feature extraction backbone as the $LENet_{CCB}$ variant, including the parallel TCBs, TCB fusion, SCB, and FFCB. The primary difference lies in the classification head. Instead of the CCB, the output from the FFCB is flattened into a one-dimensional vector. This vector is then fed into a standard Linear (fully connected) layer, which maps the flattened features to the number of output classes. This fully connected layer is dynamically initialized during the first forward pass to match the size of the incoming flattened feature vector. Both models allow tuning of the number of output classes, the input channel count, and the dropout rate. The choice between the CCB and a traditional fully connected layer for classification offers a point of comparison in terms of performance and suitability for SNN conversion. These functions and model architectures provide the framework for training ANNs on the MI-EEG data and subsequently converting them into SNNs for performance and efficiency evaluation.

Training the Models

We begin with the training of the CNNs ($LENet_{CCB}$ and $LENet_{FCL}$. The preprocessed MI-EEG dataset was first divided into training and testing subsets, typically with a 70/30 split. Model weights were initialized using Kaiming Normal initialization. The ANNs were then trained for a set number of epochs, using a batch-wise approach where data was fed in segments.

The core CNN training loop utilized the Adam optimizer to minimize the Cross-Entropy Loss, a standard objective for multi-class classification. A Cosine Annealing learning rate scheduler dynamically adjusted the learning rate throughout training to aid convergence. Each training iteration involved a forward pass to generate predictions, followed by computing the loss against the true labels. Backpropagation then calculated gradients, which the optimizer used to update the network's weights. Training progress was monitored by tracking loss and accuracy. Critically, after each epoch, the model's generalization was assessed by evaluating its performance on the unseen test dataset, and the resulting test accuracy and loss were recorded.

Resulting in two neural networks trained for the specific dataset subject EEG-MI patterns ($LENet_{CCB}, LENet_{FCL}$). For these two models, key performance metrics including classification accuracy, and detailed confusion matrices were subsequently computed; these results and their implications will be discussed in Section 3.

Summary

MI Classification Accuracy

After tuning the hyperparameters (batch size, number of epochs, drop ratio and time steps, the other hyperparameters were fixed to comply with the LENet architecture) we ran the optimal trained CNN and SNN models ($EPOCHS = 500, BATCH\_SIZE = 64, TIME\_STEPS = 100, TEST\_SIZE = 0.2, DROPOUT = 0.35$) on 3 subjects from the dataset [3], 4 confusion matrices were plot for each subject accessing each model performance, they can be seen on the Fig. 39.

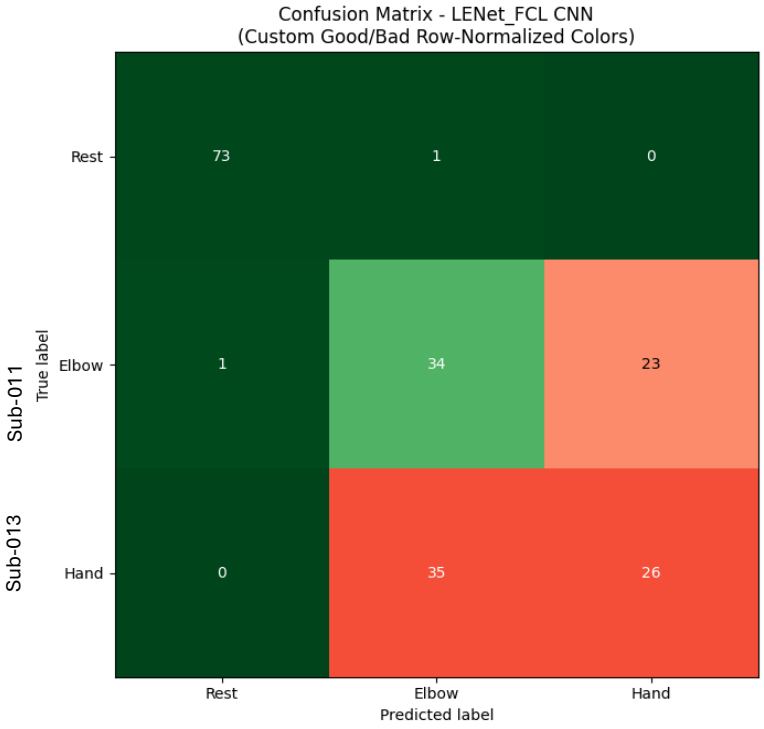
The row accuracy of the confusion matrices was calculated (Table. 2).

