A Detailed Study on Classification Accuracy and Hyperparameter Tuning of Machine Learning Algorithms in Classical Computing and Quantum Computing Platforms.

Summer Internship Report

Submitted by

Anuj Kumar Kashyap (218248)

Third Year, BTech Computer Engineering

Babasaheb Bhimrao Ambedkar University (A Central University)

LUCKNOW

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING GUINDY

ANNA UNIVERSITY, CHENNAI

JULY 2024

BONAFIDE CERTIFICATE

This Report titled "**A Detailed Study on Classification Accuracy and Hyperparameter Tuning of Machine Learning Algorithms in Classical Computing and Quantum Computing Platforms**" is the Bonafide work of **ANUJ KUMAR KASHYAP** (218248) who carried out the work under my supervision. Certified further that to the best of knowledge, the work reported herein does not form part of any other project or report.

Supervisor

Dr. P. GANESH KUMAR

Associate Professor

Department of Computer Science and Engineering

College of Engineering Guindy

Anna University, Chennai - 600 025

DECLARATION

I **Anuj Kumar Kashyap** Roll No. 218248 , student of **Bachelors of Technology** (**CSE**), a class of 2021-25, **Babasaheb Bhimrao Ambedkar University** (**A Central University**), **Lucknow** hereby declare that the summer internship project report entitled **Machine Learning And Quantum Computing** is an original work and the same has not been submitted to any other institute for the award or any other degree.

Date :

Place :

ANUJ KUMAR KASHYAP

Roll No. - 218248

Computer Science And Engineering

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Anna University, Chennai, for providing me with the opportunity to undertake this summer internship. This experience has been valuable in enhancing my knowledge and skills in the fields of machine learning and quantum computing.

I am particularly grateful to Dr. Ganesh Kumar Pugalendhi for his exceptional guidance, encouragement, and support throughout the duration of this internship. His expertise and insights have significantly contributed to the success of my project titled "Machine Learning and Quantum Computing." Dr. Pugalendhi's mentorship has been instrumental in my understanding of the complex concepts involved in machine learning and quantum computing.

I would also like to extend my thanks to the faculty and staff at Anna University for their assistance and for providing a conducive environment for research. The facilities and resources available at the university have greatly facilitated the progress of my project.

Finally, I would like to acknowledge the support of my family and friends, whose encouragement and understanding have been crucial throughout this journey.

Thank you all for making this internship a truly rewarding experience.

ANUJ KUMAR KASYAHP

PREFACE

In the 30 days of the summer internship program, I studied various python libraries before focusing on Machine Learning with Python due to its ease of management, object-oriented nature, and the availability of robust debugging tools. We then progressed to Machine Learning concepts with Python. Machine Learning, a field of Artificial Intelligence, uses statistical techniques to enable computer systems to learn from given data sets without being explicitly programmed. During the training, I delved into Neural Networks, Support Vector Machines (SVM), and hyperparameter tuning to optimize our models. I also explored the depth of quantum computing and its potential applications in Machine Learning, Deep learning, and Cryptography. After 30 days of training, I was capable of developing sophisticated models in Python.

Keywords: Python, Machine Learning, Neural Network, Support Vector Machine (SVM), Hyperparameter Tuning, Quantum Computing.

ABSTRACT

This report presents the outcomes of a 30-day summer internship program at Anna University, focusing on Python programming, machine learning, and quantum computing. The primary objectives were to gain proficiency in Python, understand machine learning concepts, and explore advanced techniques such as Support Vector Machines (SVM), Neural Networks, hyperparameter tuning, and explore quantum computing.

The methodology involved an initial phase of learning Python fundamentals, followed by hands-on exercises in developing basic programs. The training then progressed to machine learning, where I studied and implemented Support Vector Machine (SVM) and Neural Networks Models. I emphasized hyperparameter tuning to optimize model performance, utilizing techniques such as GridSearch to experiment with various parameters. Additionally, we explored the principles of quantum computing and its potential to enhance machine learning applications.

Key results from the training include the development of sophisticated machine learning models capable of performing tasks such as movement prediction. By applying hyperparameter tuning, we achieved significant improvements in model accuracy and efficiency. The exploration of quantum computing provided insights into its potential applications in accelerating machine learning processes and handling complex computations more efficiently than classical methods.

The conclusions drawn from this training are multifaceted. Firstly, Python, with its extensive libraries and tools, proved to be an effective language for implementing machine learning algorithms and conducting hyperparameter tuning. Secondly, Support Vector Machine (SVM) and Neural Networks Models demonstrated their efficacy in classification tasks, with hyperparameter tuning playing a crucial role in enhancing their performance. Lastly, Quantum computing emerged as a promising field that, despite its current nascent state, holds the potential to revolutionize machine learning and computational efficiency in the future.

This comprehensive training not only equipped us with practical skills in Python and machine learning but also broadened our understanding of emerging technologies like quantum computing, preparing us for advanced research and applications in these fields.

TABLE OF CONTENT

Item No.	Particular	Page No.
1	**Introduction**	**8 - 12**
	1.1 Background	8
	1.2 Problem Statement	9
	1.3 Objectives	11
2	**Literature Review**	**13 - 19**
	2.1 Introduction to BCI and MI EEG	13
	2.2 Previous Studies on MI EEG Classification	14
	2.3 Current Challenges and Research Gaps	17
3	**Methodology**	**20 - 29**
	3.1 Dataset 3.1.1 Source and Description of BNCI Horizon 2020 Database 3.1.2 Participants and Tasks	20
	3.2 Signal Processing	21
	3.3 Model Development 3.3.1 Support Vector Machine (SVM) 3.3.2 Artificial Neural Network (ANN)	21
	3.4 Implementation Details 3.4.1 Tools, Libraries, and Environment Setup	25
4	**Experimental Setup**	**30 - 35**
	4.1 Data Splitting	30
	4.2 Environment	30
	4. 3 Detailed Implementation Process	31
5	**Hyperparameter Tuning**	**36 - 39**
	5.1 SVM Hyperparameter Tuning 5.1.1 Parameter Grid 5.1.2 GridSearchCV Setup	36

	5.1.3 Results and Best Parameters	
	5.2 ANN Hyperparameter Tuning 5.1.1 Parameter Grid 5.2.2 GridSearch Setup 5.2.3 Results and Best Parameters	37
	5.3 Log Results from Hyperparameter Tuning	40
6	**Results and Discussion**	**47 - 51**
	6.1 Performance Metrics	48
	6.2 Comparative Analysis of SVM and ANN 6.2.1 Before Tuning 6.2.2 After Tuning	50
	6.3 Insights and Interpretations	51
7	**Applications**	**52 - 53**
	7.1 Potential Real-World Applications 7.1.1 Wheelchair Control 7.1.2 Exoskeletons 7.1.3 Robotic Arms 7.1.4 Home Appliances	53
8	**Personal Experience and Learning**	**54 - 56**
	8.1 Challenges Faced 8.1.1 Technical Challenges 8.1.2 Non-Technical Challenges	54
	8.2 Learning Outcomes	54
	8.3 Future Directions	55
9	**Conclusion**	**57 - 58**
	9.1 Summary of Key Findings	57
	9.2 Implications of Results	57
	9.3 Suggestions for Future Work	58
10	**References**	59

Introduction

1.1 Background

Brain-Computer Interfaces (BCIs) represent a groundbreaking technology that establishes a direct communication pathway between the human brain and external devices. This technology holds immense potential, particularly for individuals with severe motor impairments, as it enables control over various devices without requiring physical movement. BCIs translate neural activity into actionable commands, allowing users to interact with their environment in ways that were previously impossible.

