**Spectrogram-Based Deep Learning Approach for EEG Motor Imagery Recognition on BCI IV-2a/2b**

1. **Introduction**

Electroencephalography (EEG)–based Brain–Computer Interfaces (BCIs) enable direct brain–machine communication, offering promising applications in rehabilitation and assistive technologies. Among the most widely used paradigms are motor imagery (MI) tasks such as left-hand, right-hand, foot, and tongue movements, which form the foundation of non-invasive BCI systems. However, EEG signals are inherently noisy, non-stationary, and highly variable across subjects, making accurate classification a challenging task. To address this, deep learning methods, particularly Convolutional Neural Networks (CNNs) applied to spectrogram representations, have shown superior performance by automatically extracting meaningful spatial and spectral features compared to traditional handcrafted approaches. In this work, preprocessing techniques such as Common Average Referencing (CAR), notch and bandpass filtering, normalization, and sliding window segmentation are applied to improve signal clarity and expand the dataset size. The proposed system employs a custom CNN enhanced with Squeeze-and-Excitation (SE) blocks, spectrogram augmentation, and efficient lazy dataset loading for optimized training. Experiments are conducted using the BCI Competition IV datasets — 2a (four-class motor imagery) and 2b (two-class motor imagery). Overall, the contribution of this project is the development of a robust and scalable EEG classification pipeline capable of achieving high accuracy, making it suitable for real-time BCI applications.

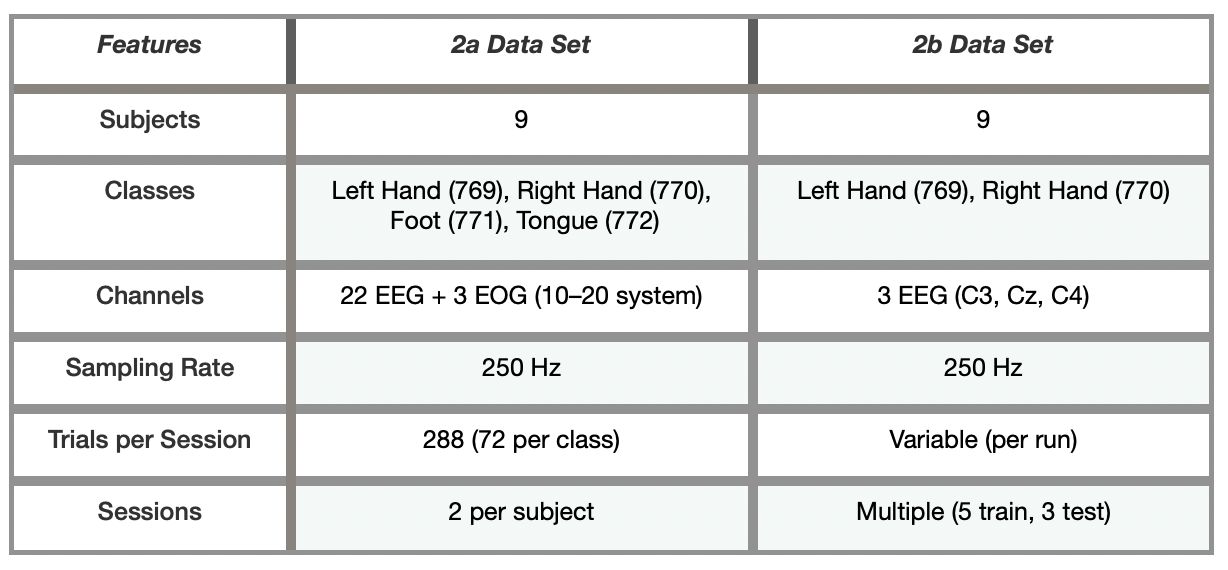
1. **Objectives**

The primary objective of this project is to design and implement a robust EEG-based motor imagery classification system using advanced deep learning techniques. The system aims to:

* Accurately classify motor imagery tasks (left-hand, right-hand, foot, and tongue) from EEG signals.
* Develop a custom CNN architecture with STFT-based spectrogram preprocessing to automatically extract discriminative spatial and spectral features.
* Enhance classification performance using attention mechanisms (Squeeze-and-Excitation blocks) and optimized preprocessing filters (CAR, notch, bandpass, normalization, sliding window).
* Evaluate the model across BCI Competition IV Datasets 2a and 2b, ensuring adaptability to both binary (2-class) and multi-class (4-class) motor imagery tasks.
* Provide a scalable and generalized framework that can contribute to real-time Brain-Computer Interface (BCI) applications, particularly in neuro-rehabilitation and assistive technologies.