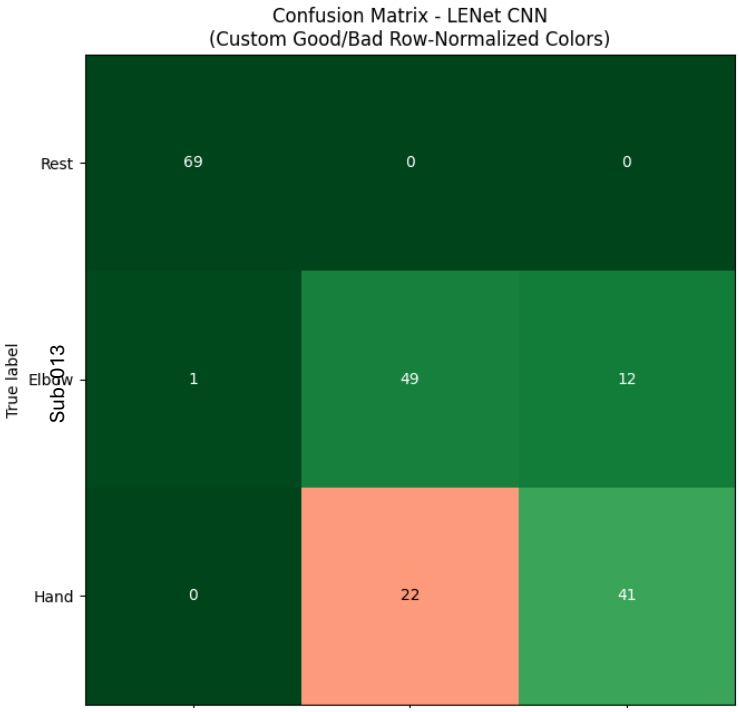
Table. 1. Neural Networks accuracy (Acc) evaluation data across subjects.

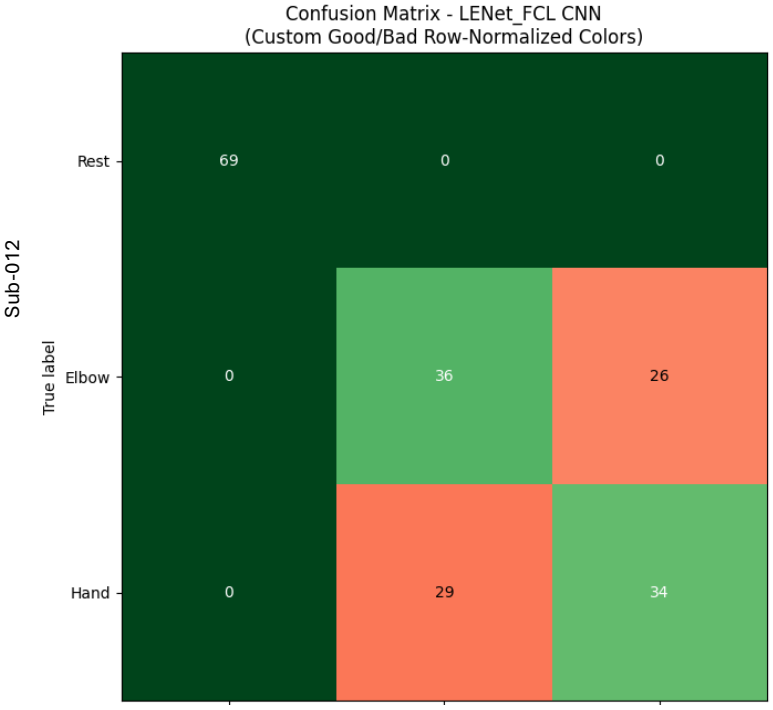
Model	Accuracy	Sub-011 (%)	Sub-012 (%)	Sub-013 (%)	Mean (%)	SD (%)
$LENet_{CCB}$	Overall	77.72	81.96	72.02	77.23	4.99
	Rest	100.00	100.00	98.65	99.55	0.78
	Elbow	61.22	79.03	51.72	63.99	13.86
	Hand	60.66	65.08	59.02	61.59	3.13
$LENet_{FCL}$	Overall	78.24	71.65	68.91	72.93	4.80
	Rest Acc.	97.59	100.00	98.65	98.75	1.21
	Elbow Acc.	55.10	58.06	58.62	57.26	1.89
	Hand Acc.	70.49	53.97	42.62	55.69	14.01