Importance of BCIs

BCIs have garnered significant attention due to their ability to improve the quality of life for individuals with disabilities. These interfaces can restore lost motor functions, enhance communication capabilities, and increase independence. For example, BCIs can be used to control prosthetic limbs, operate wheelchairs, or interact with computer systems, thereby providing users with greater autonomy and enhancing their daily lives.

MI EEG Signals

Motor Imagery (MI) Electroencephalography (EEG) signals are a critical component of BCIs. MI involves the mental simulation of movement without any actual muscle activity. When a person imagines moving a specific part of their body, distinct patterns of brain activity are generated. These patterns can be recorded using EEG, a non-invasive method that measures electrical activity along the scalp.

MI EEG signals are particularly advantageous for BCI applications due to their practical and non-invasive nature. These signals offer valuable insights into the user's intent, enabling the BCI system to interpret and execute commands based on imagined movements. The ability to decode MI EEG signals accurately is essential for developing effective BCI systems.

Applications of MI EEG Signals

The applications of MI EEG signals in BCI systems are vast and varied. Some prominent applications include:

- **Assistive Devices**: MI EEG signals can be used to control assistive devices such as wheelchairs, robotic arms, and prosthetic limbs, providing individuals with enhanced mobility and independence.
- **Communication Systems**: BCIs can facilitate communication for individuals with severe speech or motor impairments by translating MI EEG signals into text or speech.
- **Neurorehabilitation**: BCIs utilizing MI EEG signals can aid in neurorehabilitation by promoting motor recovery in stroke or spinal cord injury patients through targeted brain training.
- **Smart Home Control**: BCIs can enable users to control home appliances and smart devices using their thoughts, increasing convenience and accessibility.
- **Gaming and Virtual Reality**: MI EEG signals can enhance gaming experiences and virtual reality interactions by providing a more immersive and intuitive control mechanism.

In summary, the importance of BCIs lies in their transformative potential to improve the lives of individuals with disabilities. MI EEG signals serve as a crucial foundation for developing these interfaces, offering a non-invasive and practical means of decoding user intent. The wide range of applications underscores the versatility and impact of BCIs in various domains, from healthcare to daily living.

1.2 Problem Statement

The Need for Accurate Classification of MI EEG Signals

In the realm of Brain-Computer Interfaces (BCIs), the accurate classification of Motor Imagery (MI) Electroencephalography (EEG) signals is paramount. MI EEG signals, which are generated when a person imagines moving specific parts of their body, hold the key to enabling direct communication between the brain and external devices. However, the inherent complexity and variability of these signals pose significant challenges to their classification and interpretation.

Challenges in MI EEG Signal Classification

- **Noise and Artifacts**: MI EEG signals are often contaminated with noise and artifacts from various sources, such as muscle movements, eye blinks, and environmental interference. These extraneous signals can obscure the true MI patterns, making it difficult to accurately classify the user's intent.
- **Inter-subject Variability**: There is considerable variability in MI EEG signals between different individuals. Factors such as brain anatomy, signal strength, and cognitive strategies can vary widely, necessitating personalized models that can adapt to each user's unique signal characteristics.
- **Intra-subject Variability**: Even within the same individual, MI EEG signals can vary across different sessions due to changes in mental state, fatigue, or attention levels. This variability adds another layer of complexity to the classification process.
- **Signal Dimensionality**: MI EEG signals are high-dimensional, with multiple channels capturing data at high sampling rates. This high dimensionality can lead to issues such as overfitting in machine learning models, where the model performs well on training data but fails to generalize to new, unseen data.
- **Feature Extraction**: Extracting meaningful features from raw MI EEG signals is a critical step in the classification process. The choice of feature extraction methods, such as time-domain, frequency-domain, or time-frequency domain techniques, can significantly impact the accuracy of the classification.
- **Model Selection**: Selecting appropriate machine learning models for MI EEG signal classification is challenging due to the diverse nature of the signals and the specific requirements of the application. Commonly used models include Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), and Artificial Neural Networks (ANN), each with their own strengths and weaknesses.

Importance of Accurate Classification

The ability to accurately classify MI EEG signals is crucial for the development of effective BCI systems. High classification accuracy ensures that the system can reliably interpret the user's intentions and translate them into precise control commands. This reliability is essential for applications such as :

- **Assistive Technologies**: For individuals relying on BCIs to control assistive devices like wheelchairs or robotic arms, accurate classification is critical for safe and effective

operation.

- **Communication Aids**: Accurate classification enables seamless communication for individuals with severe speech or motor impairments, allowing them to convey their thoughts and needs more effectively.
- **Neurorehabilitation**: In rehabilitation settings, accurate MI EEG classification can facilitate targeted brain training, promoting motor recovery and improving patient outcomes.
- **Smart Home Integration**: For smart home control, accurate classification ensures that commands are executed correctly, enhancing convenience and accessibility for users.

In conclusion, the need for accurate classification of MI EEG signals is driven by the desire to develop robust and reliable BCI systems that can transform the lives of individuals with disabilities. Overcoming the challenges associated with MI EEG signal classification will pave the way for more advanced and user-friendly BCIs, broadening their application and impact.

1.3 Objectives

The primary objectives of this study are centered around improving the classification accuracy of Motor Imagery (MI) Electroencephalography (EEG) signals for Brain-Computer Interface (BCI) applications. This involves the development and optimization of machine learning models tailored to handle the unique challenges posed by MI EEG signals. The specific goals are as follows:

1. Development of Machine Learning Models

- **Support Vector Machine (SVM) Model**: Develop an SVM model to classify MI EEG signals into distinct classes. SVM is chosen for its effectiveness in binary classification and its ability to handle high-dimensional data.
- **Artificial Neural Network (ANN) Model**: Develop an ANN model to classify MI EEG signals. ANNs are selected for their flexibility and capability to model complex patterns in the data.

2. Hyperparameter Tuning

- **GridSearchCV for SVM**: Utilize GridSearchCV to systematically explore and identify the optimal hyperparameters for the SVM model. Parameters to be tuned include the regularization parameter (C), kernel type, and gamma value.
- **GridSearch for ANN**: Conduct a comprehensive hyperparameter search for the ANN model, focusing on parameters such as learning rate, activation function, number of

epochs, batch size, and network architecture (number of layers and neurons).

3. Model Evaluation

- **Training and Testing Split**: Use an 90/10 split for training and testing data to ensure robust model evaluation and avoid overfitting.
- **Performance Metrics**: Evaluate the performance of both models using metrics such as accuracy, precision, recall, F1-score, and confusion matrix. These metrics will provide a comprehensive understanding of the models' classification capabilities.

4. Comparative Analysis

- **Pre-tuning vs. Post-tuning Performance**: Compare the performance of the SVM and ANN models before and after hyperparameter tuning to assess the impact of the tuning process.
- **SVM vs. ANN Performance**: Conduct a comparative analysis between the SVM and ANN models to determine which model performs better for MI EEG signal classification and under what conditions.

7. Real-World Applications

- **Control Commands for Devices**: Explore the potential use of the classified MI EEG signals as control commands for external devices such as wheelchairs, exoskeletons, robotic arms, and smart home appliances.
- **Feasibility and Practicality**: Assess the feasibility and practicality of implementing the developed models in real-world BCI systems.

8. Documentation and Reporting

- **Detailed Report**: Document the entire research process, including methodology, results, discussions, and insights gained. The report will serve as a comprehensive guide for future research and development in the field of MI EEG signal classification for BCI applications.