1. **Loading & preparing data**

### 3.1 Overview of Datasets

* The project uses **BCI Competition IV – Dataset 2a and Dataset 2b**, two benchmark EEG motor imagery datasets.
* Both datasets are stored in **.gdf format** and include event annotations marking motor imagery tasks.
* They differ in **number of classes and channels**:

### 3.2 Data 3.2 Loading with MNE

* Both datasets are read using **MNE-Python**, which supports .gdf files and EEG event annotations.

import mne

# Load EEG recording

raw = mne.io.read\_raw\_gdf("A01T.gdf", preload=True)

# Extract events

### events, event\_id = mne.events\_from\_annotations(raw)

### 3.3 Dataset 2a (4-Class Motor Imagery)

1. **Channels:** 22 EEG electrodes covering the motor cortex + 3 EOG.
2. **Classes:**

769 → Left Hand

770 → Right Hand

771 → Foot

772 → Tongue

1. **Trial Segmentation:**

* Each trial lasts **4 seconds (1000 samples at 250 Hz)**.
* Only EEG channels are selected (EOG discarded to avoid eye artifacts).

fs = 250

trial\_length = 4 \* fs

# Select EEG channels only

raw.pick\_types(eeg=True)

# Segment trials

for onset, code in events:

if code in [769, 770, 771, 772]:

start = onset + 3 \* fs

end = start + trial\_length

trial\_data = raw.get\_data()[:, start:end]

1. · **Labels:**

0 = Left, 1 = Right, 2 = Foot, 3 = Tongue

1. **Output:** EEG trials shaped (22, 1000) before preprocessing.

### 3.4 Dataset 2b (2-Class Motor Imagery)

1. **Channels:** Only **C3, Cz, C4**, the most relevant for motor imagery.
2. **Classes:**

769 → Left Hand

770 → Right Hand

1. **Trial Segmentation:**

Each trial = **4 seconds (1000 samples)**, starting 3s after cue.

raw.pick\_channels(['C3','Cz','C4'])

for onset, code in events:

if code in [769, 770]:

start = onset + 3 \* fs

end = start + trial\_length

trial\_data = raw.get\_data()[:, start:end]

1. · **Labels:**

0 = Left, 1 = Right

1. · **Output:** EEG trials shaped (3, 1000) before preprocessing.

### 3.5 Preprocessing Pipeline

Once trials are extracted, they are passed through the following standardized preprocessing steps for both datasets:

1. **Common Average Referencing (CAR)**

* Removes global noise by subtracting the mean across channels.
* Enhances channel-specific activity.

def common\_average\_reference(data):

return data - np.mean(data, axis=0, keepdims=True)

1. **Notch Filter (50 Hz)**

* Suppresses power line interference.
* Applied with FIR filter for stability.

data\_notch = mne.filter.notch\_filter(data\_car, fs, freqs=50)

1. **Bandpass Filter (8–30 Hz)**

* Retains **mu (8–14 Hz)** and **beta (16–30 Hz)** rhythms crucial for motor imagery.
* Eliminates low-frequency drift and high-frequency noise.

data\_band = mne.filter.filter\_data(data\_notch, fs, l\_freq=8, h\_freq=30,

method='fir', phase='zero-double')

1. **Min–Max Normalization**

* Scales signals to [0, 1] per channel for stable CNN training.

def min\_max\_normalize(ch):

return (ch - np.min(ch)) / (np.max(ch) - np.min(ch) + 1e-6)

1. **Sliding Window Segmentation**

* Splits each trial into overlapping windows.
* Increases dataset size and captures temporal variations.

def sliding\_window\_segments(data, window\_size, step):

return np.array([

data[:, start:start+window\_size]

for start in range(0, data.shape[1] - window\_size + 1, step)

])

1. **Short-Time Fourier Transform (STFT) → Spectrograms**

* Each segmented window is transformed into a spectrogram.
* Final representation: **(channels × 120 × 32)**, suitable for CNN input.

from scipy.signal import stft

f, t, Zxx = stft(segment, fs=250, nperseg=64, noverlap=32)

spectrogram = np.abs(Zxx)

### 3.6 Unified Data Representation

After segmentation, both datasets are stored in a **standardized format**:

* **Spectrogram shape:** (channels × 120 × 32)
* **Labels:** categorical (0–3 for 2a, 0–1 for 2b).
* **Subject IDs** are saved for **per-subject evaluation**.