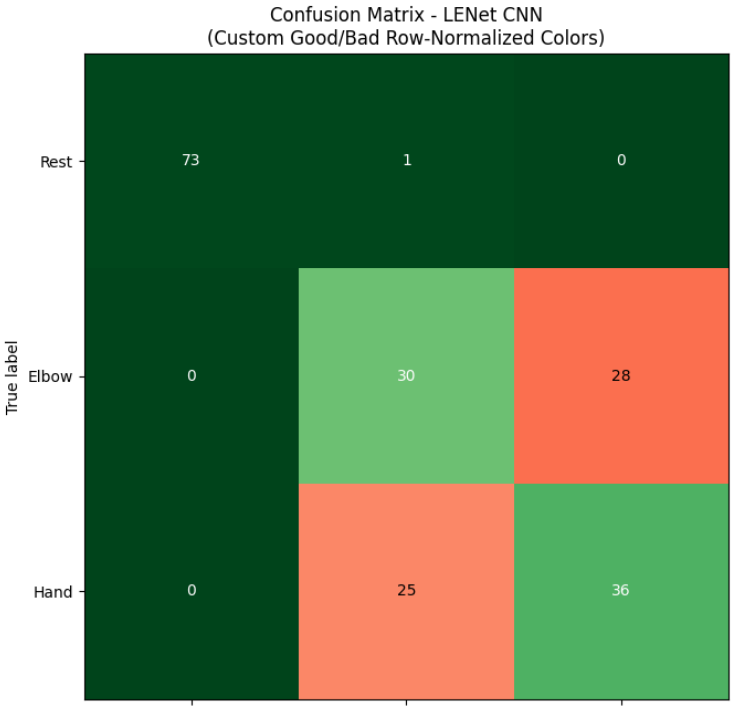
Table. 1. accuracy benchmark comparison with other SoA models for the [3] dataset

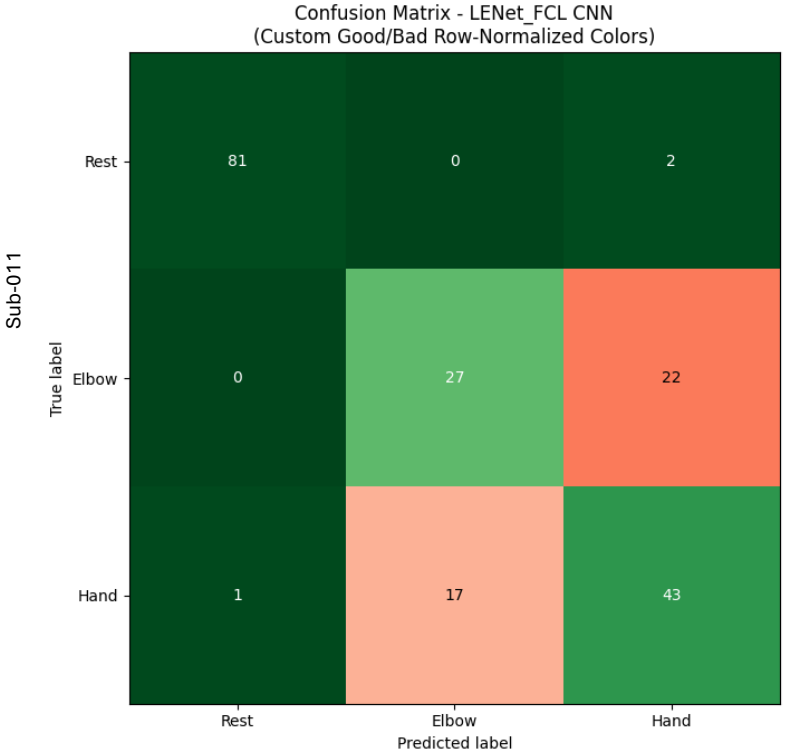
Model	Accuracy (%)	STD (%)
CAMLP-Net [9]	77.44	0.57
$\boldsymbol{LENet_{CCB}}$	77.23	4.99
MLP-Mixer [9]	76.51	0.77
Corr+CNN [9]	75.03	3.37
DeepConvNet [9]	73.07	1.44
$\boldsymbol{LENet_{FCL}}$	72.93	4.80
TSception [9]	71.97	0.82
$\boldsymbol{SNN_{FCL}}$	71.90	3.95
EEGNet [9]	70.44	1.18
FBCSP+SVM [8]	68.68	2.44


Confusion Matrix - LENet_FCL CNN
(Custom Good/Bad Row-Normalized Colors)
Sub-011
Sub-013
True label
Rest
Elbow
Hand
73
1
0
1
34
23
0
35
26
Rest
Elbow
Hand
Predicted label