In summary, this study aims to develop and optimize machine learning models for MI EEG signal classification, enhance their performance through systematic hyperparameter tuning, and evaluate their potential for real-world BCI applications. The ultimate goal is to contribute to the advancement of BCI technology and improve the quality of life for individuals relying on these systems.

Literature Review

2.1 Introduction to BCI and MI EEG: Definition and significance.

Brain-Computer Interface (BCI)

Definition : A Brain-Computer Interface (BCI) is a technology that facilitates direct communication between the brain and an external device. It translates neural activity into commands that can control software or hardware, bypassing traditional neuromuscular pathways. This interface leverages brain signals to interact with the environment, enabling new forms of control for individuals with severe motor impairments.

Significance : BCIs hold transformative potential in various fields, particularly in healthcare, rehabilitation, and assistive technologies. They offer new communication avenues for individuals with disabilities, provide innovative tools for neurorehabilitation, and open up novel interactive experiences in gaming and virtual reality. Key areas of impact include :

- **Assistive Technologies:** BCIs can restore movement and communication abilities for people with conditions such as amyotrophic lateral sclerosis (ALS), spinal cord injuries, and severe strokes. By translating neural signals into commands, BCIs allow users to control wheelchairs, robotic arms, and computer interfaces, significantly enhancing their independence and quality of life.
- **Neurorehabilitation:** BCIs are used in therapeutic settings to promote motor recovery after neurological injuries. By engaging patients in brain-controlled tasks, these systems can facilitate neuroplasticity and improve functional outcomes.
- **Research and Understanding:** BCIs contribute to neuroscience research by providing insights into brain function and neuroplasticity. They enable detailed studies of neural mechanisms underlying movement, cognition, and sensory processing.
- **Enhanced Interaction:** In gaming and virtual reality, BCIs create more immersive and intuitive user experiences by allowing players to control environments and characters using their thoughts. This technology pushes the boundaries of interactive entertainment.

Motor Imagery Electroencephalography (MI EEG)

Definition : Motor Imagery (MI) Electroencephalography (EEG) involves the non-invasive recording of electrical activity generated by the brain during imagined movements. MI refers to the mental rehearsal of a movement without any actual physical execution, which activates specific neural patterns similar to those observed during real movement. EEG captures these neural signals via electrodes placed on the scalp.

Significance : MI EEG is particularly valuable for BCIs due to its non-invasive nature and the rich information it provides about brain activity. Key aspects of its significance include :

- **Non-Invasiveness:** EEG is a widely used technique that does not require surgical implantation, making it safe and relatively easy to use. This characteristic makes MI EEG a practical choice for developing BCI systems, especially for long-term and widespread applications.
- **Real-Time Feedback:** MI EEG allows for real-time monitoring and interpretation of brain activity, enabling immediate feedback in BCI applications. This real-time capability is crucial for interactive tasks and training protocols in neurorehabilitation.
- **Neuroplasticity and Learning:** Engaging in motor imagery has been shown to promote neuroplasticity, the brain's ability to reorganize itself by forming new neural connections. This property is harnessed in neurorehabilitation to aid recovery by reinforcing neural pathways involved in movement.
- **User-Centric Control:** MI EEG provides a direct link between a user's intentions and external devices. This direct control is essential for assistive technologies, allowing users with motor impairments to interact with their environment and perform daily activities independently.

2.2 Previous Studies on MI EEG Classification

Overview of Notable Studies

Research in the field of Motor Imagery (MI) Electroencephalography (EEG) classification has been extensive, given its critical role in the development of effective Brain-Computer Interfaces (BCIs). Numerous studies have explored various algorithms and techniques to enhance the accuracy and reliability of MI EEG signal classification. This section provides an overview of some notable studies and their contributions to the field.

- **Wolpaw et al.** (**2002**): This foundational study laid the groundwork for non-invasive BCIs using EEG. The researchers demonstrated the feasibility of using EEG signals for real-time control of computer cursors and other devices. They explored linear classifiers and emphasized the importance of adaptive algorithms for improving user performance over time.

- **Pfurtscheller and Neuper** (**2001**): This study focused on the detection of MI-related EEG patterns and the use of these patterns to control external devices. They highlighted the significance of Mu and Beta rhythms in MI tasks and explored methods for enhancing signal-to-noise ratio through spatial filtering techniques such as Common Spatial Patterns (CSP).
- **Blankertz et al.** (**2007**): This work introduced the Berlin Brain-Computer Interface (BBCI) project, which aimed to improve the accuracy of MI EEG classification through advanced signal processing and machine learning techniques. The study demonstrated the effectiveness of Linear Discriminant Analysis (LDA) and Support Vector Machines (SVM) for classifying MI tasks.
- **Ang et al.** (**2012**): This study compared various feature extraction methods and classifiers for MI EEG signal classification. The researchers evaluated techniques such as CSP, Wavelet Transform, and Band Power features combined with classifiers like LDA, SVM, and Artificial Neural Networks (ANN). They provided insights into the strengths and weaknesses of each approach.

Discussion on Various Algorithms

Several algorithms have been explored for MI EEG signal classification, each with its unique advantages and challenges. The following section discusses some of the most commonly used algorithms:

- **Linear Discriminant Analysis (LDA) :**
    - **Description :** LDA is a linear classifier that aims to find a linear combination of features that best separates two or more classes. It assumes that different classes generate data based on Gaussian distributions with the same covariance matrix.
    - **Advantages :** Simple to implement, computationally efficient, and effective for linearly separable data.
    - **Challenges :** Limited performance on non-linearly separable data and sensitivity to noise and outliers.
- **Support Vector Machine (SVM) :**
    - **Description :** SVM is a powerful classification algorithm that seeks to find the hyperplane that maximizes the margin between different classes in the feature space. It can handle both linear and non-linear classification tasks using kernel functions.
    - **Advantages :** High accuracy, effective in high-dimensional spaces, and robust to overfitting with the use of regularization parameters.
    - **Challenges :** Computationally intensive for large datasets, requires careful tuning of hyperparameters, and can be sensitive to the choice of kernel function.
- **Artificial Neural Networks (ANN) :**
    - **Description :** ANNs are computational models inspired by the human brain, consisting of interconnected layers of neurons. They can model complex, non-linear relationships between inputs and outputs, making them suitable for MI EEG signal classification.
    - **Advantages :** High flexibility, ability to learn complex patterns, and potential for high accuracy with sufficient training data.
    - **Challenges :** Requires large amounts of data for effective training, prone to overfitting, and computationally expensive.

Comparative Analysis

Comparative studies have highlighted the relative strengths and weaknesses of different algorithms for MI EEG classification:

- **LDA vs. SVM :** SVM generally outperforms LDA in terms of accuracy, especially for non-linearly separable data. However, LDA remains a popular choice due to its simplicity and computational efficiency.
- **SVM vs. ANN :** While SVMs are effective for smaller datasets and can achieve high accuracy, ANNs excel with larger datasets and more complex patterns. However, ANNs require more computational resources and careful tuning to avoid overfitting.

In conclusion, the field of MI EEG classification for BCIs has seen significant advancements through the application of various algorithms and techniques. Each method offers unique advantages and trade-offs, and the choice of algorithm often depends on the specific requirements of the application and the characteristics of the EEG data. Future research continues to explore innovative combinations and enhancements to further improve classification performance and expand the practical applications of BCIs.