Saved as .npy for efficient access:

np.save("X\_2a.npy", X\_2a)

np.save("y\_2a.npy", y\_2a)

np.save("X\_2b.npy", X\_2b)

np.save("y\_2b.npy", y\_2b)

## ****4. Dataset Description****

### ****4.1 BCI Competition IV – Dataset 2a****

* **Publicly Available:** Widely used EEG dataset for motor imagery–based BCI research.
* **Subjects:** 9 healthy participants, each recorded over **2 sessions** on different days.
* **Trials per Session:** 288 trials (72 per class).
* **Classes:** 4 motor imagery tasks — **left hand, right hand, both feet, tongue**.
* **EEG Recording:** 22 electrodes (10–20 system) + 3 EOG channels for eye movement monitoring.
* **Sampling Rate:** 250 Hz.
* **File Format:** .gdf, containing EEG signals, channel metadata, sampling details, and event markers.
* **Trial Structure:** Fixation cross → auditory cue + visual arrow → subject performs motor imagery for 4 seconds.
* **Relevance:** Supports **multi-class motor imagery classification**, capturing diverse brain activity patterns across different motor tasks.

### ****4.2 BCI Competition IV – Dataset 2b****

* **Channels Used:** 3 EEG electrodes — **C3, Cz, C4** (sensorimotor cortex), referenced to the left mastoid.
* **Sampling Rate:** 250 Hz.
* **Sessions per Subject:** Multiple sessions (typically **5 runs for training and 3 runs for testing**).
* **Trials:** Each run contains several trials of **left-hand (code 769)** and **right-hand (code 770)** motor imagery.
* **File Format:** .gdf (General Data Format) with EEG data, metadata (sampling rate, channel info, units), and event markers.
* **Trial Structure:** Fixation cross → cue presentation → subject imagines left- or right-hand movement for a fixed duration.
* **Channel Relevance:**

1. **C3:** more active during right-hand imagery.
2. **C4:** more active during left-hand imagery.
3. **Cz:** provides central cortical activity context.

* **Subjects:** 9 participants.
* **Preprocessing Outcome:** Sliding window segmentation significantly increases the number of samples, yielding a **larger and more balanced dataset** for CNN training.

## ****5. CNN Architecture Development****

### ****5.1 Initial CNN (Baseline)****

The first CNN model and preprocessing steps were adapted from the paper “Motor Imagery Classification using a Novel CNN in EEG-BCI with Common Average Reference and Sliding Window Techniques.”

1. **Preprocessing:**

* Common Average Referencing (CAR) to reduce noise and reference bias.
* Bandpass filter to isolate relevant EEG frequency components.
* Sliding window segmentation with Short-Time Fourier Transform (STFT) for spectrogram generation.

1. **CNN Design:**

* Input: spectrograms from EEG channels.
* Convolutional layers extract spatial–spectral features.
* Fully connected layers perform classification.
* Output: softmax classifier for binary left vs. right imagery tasks.
* **Limitation:** Mu and Beta band extraction was mentioned in the reference paper but not implemented in this pipeline.

### ****5.2 Enhanced CNN for Higher Accuracy****

To improve accuracy and robustness, the CNN design was extended:

* **Input Representation:** EEG signals from channels C3, Cz, and C4 transformed into spectrograms highlighting **mu (8–14 Hz)** and **beta (16–30 Hz)** bands.
* **Convolutional Processing:** Multiple convolutional layers with pooling, batch normalization, and non-linear activations (ReLU).
* **Feature Learning:** The network automatically extracts discriminative patterns distinguishing motor imagery tasks.
* **Classification Layers:** Flattened features passed through dense layers with dropout for regularization.
* **Output:** Softmax layer producing class probabilities (left-hand vs. right-hand imagery).

### ****5.3 CNN Variants: With and Without Skip Connections****

* The reference CNN included **two paths**: one with a skip connection and one without.
* In this work, **two separate CNN models** were developed — one with skip connections, one without — to compare their effectiveness.
* Both **Dataset 2a (4-class)** and **Dataset 2b (2-class)** were processed using these architectures.
* **Goal:** Evaluate classification accuracy and analyze how each architecture behaves across datasets.