Confusion Matrix - LENet CNN
(Custom Good/Bad Row-Normalized Colors)
Sub-013
True label
Rest
Elbow
Hand
69
0
0
1
49
12
0
22
41


Confusion Matrix - LENet_FCL CNN
(Custom Good/Bad Row-Normalized Colors)
Sub-012
True label
Rest
Elbow
Hand
69
0
0
0
36
26
0
29
34


Confusion Matrix - LENet CNN
(Custom Good/Bad Row-Normalized Colors)
True label
Rest
Elbow
Hand
73
1
0
0
30
28
0
25
36


Confusion Matrix - LENet_FCL CNN
(Custom Good/Bad Row-Normalized Colors)
Sub-011
True label
Rest
Elbow
Hand
81
0
2
0
27
22
1
17
43
Rest
Elbow
Hand
Predicted label

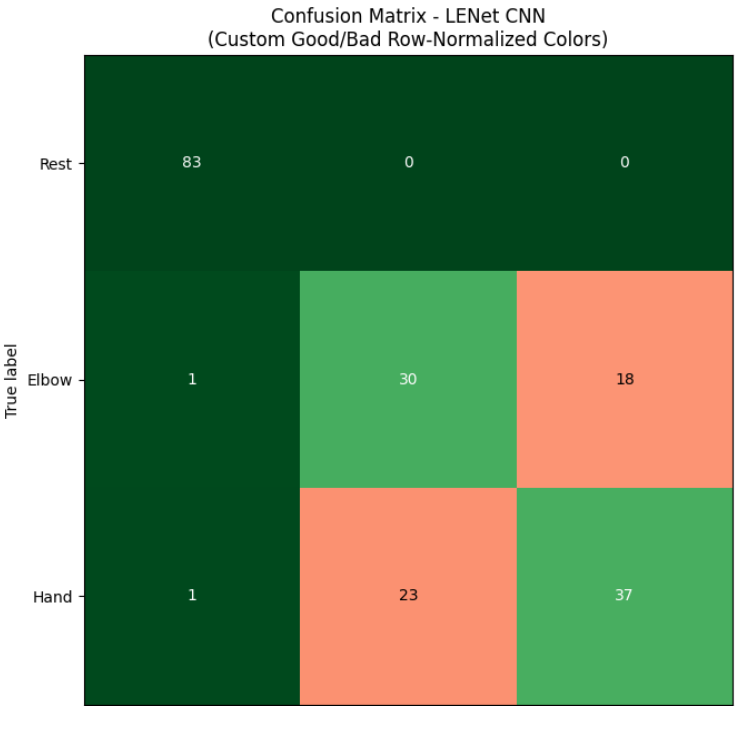

Confusion Matrix - LENet CNN
(Custom Good/Bad Row-Normalized Colors)
True label
Rest
Elbow
Hand
83
0
0
1
30
18
1
23
37

Fig. 1. *Confusion matrices for subjects 11, 12, 13. On each matrix, the rows are the true label as recorded in the dataset, the columns are the labels each respective NN predicted. Both columns and rows are in order the Rest, Elbow and Hand classes. The numbers on each cell represent the number of epochs (i.e., recorded and trimmed labeled data ,4s of MI or Rest eeg data for the given dataset) that the true label of the row was classified with the column predicted label. The matrices are custom color coded, gradient from green (cell row accuracy >50% classified correctly, or <20% incorrect classification) to red (cell row accuracy <50% classified correctly, or >20% incorrect classification).*

The results presented demonstrate the successful application of both CNN architecture models for MI-EEG classification across three subjects. The confusion matrices in Fig. 39 and the aggregated accuracies in Table. 2 provide a detailed view of model performance.

Discussion

The research explored two lightweight CNN architectures, $LENet_{CCB}$ and $LENet_{FCL}$ evaluating their performance on a publicly available MI dataset [3] with 3 classes from the same limb (rest, elbow and hand).

The results presented in the Sections 3.2 and 3.3 demonstrate that SNNs can indeed achieve classification accuracies comparable to their traditional CNN counterparts.

When compared to other state-of-the-art models benchmarked on the same dataset [3] (Table. 3), the $LENet_{CCB}$ model demonstrated competitive mean overall accuracies, ranking highly among the compared methods. This validates the chosen LENet architecture, particularly the version employing a Classification Convolution Block (CCB), as a suitable model for this type of EEG data. However, it is noted that the standard deviation of accuracy for the proposed models (e.g., 4.99% for $LENet_{CCB}$) was higher than that reported for some other leading models, such as CAMLP-Net [9] (0.57% STD), we suggest that this variability was due to a short training (epochs =500). The CCB variants consistently outperformed the FCL variants by approximately 4-5% in mean overall accuracy, suggesting that the convolutional classifier might be more effective at extracting and classifying the spatiotemporal features inherent in MI-EEG signals than a traditional fully connected layer.