2.3 Current Challenges and Research Gaps

Challenges in MI EEG Classification

- **Signal Variability**:
    - **Inter-Subject Variability**: Significant differences in EEG signals between individuals due to anatomical and functional brain differences.
    - **Intra-Subject Variability**: Variations in EEG signals within the same individual across different sessions due to factors like mental state, fatigue, and attention.
- **Noise and Artifacts**:
    - EEG signals are susceptible to various types of noise and artifacts, such as muscle movements, eye blinks, and electrical interference, which can obscure relevant MI patterns.
- **High Dimensionality**:
    - EEG data typically involve multiple channels and high sampling rates, resulting in high-dimensional datasets. This dimensionality poses challenges for efficient and effective feature extraction and classification.

- **Model Selection and Optimization**:
    - Choosing appropriate machine learning models and optimizing their hyperparameters are challenging tasks. Different models (e.g., LDA, SVM, ANN) have their own strengths and weaknesses, and their performance can vary depending on the specific dataset and application.
- **Real-Time Processing**:
    - Implementing real-time MI EEG classification for BCI applications requires efficient algorithms and computational resources to ensure timely and accurate responses.

Research Gaps

Limited Generalizability:

- Many studies focus on specific datasets or small sample sizes, limiting the generalizability of their findings. There is a need for studies that validate models across diverse datasets and larger participant groups.

Hyperparameter Tuning:

- While hyperparameter tuning is crucial for optimizing model performance, many studies do not thoroughly explore the impact of different hyperparameter settings. Systematic approaches to hyperparameter tuning, such as GridSearch, are often underutilized.

Comparison of Algorithms:

- Comparative studies that systematically evaluate the performance of different machine learning algorithms on the same dataset are limited. There is a need for comprehensive comparisons that consider various performance metrics and real-world applicability.

Integration of Advanced Techniques:

- The potential of integrating advanced techniques, such as deep learning, transfer learning, and ensemble methods, with traditional machine learning models for MI EEG classification remains underexplored.

Feature Extraction Innovations:

- New methods for feature extraction that can better capture the relevant characteristics of MI EEG signals are needed. This includes exploring novel time-frequency domain techniques and hybrid approaches that combine multiple feature extraction methods.

Justification for the Study

Given the identified challenges and research gaps, this study aims to contribute to the field of MI EEG classification for BCI applications in the following ways:

1. **Development and Optimization of Models**:
    - By developing and optimizing both SVM and ANN models, this study aims to provide a comprehensive comparison of their performance and identify the most effective approaches for MI EEG classification.
2. **Systematic Hyperparameter Tuning**:
    - The study employs GridSearch for systematic hyperparameter tuning, exploring a wide range of parameters to achieve optimal model performance. This addresses the gap related to insufficient hyperparameter exploration in existing studies.
3. **Comprehensive Evaluation**:
    - The study evaluates model performance using multiple metrics, including accuracy, precision, recall, F1-score, and confusion matrix. This thorough evaluation provides a detailed understanding of the strengths and limitations of each model.
4. **Real-World Applicability**:
    - By considering the potential applications of the classified MI EEG signals for controlling assistive devices, the study bridges the gap between theoretical research and practical implementation, contributing to the advancement of real-world BCI systems.

In summary, this study aims to address critical challenges and research gaps in MI EEG classification for BCI applications by developing and optimizing robust models, systematically tuning hyperparameters, and evaluating their performance comprehensively. The ultimate goal is to enhance the accuracy and applicability of MI EEG-based BCIs, contributing to their advancement and practical use in improving the quality of life for individuals with disabilities.

Methodology

3.1 Dataset

3.1.1 Source and Description of the BNCI Horizon 2020 Database

Source of the Dataset

The dataset used in this study is obtained from the BNCI Horizon 2020 database, a well-established repository for Brain-Computer Interface (BCI) research. The BNCI Horizon 2020 project aims to advance the field of BCI by providing high-quality datasets and fostering collaboration among researchers.

Description of the Dataset

The dataset includes electroencephalography (EEG) recordings of motor imagery (MI) tasks from multiple participants. It is designed to facilitate research on MI EEG signal classification, which is crucial for developing effective BCI systems.

- **Participants**:
    - The dataset comprises EEG recordings from nine participants.
    - The participants were selected based on specific inclusion criteria to ensure the quality and reliability of the data.
    - Each participant provided informed consent, and the study was conducted following ethical guidelines.
- **Tasks**:
    - The MI EEG data includes four distinct motor imagery tasks:
        - **Class 1**: Imaginative movement of the left hand.
        - **Class 2**: Imaginative movement of the right hand.
        - **Class 3**: Imaginative movement of both feet.
        - **Class 4**: Imaginative movement of the tongue.
    - Participants were instructed to perform these tasks while their EEG signals were recorded.
- **EEG Recording Setup**:
    - **Channels**: The EEG data were recorded using 22 channels placed on the scalp according to the international 10-20 system.
    - **Sampling Rate**: The signals were sampled at a rate of 250 Hz, ensuring a high temporal resolution for capturing brain activity.

- **Bandpass Filtering**: The raw EEG signals were bandpass filtered with a lower cut-off frequency of 0.5 Hz and an upper cut-off frequency of 100 Hz to remove noise and irrelevant frequencies.

3.2 Signal Processing Steps

To prepare the dataset for classification, several signal processing steps were applied:

1. **Channel Selection**:
   - Using the energy count threshold approach (ECTA), nine dominant channels were identified from the 22 available EEG channels. These channels were selected based on their relevance to the motor imagery tasks.
2. **Feature Extraction**:
   - **Discrete Wavelet Transform** (**DWT**): The EEG signals from the nine dominant channels were decomposed using the discrete wavelet transform with the Daubechies 4 mother wavelet and four-level decomposition.
   - **Mu and Beta Rhythms**: From the decomposed signals, Mu (8-12 Hz) and Beta (12-30 Hz) rhythms were extracted, as these frequency bands are known to be associated with motor imagery.
   - **Energy and Entropy Features**: For each 3-second window from the nine dominant channels, the energy and entropy feature values were calculated from the Mu and Beta rhythms. These features capture the power and complexity of the EEG signals, respectively.
3. **Data Splitting**:
   - The extracted features were divided into training and testing sets.
   - **Training Set** : 90% of the data was used for training the classification models.
   - **Testing Set** : 10% of the data was reserved for testing and evaluating the performance of the models.

Summary

The BNCI Horizon 2020 dataset provides a rich source of MI EEG signals from multiple participants performing various motor imagery tasks. The dataset includes carefully selected EEG channels, bandpass-filtered signals, and features extracted using advanced signal processing techniques. These data were used to train and evaluate machine learning models for MI EEG classification, contributing to the development of effective BCI systems.

3.3 Model Development

3.3.1 Support Vector Machine (SVM)

Initial Setup : Support Vector Machine (SVM) is a powerful classification algorithm that aims to find the optimal hyperplane that separates different classes in the feature space. For this study, the SVM model was developed using the scikit-learn library in Python.

Parameters and Rationale:

- **Kernel Function**:
    - **Type**: Radial Basis Function (RBF) Kernel.
    - **Rationale**: The RBF kernel is chosen because it can handle non-linear relationships between the features and the target classes, making it suitable for complex MI EEG signal patterns.
- **Regularization Parameter (C)**:
    - **Initial Value**: 1.0.
    - **Rationale**: The regularization parameter C controls the trade-off between maximizing the margin and minimizing the classification error. An initial value of 1.0 is chosen as a baseline, with plans to optimize it using hyperparameter tuning.
- **Gamma**:
    - **Initial Value**: 'scale' (1 / (n_features * X.var())).
    - **Rationale**: Gamma defines the influence of a single training example. The 'scale' setting is chosen as it adjusts gamma based on the number of features and the variance of the data, providing a balanced starting point.
- **Class Weight**:
    - **Initial Value**: None.
    - **Rationale**: Initially, no class weight adjustments are applied. If class imbalance is observed, class weights will be adjusted to ensure balanced learning.