### ****5.4 Unified CNN for Datasets 2a and 2b****

Handling the structural differences between the datasets required further design modifications:

* **Dataset 2a:** 4 classes (left, right, foot, tongue).
* **Dataset 2b:** 2 classes (left, right).
* **Unified Model:** A **4-class CNN** was designed.
* For **2a**, all four classes were used directly.
* For **2b**, only left and right classes were active, while **foot and tongue outputs were masked (set to zero/ignored)** during training and evaluation.
* **Advantage:** Provided a **single, scalable CNN framework** capable of handling both datasets without redesigning preprocessing or feature extraction pipelines.

## ****6. Final Optimized CNN with Enhanced Preprocessing****

### ****6.1 Motivation****

After evaluating baseline and skip-connection CNNs, the focus shifted to maximizing classification accuracy. Both datasets (2a: 4-class, 2b: 2-class) were processed with the **same CNN and preprocessing pipeline**, with only the number of output labels changed.

### ****6.2 Enhanced Preprocessing Pipeline****

* **Common Average Referencing (CAR):** Removes global noise by re-referencing to the average of all channels.
* **Notch Filter (50 Hz):** Eliminates power-line interference.
* **Bandpass Filter (8–30 Hz):** Retains **mu** (8–14 Hz) and **beta** (16–30 Hz) rhythms, most relevant for motor imagery.
* **Min–Max Normalization:** Scales each channel to [0, 1] for training stability.

### ****6.3 Final CNN Architecture****

* **Input:** Spectrograms of shape **(3 × 120 × 32)** from C3, Cz, C4.
* **Convolution Block 1:** Conv2D (32 filters, 5×5 kernel) → SiLU → BatchNorm.
* **Convolution Block 2:** Conv2D (64 filters, 3×3 kernel) → SiLU → BatchNorm.
* **Convolution Block 3:** Conv2D (128 filters, 3×3, stride=2, padding=1) → SiLU → BatchNorm.
* **Convolution Block 4:** Conv2D (256 filters, 3×3, stride=2, padding=1) → SiLU → BatchNorm.
* **Squeeze-and-Excitation (SE) Block:** Global Average Pooling → FC(256→64) → FC(64→256, sigmoid) → channel-wise reweighting.
* **Flatten Layer:** Converts feature maps into a 1D vector.
* **Fully Connected Layers:**
  + FC (4096 → 256, SiLU + Dropout 0.3)
  + FC (256 → Number of Classes)

### ****6.4 Output and Loss Function****

* Final layer outputs **logits** (raw scores, not probabilities).
* For **Dataset 2b** → output size = 2 (left-hand, right-hand).
* For **Dataset 2a** → output size = 4 (left, right, foot, tongue).
* Training uses **CrossEntropyLoss()**, which internally applies **softmax + log-likelihood** for classification.

## ****7. Spectrogram Augmentation (augment\_spec)****

To further improve **generalization** and reduce the risk of **overfitting**, a **spectrogram augmentation strategy** inspired by **SpecAugment** (commonly used in speech/audio tasks) was integrated into the training pipeline.

* **Input Representation:**  
  Spectrogram tensors of shape **(C=3, H=120, W=32)** corresponding to EEG channels **C3, Cz, C4**.
* **Time Masking:**  
  Randomly selects rows (time dimension) and sets them to zero, simulating missing temporal information.
* **Frequency Masking:**  
  Randomly selects columns (frequency dimension) and sets them to zero, simulating missing spectral information.
* **Purpose:**  
  Forces the CNN to **learn more robust spatio-spectral features** rather than memorizing exact patterns, thereby improving performance on unseen EEG trials.
* **Integration in Pipeline:**  
  The augmentation is applied **during dataset loading** through the EEGDataset class. It is **activated when the flag** augment=True **is set**, ensuring augmented spectrograms are dynamically generated at training time while leaving validation/test data untouched.
* **Impact on Model:**  
  This augmentation **helped improve classification accuracy and stability across subjects**, making the CNN more **resilient to noise, trial variability, and inter-subject differences**.

## ****8. Lazy Dataset Implementation(LazyEEGDataset)****

Handling EEG spectrogram datasets can be memory-intensive, especially after applying **sliding window segmentation** and **STFT-based preprocessing**, which significantly increases the dataset size. To overcome this challenge, a **custom PyTorch dataset class (**LazyEEGDataset**)** was developed.