The per-class accuracy analysis (data in Fig. 39 and analysis in Table. 2) corroborate with a common pattern in MI-BCI research: high accuracy for the 'Rest' class (mean >98.7%) but more modest performance in distinguishing between the 'Elbow' and 'Hand' MI tasks (means ranging roughly from 54% to 65%). This highlights the inherent difficulty and subtlety of discriminating between different imagined limb movements based on EEG. The observed standard deviations in accuracy across subjects also underscore the significant inter-subject variability that remains a challenge in BCI development.

Limitations and Future Work

While the results are promising, this study has several limitations. The evaluation was conducted on data from three subjects. Although this is a common practice for initial investigations with this dataset, a larger and more diverse subject pool would be necessary for more robust validation and to better understand the generalizability of the models. The observed higher standard deviation in accuracy across subjects for the trained models compared to some benchmarks is a limitation. This variability could be associated with the fixed number of training epochs (500) used for all subjects. As the MI-EEG data is complex and noisy, 500 epochs might still be on the cusp of optimal convergence, leading to more varied final model states. The extensive computational resources and time required to explore a wider range of epoch numbers or implement subject-specific adaptive training durations made such investigations unviable within the timeframe of this thesis.

Future research should therefore include evaluating the models on a larger cohort and investigating strategies to mitigate inter-subject performance variability. This could involve exploring subject-specific training regimens, such as adaptive stopping criteria for epochs based on validation performance, or employing more advanced regularization techniques. These studies would be essential to evaluate the naturalness and intuitiveness of such a system.

Conclusion

This thesis successfully demonstrated that lightweight can achieve competitive MI-EEG classification accuracy comparable to other state-of-the-art models. The $LENet_{CCB}$ architecture, in particular, emerged as an effective and efficient model for this task.

References

[1] C. R. Pernet *et al.*, “EEG-BIDS, an extension to the brain imaging data structure for electroencephalography,” *Sci Data*, vol. 6, p. 103, June 2019, doi: 10.1038/s41597-019-0104-8.

[2] A. Delorme and S. Makeig, “EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis,” *J Neurosci Methods*, vol. 134, no. 1, pp. 9–21, Mar. 2004, doi: 10.1016/j.jneumeth.2003.10.009.

[3] X. Ma, S. Qiu, and H. He, “Multi-channel EEG recording during motor imagery of different joints from the same limb,” *Sci Data*, vol. 7, p. 191, June 2020, doi: 10.1038/s41597-020-0535-2.

[4] A. Saibene, M. Caglioni, S. Corchs, and F. Gasparini, “EEG-Based BCIs on Motor Imagery Paradigm Using Wearable Technologies: A Systematic Review,” *Sensors*, vol. 23, no. 5, Art. no. 5, Jan. 2023, doi: 10.3390/s23052798.

[5] H. Altaheri *et al.*, “Deep learning techniques for classification of electroencephalogram (EEG) motor imagery (MI) signals: a review,” *Neural Comput & Applic*, vol. 35, no. 20, pp. 14681–14722, July 2023, doi: 10.1007/s00521-021-06352-5.

[6] B. Hjorth, “EEG analysis based on time domain properties,” *Electroencephalography and Clinical Neurophysiology*, vol. 29, no. 3, pp. 306–310, Sept. 1970, doi: 10.1016/0013-4694(70)90143-4.

[7] A. Craik, Y. He, and J. L. Contreras-Vidal, “Deep learning for electroencephalogram (EEG) classification tasks: a review,” *J. Neural Eng.*, vol. 16, no. 3, p. 031001, Apr. 2019, doi: 10.1088/1741-2552/ab0ab5.

[8] X. Liao, G. Li, Y. Wang, L. Sun, and H. Zhang, “Constructing lightweight and efficient spiking neural networks for EEG-based motor imagery classification,” *Biomedical Signal Processing and Control*, vol. 100, p. 107000, Feb. 2025, doi: 10.1016/j.bspc.2024.107000.

[9] Y. He, Z. Lu, J. Wang, and J. Shi, “A Channel Attention Based MLP-Mixer Network for Motor Imagery Decoding With EEG,” in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 1291–1295. doi: 10.1109/ICASSP43922.2022.9747488.