Model Training:

- The SVM model is trained using the training set (90% of the data).
- The model's performance is evaluated using the testing set (10% of the data) to ensure generalization.

Rationale:

- SVM is chosen for its robustness in high-dimensional spaces and its ability to handle non-linear relationships, which are common in MI EEG data.
- The parameters are selected to provide a solid baseline, with plans to optimize them through GridSearch to achieve the best performance.

3.3.2 Artificial Neural Network (ANN)

Initial Setup: Artificial Neural Networks (ANNs) are inspired by the human brain and consist of interconnected layers of neurons. For this study, the ANN model is developed using the TensorFlow and Keras libraries in Python.

Architecture and Parameters:

1. **Input Layer**:
    - **Size**: Number of input features (18 features: energy and entropy values from Mu and Beta rhythms of nine dominant channels).
    - **Rationale**: The input layer size matches the number of features extracted from the MI EEG signals.
2. **Hidden Layers**:
    - **Architecture**:
        - **First Hidden Layer**: 64 neurons, activation function: ReLU.
        - **Second Hidden Layer**: 32 neurons, activation function: ReLU.
    - **Rationale**: The hidden layers are designed to capture complex patterns in the data. The ReLU activation function is chosen for its ability to introduce non-linearity and prevent vanishing gradient issues.
3. **Output Layer**:
    - **Size**: 2 neurons (corresponding to the four MI tasks), activation function: Softmax.
    - **Rationale**: The softmax activation function is used in the output layer to produce probability distributions over the four classes, enabling multi-class classification.
4. **Optimizer**:
    - **Type**: Adam.
    - **Rationale**: The Adam optimizer is chosen for its efficiency and ability to handle sparse gradients, which are common in EEG data.

5. **Loss Function**:
    - **Type**: Categorical Crossentropy.
    - **Rationale**: The categorical cross entropy loss function is suitable for multi-class classification problems.
6. **Metrics**:
    - **Type**: Accuracy.
    - **Rationale**: Accuracy is used as the primary metric to evaluate model performance, with plans to monitor additional metrics such as precision, recall, and F1-score.
7. **Batch Size and Epochs**:
    - **Initial Values**: Batch size: 32, Epochs: 100.
    - **Rationale**: The batch size and number of epochs are chosen based on common practices in ANN training, with plans to optimize them through GridSearch.

Model Training:

- The ANN model is trained using the training set (90% of the data) and validated using the testing set (10% of the data).
- Early stopping and model checkpointing are employed to prevent overfitting and ensure the best model is saved.

Rationale:

- ANNs are chosen for their ability to learn complex, non-linear patterns in the data, making them suitable for MI EEG classification.
- The architecture and parameters are selected to provide a strong baseline, with plans to optimize them through GridSearch to achieve the best performance.

3.4 Implementation Details

3.4.1 Tools, Libraries, and Environment Setup

For the development and implementation of the SVM and ANN models, several tools and libraries are utilized. Below is an overview of the tools, libraries, and environment setup required for this project.

Tools and Libraries:

1. **Python**: The primary programming language used for implementing the models.
2. **Google Colab** : An interactive environment for writing and running code, ideal for data analysis and visualization.
3. **NumPy**: A library for numerical computations, used for handling arrays and mathematical operations.
4. **Pandas**: A data manipulation library, used for data loading, preprocessing, and handling datasets.
5. **scikit-learn**: A machine learning library, used for implementing the SVM model and hyperparameter tuning.
6. **Matplotlib and Seaborn**: Visualization libraries, used for plotting data and model performance metrics.

Environment Setup :

1. **Python Installation:** Ensure that Python is installed on your system. You can download it from the official website.
2. **Virtual Environment:** Create a virtual environment to manage dependencies. This can be done using 'venv' or 'conda'.
    - `python -m venv bci_env`
    - `source bci_env/bin/activate  #On Windows: bci_env\Scripts\activate`
3. **Library Installation**: Install the necessary libraries using 'pip'.
    - `pip install numpy pandas scikit-learn tensorflow keras matplotlib seaborn jupyter`

Code Snippets for Key Steps

1. Data Loading and Preprocessing :

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


# Load the dataset

data = pd.read_csv('mi_eeg_data.csv')


# Split the data into features and labels

X = data.drop('label', axis=1)

y = data['label']


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=0)


# Standardize the features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

2. SVM Model Implementation :

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix


# Initialize the SVM model
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale')


# Train the model
svm_model.fit(X_train, y_train)


# Predict on the test set
y_pred = svm_model.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'SVM Model Accuracy: {accuracy}')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

3. ANN Model Implementation :

```
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix



# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
test_size=0.1, random_state=0)



# Initialize the ANN model with similar parameters as your TensorFlow
model

clf = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu',
solver='adam', learning_rate_init=0.001, max_iter=100)



# Train the model

clf.fit(X_train, y_train)



# Predict on the test set

y_pred = clf.predict(X_test)



# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f'ANN Model Accuracy: {accuracy}')

print(classification_report(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))
```

4. Visualizing Model Performance :

```
import matplotlib.pyplot as plt

import seaborn as sns


# Plot training and validation accuracy for ANN

plt.figure(figsize=(12, 6))

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.title('ANN Training and Validation Accuracy')

plt.show()


# Plot confusion matrix for SVM

plt.figure(figsize=(8, 6))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('SVM Confusion Matrix')

plt.show()
```

These code snippets provide a comprehensive guide to implementing and evaluating the SVM and ANN models for MI EEG classification, including data preprocessing, model training, hyperparameter tuning, and performance visualization. The use of Jupyter Notebook allows for an interactive and iterative development process, facilitating thorough analysis and experimentation.

Experimental Setup

4.1 Data Splitting : Training and Testing Split

To ensure that the models are trained effectively and their performance is evaluated accurately, the dataset is divided into training and testing sets. An 80/20 split is used for this purpose:

- **Training Set**: 90% of the data
- **Testing Set**: 10% of the data

This split ensures that there is enough data for both training the models and evaluating their performance.

Code for Data Splitting

```
from sklearn.model_selection import train_test_split



# Assuming 'data' is a DataFrame containing the MI EEG data with features and labels

X = data.drop('label', axis=1)

y = data['label']



# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
```

4.2 Environment : Hardware and Software Used

Hardware

- Processor : Apple M1
- RAM : 8 GB
- Storage : 256 GB SSD

Software

- Operating System : macOS (Sonoma 14.5)
- Python : Version 3.12
- Libraries :
    - numpy : 1.19.2
    - pandas : 1.1.3
    - Scikit-learn : 0.23.2
    - matplotlib : 3.3.2
    - Seaborn : 0.11.0

4.3 Implementation : Detailed Description of the Implementation Process

SVM Model Implementation

1. Importing Libraries :

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn import svm

from sklearn import metrics

from sklearn.metrics import confusion_matrix
```

The necessary libraries are imported at the beginning. These include '`pandas`' for data manipulation, '`numpy`' for numerical operations, and several modules from '`scikit-learn`' for machine learning tasks and metrics.

2. Loading the Dataset :

```
dataset = pd.read_excel('subject2.xlsx', engine='openpyxl')  # Use openpyxl
engine for Excel files
```

The dataset is loaded from an Excel file named 'subject2.xlsx'. The 'openpyxl' engine is specified to handle the Excel file format.

3. Exploratory Data Analysis (EDA) :

```
dataset.shape
```

```
dataset.describe()
```

- dataset.shape: This command checks the dimensions of the dataset, providing the number of rows and columns.
- dataset.describe() : This provides a statistical summary of the dataset, including count, mean, standard deviation, minimum, maximum, and quartiles for each feature.

4. Preparing the Data :

```
x = dataset.iloc[:, 0:35]
```

```
y = dataset.iloc[:, 36]
```

The features (x) and target labels (y) are extracted from the dataset. Here, columns 0 to 35 are assumed to be the features, and column 36 is the target label.

5. Splitting the Data :

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=0)
```

The dataset is split into training and testing sets using a 90-10 split ratio (test_size=0.1). A random_state of 0 ensures the split is reproducible.