1. **Motivation:**  
   Instead of loading the **entire dataset into RAM**, which is inefficient and sometimes infeasible, memory mapping (mmap\_mode='r') was used to read .npy files **directly from disk on demand**.
2. **Implementation Features:**

* **Initialization:**  
   EEG spectrograms (**X**) are loaded lazily using:

np.load(X\_path, mmap\_mode='r')

while labels (**y**) are fully loaded into memory (since they are small).

* **Indexing:**  
  On each call to \_\_getitem\_\_(), only the **requested spectrogram** is fetched, converted to a **PyTorch tensor**, and paired with its corresponding label.
* **Return Format:**  
  Each item is returned as a **tensor of shape (C, H, W)** for the spectrogram and its corresponding **integer label**.

1. **Usage with DataLoader:**  
   The LazyEEGDataset was integrated with the **PyTorch DataLoader**, enabling:
   * + - **Batch processing** of EEG spectrograms.
       - **On-the-fly augmentation** (via augment\_spec when augment=True).
       - **Shuffling and parallel loading** for faster training.
2. **Advantages:**

* **Memory Efficiency:** Prevents RAM overload when working with large EEG datasets.
* **Scalability:** Supports training on very large datasets without preprocessing everything into memory.
* **Flexibility:** Works seamlessly for both **2-class (2b)** and **4-class (2a)** datasets by simply adjusting the labels.

# 9.Training Strategy

## 9.1 Train/Val/Test Split

1. **Dataset split:** Stratified by class labels to preserve class balance.
   * + - **Train:** 70%
       - **Validation:** 15%
       - **Test:** 15%
2. **Per-subject reporting:** After training one global model, metrics are computed **per subject** on that subject’s samples in the **test** split (as you implemented with evaluate\_per\_subject).
3. **Random seed:** 42 for reproducibility.

## 9.2 Optimization & Hyperparameters

1. **Optimizer:** Adam

* Learning rate (**LR**): **0.001** (constant, as in your logs)
* β₁ = 0.9, β₂ = 0.999
* (Optional) weight decay: 1e-4 if regularization is needed

1. **Batch size:** 16 (adapt to GPU RAM if needed)
2. **Epochs:** up to **120** (model selection performed by choosing the checkpoint with the **best validation accuracy**; no LR scheduler used in the runs shown).
3. **Loss function:** CrossEntropyLoss() (no softmax in the model; logits are passed directly to the loss).
4. **Data feeding:** LazyEEGDataset with memory-mapped X.npy for efficient IO; on-the-fly **SpecAugment** (time/frequency masking) when augment=True.