6. Training the SVM Model :

```
support = svm.SVC(kernel='linear')
```

```
model = support.fit(x_train, y_train)
```

An SVM model with a linear kernel is instantiated and trained using the training data.

7. Model Insights :

```
print("number of support vectors for each class: ", model.n_support_)
```

```
print("indices of support vectors: ", model.support_)

print("support vectors: ", model.support_vectors_)
```

The trained model provides insights into the support vectors:

- model.n_support_: Number of support vectors for each class.
- model.support_: Indices of the support vectors.
- model.support_vectors_: The actual support vectors.

8. Making Predictions :

```
y_pred = model.predict(x_test)
```

The trained model makes predictions on the test data.

9. Evaluating the Model :

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred.flatten()})

print(df)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))

print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

print("Precision:", metrics.precision_score(y_test, y_pred))

print("Recall:", metrics.recall_score(y_test, y_pred))
```

The performance of the model is evaluated using several metrics:

- **Mean Absolute Error** (**MAE**): Measures the average magnitude of errors in a set of predictions, without considering their direction.
- **Mean Squared Error** (**MSE**): Measures the average squared difference between the estimated values and the actual value.
- **Root Mean Squared Error** (**RMSE**): The square root of the mean squared error.
- **Accuracy**: The ratio of correctly predicted observation to the total observations.
- **Precision**: The ratio of correctly predicted positive observations to the total predicted

positives.

- **Recall**: The ratio of correctly predicted positive observations to the all observations in actual class.

10. Confusion Matrix:

```
print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred))
```

A confusion matrix is generated to visualize the performance of the model. It shows the number of true positives, true negatives, false positives, and false negatives.

11. Full Code

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn import svm

from sklearn import metrics

from sklearn.metrics import confusion_matrix



# Try reading the Excel file with a different encoding

dataset = pd.read_excel('subject2.xlsx', engine='openpyxl') # Use openpyxl
engine for Excel files



dataset.shape

dataset.describe()

x=dataset.iloc[:,0:35]

y=dataset.iloc[:,36]



x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
random_state=0)
```

```
support = svm.SVC(kernel='linear')

model=support.fit(x_train, y_train)

print("number of support vectors for each class: ", model.n_support_)

print("indices of support vectors: ", model.support_)

print("support vectors: ", model.support_vectors_)

y_pred = model.predict(x_test)

df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred.flatten()})

print(df)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

print("Precision:",metrics.precision_score(y_test, y_pred))

print("Recall:",metrics.recall_score(y_test, y_pred))

confusion_matrix(y_test, y_pred)
```

This detailed description outlines each step of the implementation process, providing clarity on how the dataset is handled, how the model is trained and evaluated, and how the results are analyzed. This will form an important part of your Methodology chapter, demonstrating a thorough and methodical approach to your research.

Hyperparameter Tuning

5.1 SVM Hyperparameter Tuning

5.1.1 Parameter Grid : Define the Grid for Parameters

To optimize the performance of the Support Vector Machine (SVM), we tuned the following parameters:

- C: Regularization parameter, controls the trade-off between achieving a low training error and a low testing error.
- Gamma: Kernel coefficient for ‘rbf’ and ‘poly’ kernels, controls the shape of the decision boundary.
- Kernel: Specifies the kernel type to be used in the algorithm (e.g., 'linear', 'poly', 'rbf', 'sigmoid').

Parameter Grid :

```
# Hyperparameter tuning using GridSearchCV

param_grid_svm = {'C': [0.1, 1, 10, 100, 1000],

              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],

              'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}
```

5.1.2 GridSearchCV Setup : Explain the Setup and Cross-Validation Process

To find the optimal combination of parameters, we used GridSearchCV, which exhaustively searches through the specified parameter grid using cross-validation.

```
from sklearn.model_selection import GridSearchCV

from sklearn.svm import SVC

# Initialize GridSearchCV with the SVM model and parameter grid

Grid_search_svm = GridSearchCV(SVC(), param_grid_svm, cv=5,
scoring='accuracy', verbose=2, n_jobs=-1)

# Fit the GridSearch on the training data

grid_search_svm.fit(X_train, y_train)
```

5.1.3 Results and Best Parameters :

Present the Performance Metrics for Each Combination

The performance of each parameter combination is evaluated using cross-validation, and the results are stored in the GridSearchCV object.

```
# Extract results

results_svm = pd.DataFrame(grid_search_svm.cv_results_)

results_svm = results_svm[['param_C', 'param_gamma', 'param_kernel',
'mean_test_score']]

# Display the results

print(results_svm)
```

Best Parameters : Highlight the Best-Performing Parameters

The best parameters are selected based on the highest cross-validated accuracy.

```
best_params_svm = grid_search_svm.best_params_

best_score_svm = grid_search_svm.best_score_

print(f'Best Parameters for SVM: {best_params_svm}')

print(f'Best Cross-Validated Accuracy: {best_score_svm}')
```

5.2 ANN Hyperparameter Tuning

5.2.1 Parameter Grid : Define the Grid for Parameters

To optimize the performance of the Artificial Neural Network (ANN), we tuned the following parameters:

- **Learning Rate**: Step size for updating the weights.
- **Activation Function**: Non-linear activation function (e.g., 'relu', 'tanh').
- **Epochs**: Number of passes through the entire training dataset.
- **Batch Size**: Number of samples per gradient update.

Parameter Grid :

```
# Hyperparameter tuning using GridSearchCV

param_grid_ann = {

   'hidden_layer_sizes': [(100,), (100, 50), (100, 50, 25), (50, 25)],

   'activation': ['relu', 'tanh', 'logistic', 'sigmoid'],

   'solver': ['adam', 'sgd'],

   'learning_rate': ['constant', 'adaptive'],

   'max_iter': [100, 200, 500],

   'batch_size': [16, 32, 64]
```

5.2.2 GridSearch Setup: Explain the Setup and Cross-Validation Process

To find the optimal combination of parameters, we used a KerasClassifier wrapped with GridSearchCV.

```
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from sklearn.model_selection import GridSearchCV

# Define the model-building function for KerasClassifier

def create_ann(learning_rate=0.001, activation='relu'):

   model = Sequential()

   model.add(Dense(64, input_dim=X_train.shape[1], activation=activation))

   model.add(Dense(32, activation=activation))

   model.add(Dense(4, activation='softmax'))

   model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

   return model

# Wrap the model using KerasClassifier

ann_model = KerasClassifier(build_fn=create_ann, verbose=0)
```

```
# Initialize GridSearchCV

grid_search_ann = GridSearchCV(estimator=ann_model,
param_grid=param_grid_ann, cv=3, verbose=2, n_jobs=-1)

# Fit the GridSearch on the training data

grid_search_ann.fit(X_train, y_train)
```

5.2.3 Results and Best Parameters :

Present the Performance Metrics for Each Combination

The performance of each parameter combination is evaluated using cross-validation, and the results are stored in the GridSearchCV object.

```
# Extract results

results_ann = pd.DataFrame(grid_search_ann.cv_results_)

results_ann = results_ann[['param_learning_rate', 'param_activation',
'param_epochs', 'param_batch_size', 'mean_test_score']]

# Display the results

print(results_ann)
```

Best Parameters : Highlight the Best-Performing Parameters

The best parameters are selected based on the highest cross-validated accuracy.

```
best_params_ann = grid_search_ann.best_params_

best_score_ann = grid_search_ann.best_score_

print(f'Best Parameters for ANN: {best_params_ann}')

print(f'Best Cross-Validated Accuracy: {best_score_ann}')
```

5.3 Log Results from Hyperparameter Tuning

Log Results : Include Logs from Every Possible Combination of Hyperparameters

To document the detailed logs from each combination of hyperparameters, we include tables and graphs showing the performance metrics for different combinations.

Table for SVM Results :

param_C	param_gamma	param_kernel	mean_test_score	rank_test_score
0.1	1	linear	0.6138763198	21
0.1	1	poly	0.5600301659	48
0.1	1	rbf	0.5173453997	68
0.1	1	sigmoid	0.5173453997	68
0.1	0.1	linear	0.6138763198	21
0.1	0.1	poly	0.5600301659	48
0.1	0.1	rbf	0.5173453997	68
0.1	0.1	sigmoid	0.5173453997	68
0.1	0.01	linear	0.6138763198	21
0.1	0.01	poly	0.5564102564	62
0.1	0.01	rbf	0.5173453997	68
0.1	0.01	sigmoid	0.5173453997	68
0.1	0.001	linear	0.6138763198	21
0.1	0.001	poly	0.5752639517	43
0.1	0.001	rbf	0.5019607843	92
0.1	0.001	sigmoid	0.5173453997	68
0.1	0.0001	linear	0.6138763198	21
0.1	0.0001	poly	0.5173453997	68
0.1	0.0001	rbf	0.5173453997	68
0.1	0.0001	sigmoid	0.5173453997	68
1	1	linear	0.6448717949	2
1	1	poly	0.5600301659	48
1	1	rbf	0.5598039216	61
1	1	sigmoid	0.5173453997	68
1	0.1	linear	0.6448717949	2
1	0.1	poly	0.5600301659	48
1	0.1	rbf	0.6101055807	26
1	0.1	sigmoid	0.5173453997	68
1	0.01	linear	0.6448717949	2
1	0.01	poly	0.5406485671	64
1	0.01	rbf	0.5905731523	38

param_C	param_gamma	param_kernel	mean_test_score	rank_test_score
0.1	1	linear	0.6138763198	21
1	0.01	sigmoid	0.5173453997	68
1	0.001	linear	0.6448717949	2
1	0.001	poly	0.6143288084	19
1	0.001	rbf	0.5828808446	39
1	0.001	sigmoid	0.509653092	90
1	0.0001	linear	0.6448717949	2
1	0.0001	poly	0.5019607843	92
1	0.0001	rbf	0.4671191554	100
1	0.0001	sigmoid	0.5174962293	67
10	1	linear	0.6217948718	7
10	1	poly	0.5600301659	48
10	1	rbf	0.5714177979	45
10	1	sigmoid	0.5173453997	68
10	0.1	linear	0.6217948718	7
10	0.1	poly	0.5600301659	48
10	0.1	rbf	0.6061085973	27
10	0.1	sigmoid	0.5173453997	68
10	0.01	linear	0.6217948718	7
10	0.01	poly	0.5600301659	48
10	0.01	rbf	0.5828054299	40
10	0.01	sigmoid	0.5173453997	68
10	0.001	linear	0.6217948718	7
10	0.001	poly	0.6029411765	30
10	0.001	rbf	0.6019607843	36
10	0.001	sigmoid	0.5021870287	91
10	0.0001	linear	0.6217948718	7
10	0.0001	poly	0.4980392157	94
10	0.0001	rbf	0.5365007541	66
10	0.0001	sigmoid	0.4905731523	95
100	1	linear	0.6026395173	31
100	1	poly	0.5600301659	48
100	1	rbf	0.5714177979	45
100	1	sigmoid	0.5173453997	68
100	0.1	linear	0.6026395173	31
100	0.1	poly	0.5600301659	48
100	0.1	rbf	0.6061085973	27
100	0.1	sigmoid	0.5173453997	68

param_C	param_gamma	param_kernel	mean_test_score	rank_test_score
0.1	1	linear	0.6138763198	21
100	0.01	linear	0.6026395173	31
100	0.01	poly	0.5600301659	48
100	0.01	rbf	0.578959276	41
100	0.01	sigmoid	0.5173453997	68
100	0.001	linear	0.6026395173	31
100	0.001	poly	0.5564102564	62
100	0.001	rbf	0.621719457	12
100	0.001	sigmoid	0.4791101056	96
100	0.0001	linear	0.6026395173	31
100	0.0001	poly	0.5752639517	43
100	0.0001	rbf	0.6177978884	18
100	0.0001	sigmoid	0.4751131222	98
1000	1	linear	0.6178733032	13
1000	1	poly	0.5600301659	48
1000	1	rbf	0.5714177979	45
1000	1	sigmoid	0.5173453997	68
1000	0.1	linear	0.6178733032	13
1000	0.1	poly	0.5600301659	48
1000	0.1	rbf	0.6061085973	27
1000	0.1	sigmoid	0.5173453997	68
1000	0.01	linear	0.6178733032	13
1000	0.01	poly	0.5600301659	48
1000	0.01	rbf	0.578959276	41
1000	0.01	sigmoid	0.5173453997	68
1000	0.001	linear	0.6178733032	13
1000	0.001	poly	0.5367269985	65
1000	0.001	rbf	0.5906485671	37
1000	0.001	sigmoid	0.4791101056	96
1000	0.0001	linear	0.6178733032	13
1000	0.0001	poly	0.6143288084	19
1000	0.0001	rbf	0.679638009	1
1000	0.0001	sigmoid	0.4751131222	98

Table for ANN Results :

param_activation	param_batch_size	param_hidden_layer_sizes	param_learning_rate	param_max_iter	param_solver	mean_test_score	rank_test_score
relu	16	(100,)	constant	100	adam	0.5636639045	302
relu	16	(100,)	constant	100	sgd	0.5327898066	362
relu	16	(100,)	constant	200	adam	0.5636639045	302
relu	16	(100,)	constant	200	sgd	0.5327898066	362
relu	16	(100,)	constant	500	adam	0.5636639045	302
relu	16	(100,)	constant	500	sgd	0.5327898066	362
relu	16	(100,)	adaptive	100	adam	0.5636639045	302
relu	16	(100,)	adaptive	100	sgd	0.6178383676	36
relu	16	(100,)	adaptive	200	adam	0.5636639045	302
relu	16	(100,)	adaptive	200	sgd	0.5985476254	126
relu	16	(100,)	adaptive	500	adam	0.5636639045	302
relu	16	(100,)	adaptive	500	sgd	0.5985476254	126
relu	16	(100, 50)	constant	100	adam	0.5830437494	218
relu	16	(100, 50)	constant	100	sgd	0.5752027087	256
relu	16	(100, 50)	constant	200	adam	0.5868751671	203
relu	16	(100, 50)	constant	200	sgd	0.5675398735	292
relu	16	(100, 50)	constant	500	adam	0.5868751671	203
relu	16	(100, 50)	constant	500	sgd	0.5675398735	292
relu	16	(100, 50)	adaptive	100	adam	0.5830437494	218
relu	16	(100, 50)	adaptive	100	sgd	0.5907065847	181
relu	16	(100, 50)	adaptive	200	adam	0.5868751671	203
relu	16	(100, 50)	adaptive	200	sgd	0.5984139713	144
relu	16	(100, 50)	adaptive	500	adam	0.5868751671	203
relu	16	(100, 50)	adaptive	500	sgd	0.5790786777	235
relu	16	(100, 50, 25)	constant	100	adam	0.602379043	100
relu	16	(100, 50, 25)	constant	100	sgd	0.6216697853	17
relu	16	(100, 50, 25)	constant	200	adam	0.602379043	100
relu	16	(100, 50, 25)	constant	200	sgd	0.6255457543	11
relu	16	(100, 50, 25)	constant	500	adam	0.602379043	100
relu	16	(100, 50, 25)	constant	500	sgd	0.6255457543	11
relu	16	(100, 50, 25)	adaptive	100	adam	0.602379043	100
relu	16	(100, 50, 25)	adaptive	100	sgd	0.5908847902	167
relu	16	(100, 50, 25)	adaptive	200	adam	0.602379043	100
relu	16	(100, 50, 25)	adaptive	200	sgd	0.5907511361	176