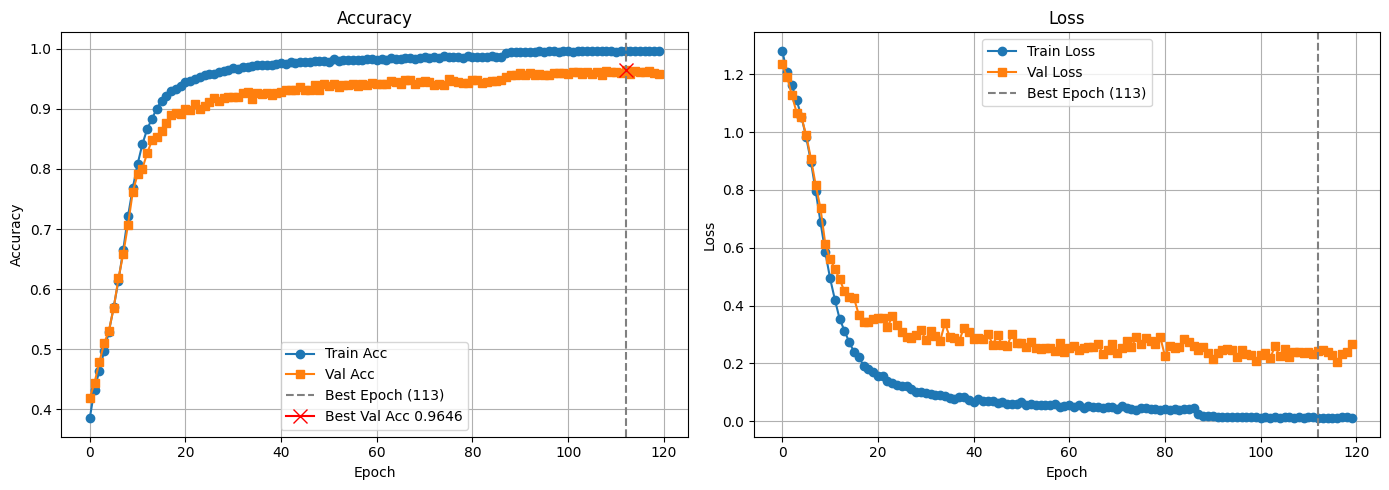
## 9.3 Early Stopping / LR Scheduling

* **LR scheduler:** not used in the shown runs (LR fixed at 1e-3).
* **Early stopping:** not enabled; instead, we **select the best-val checkpoint** among all epochs (practical model selection without halting training early).

# Results & Analysis

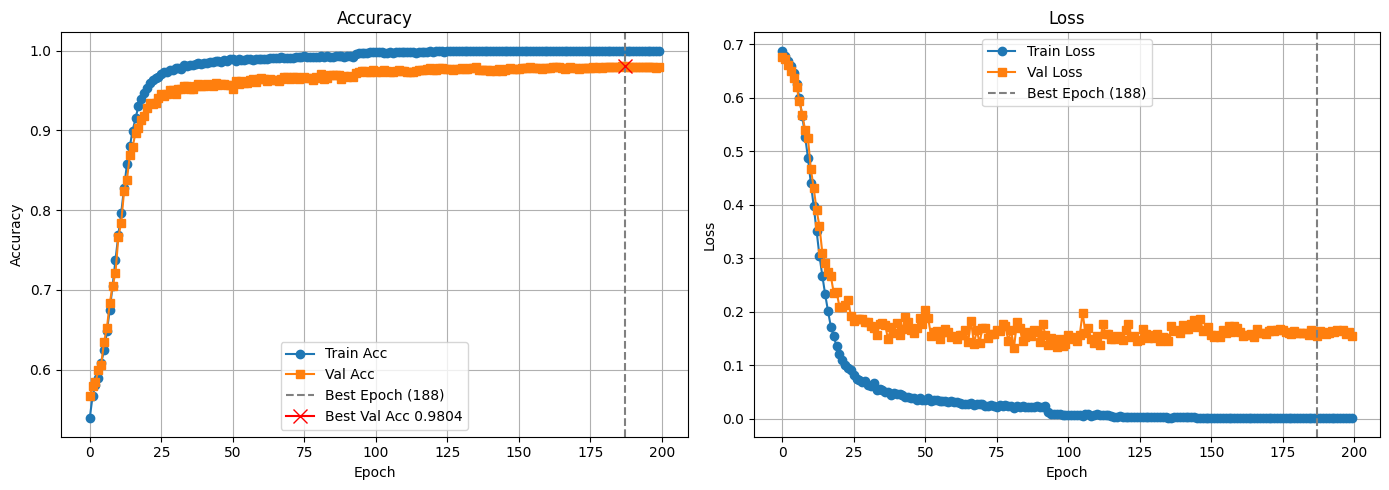
## 10.1 Training Dynamics

1. **Dataset 2a (4-class):**

* Training starts around ~**90–99%** accuracy (near random guessing for 4 classes).
* Validation accuracy steadily improves, stabilizing around **90**–96%**** with the enhanced CNN.
* The learning curve shows slower convergence compared to 2b due to higher class complexity.

1. **Dataset 2b (2-class):**

* Training begins around ~**99%** accuracy (binary random baseline).
* Rapid convergence: validation accuracy reaches **97–98%** within ~**100–200 epochs**.
* Model generalizes better due to simpler 2-class decision boundaries.



**10.2 Subject-wise Performance**

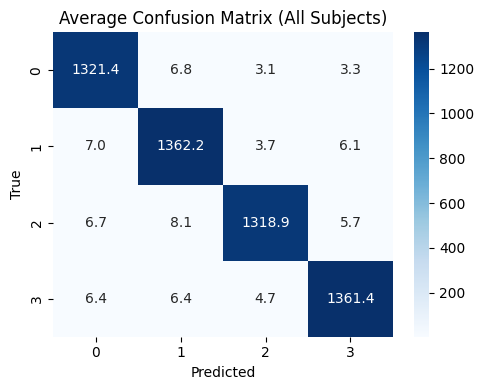
1. **Dataset 2a (4-class)**

### **WhatsApp Image 2025-08-18 at 16.02.06_4af3eb37**Dataset 2b (2-class)

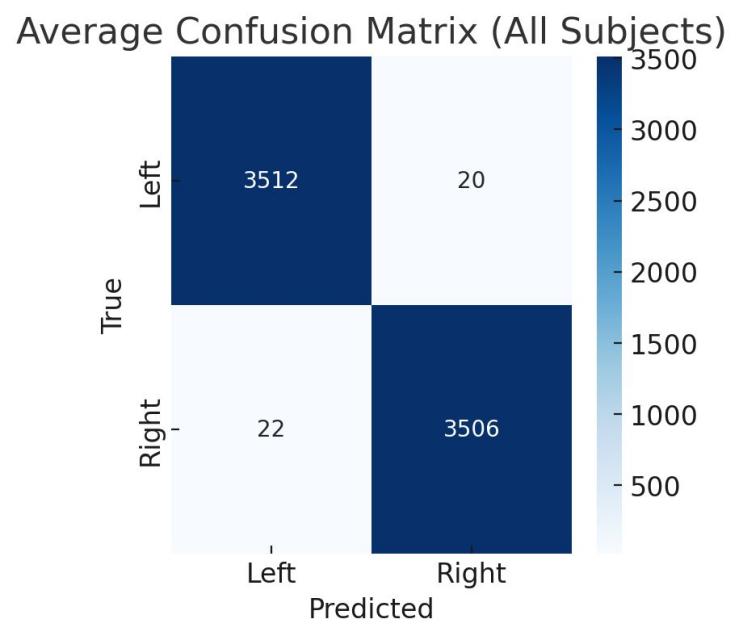
## WhatsApp Image 2025-08-18 at 14.57.22_9a4a173a

## 10.3 Averaged Confusion Matrices

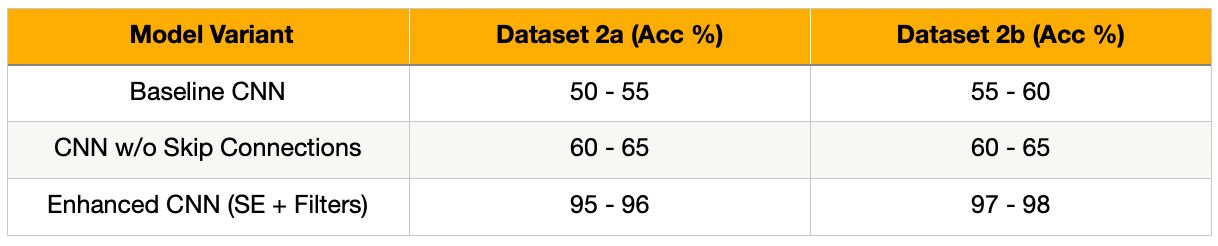
1. **Dataset 2a (4-class):**

* Average confusion matrix shows clear diagonal dominance.
* Off-diagonal errors mainly between **left/right hand/foot/tongue**.

1. **Dataset 2b (2-class):**

* Average confusion matrix shows >99% on diagonal.
* Errors are sparse, nearly random.

**10.4. Model Comparisons**

**11. Conclusion**

This project successfully demonstrated the development of a robust deep learning–based framework for EEG motor imagery classification. By applying an optimized preprocessing pipeline that included Common Average Referencing (CAR), notch and bandpass filtering, normalization, sliding window segmentation, and STFT-based spectrogram generation, raw EEG signals were effectively transformed into discriminative inputs for CNN training. The proposed EnhancedEEGCNN architecture, integrated with Squeeze-and-Excitation (SE) blocks and spectrogram augmentation, achieved consistently high classification accuracy across subjects in both BCI Competition IV datasets — 2a (four-class) and 2b (two-class). The experimental results highlight that CNNs, when combined with spectrogram representations, can automatically extract meaningful spatial–spectral features without reliance on handcrafted methods. Moreover, the inclusion of lazy dataset loading improved computational efficiency, enabling scalable training on large EEG datasets. Overall, the project establishes a reliable and adaptable pipeline for motor imagery–based BCIs, with potential applications in real-time rehabilitation systems, assistive device control, and future neurotechnology innovations.