relu	16	(100, 50, 25)	adaptive	500	adam	0.602379043	100
relu	16	(100, 50, 25)	adaptive	500	sgd	0.5906620333	183
relu	16	(50, 25)	constant	100	adam	0.5753363628	244
relu	16	(50, 25)	constant	100	sgd	0.5715494966	273
relu	16	(50, 25)	constant	200	adam	0.5753363628	244
relu	16	(50, 25)	constant	200	sgd	0.5715494966	273
relu	16	(50, 25)	constant	500	adam	0.5753363628	244
relu	16	(50, 25)	constant	500	sgd	0.5715494966	273
relu	16	(50, 25)	adaptive	100	adam	0.5753363628	244
relu	16	(50, 25)	adaptive	100	sgd	0.5907511361	176
relu	16	(50, 25)	adaptive	200	adam	0.5753363628	244
relu	16	(50, 25)	adaptive	200	sgd	0.6062995634	79
relu	16	(50, 25)	adaptive	500	adam	0.5753363628	244
relu	16	(50, 25)	adaptive	500	sgd	0.6179274704	29
relu	32	(100,)	constant	100	adam	0.5366212243	355
relu	32	(100,)	constant	100	sgd	0.5172859307	421
relu	32	(100,)	constant	200	adam	0.5366212243	355
relu	32	(100,)	constant	200	sgd	0.5172859307	421
relu	32	(100,)	constant	500	adam	0.5366212243	355
relu	32	(100,)	constant	500	sgd	0.5172859307	421
relu	32	(100,)	adaptive	100	adam	0.5366212243	355
relu	32	(100,)	adaptive	100	sgd	0.5946716564	150
relu	32	(100,)	adaptive	200	adam	0.5366212243	355
relu	32	(100,)	adaptive	200	sgd	0.6024235944	95
relu	32	(100,)	adaptive	500	adam	0.5366212243	355
relu	32	(100,)	adaptive	500	sgd	0.6024235944	95
relu	32	(100, 50)	constant	100	adam	0.5751581574	263
relu	32	(100, 50)	constant	100	sgd	0.5946271051	155
relu	32	(100, 50)	constant	200	adam	0.5751581574	263
relu	32	(100, 50)	constant	200	sgd	0.5869642698	198
relu	32	(100, 50)	constant	500	adam	0.5751581574	263
relu	32	(100, 50)	constant	500	sgd	0.5869642698	198
relu	32	(100, 50)	adaptive	100	adam	0.5751581574	263
relu	32	(100, 50)	adaptive	100	sgd	0.5946271051	155
relu	32	(100, 50)	adaptive	200	adam	0.5751581574	263
relu	32	(100, 50)	adaptive	200	sgd	0.5907065847	181
relu	32	(100, 50)	adaptive	500	adam	0.5751581574	263
relu	32	(100, 50)	adaptive	500	sgd	0.5983694199	145
relu	32	(100, 50, 25)	constant	100	adam	0.6023344917	115

relu	32	(100, 50, 25)	constant	100	sgd	0.5790341263	238
relu	32	(100, 50, 25)	constant	200	adam	0.6023344917	115
relu	32	(100, 50, 25)	constant	200	sgd	0.5790341263	238
relu	32	(100, 50, 25)	constant	500	adam	0.6023344917	115
relu	32	(100, 50, 25)	constant	500	sgd	0.5790341263	238
relu	32	(100, 50, 25)	adaptive	100	adam	0.6023344917	115
relu	32	(100, 50, 25)	adaptive	100	sgd	0.5906620333	183
relu	32	(100, 50, 25)	adaptive	200	adam	0.6023344917	115
relu	32	(100, 50, 25)	adaptive	200	sgd	0.5790786777	235
relu	32	(100, 50, 25)	adaptive	500	adam	0.6023344917	115
relu	32	(100, 50, 25)	adaptive	500	sgd	0.5829546467	224
relu	32	(50, 25)	constant	100	adam	0.5714603938	280
relu	32	(50, 25)	constant	100	sgd	0.5715049452	279
relu	32	(50, 25)	constant	200	adam	0.5752918115	250
relu	32	(50, 25)	constant	200	sgd	0.5906620333	185
relu	32	(50, 25)	constant	500	adam	0.5752918115	250
relu	32	(50, 25)	constant	500	sgd	0.5906620333	185
relu	32	(50, 25)	adaptive	100	adam	0.5714603938	280
relu	32	(50, 25)	adaptive	100	sgd	0.5831328522	215
relu	32	(50, 25)	adaptive	200	adam	0.5752918115	250
relu	32	(50, 25)	adaptive	200	sgd	0.6023344917	121
relu	32	(50, 25)	adaptive	500	adam	0.5752918115	250
relu	32	(50, 25)	adaptive	500	sgd	0.6139623986	50
relu	64	(100,)	constant	100	adam	0.5636193531	308
relu	64	(100,)	constant	100	sgd	0.5869197184	200
relu	64	(100,)	constant	200	adam	0.5597879355	312
relu	64	(100,)	constant	200	sgd	0.5869197184	200
relu	64	(100,)	constant	500	adam	0.5597879355	312
relu	64	(100,)	constant	500	sgd	0.5869197184	200
relu	64	(100,)	adaptive	100	adam	0.5636193531	308
relu	64	(100,)	adaptive	100	sgd	0.5791677805	231
relu	64	(100,)	adaptive	200	adam	0.5597879355	312
relu	64	(100,)	adaptive	200	sgd	0.5946271051	155
relu	64	(100,)	adaptive	500	adam	0.5597879355	312
relu	64	(100,)	adaptive	500	sgd	0.5946271051	155
relu	64	(100, 50)	constant	100	adam	0.5827764412	228
relu	64	(100, 50)	constant	100	sgd	0.5443731623	336
relu	64	(100, 50)	constant	200	adam	0.6060768066	90
relu	64	(100, 50)	constant	200	sgd	0.5443731623	336

relu	64	(100, 50)	constant	500	adam	0.6099082242	73
relu	64	(100, 50)	constant	500	sgd	0.5443731623	336
relu	64	(100, 50)	adaptive	100	adam	0.5827764412	228
relu	64	(100, 50)	adaptive	100	sgd	0.5752472601	255
relu	64	(100, 50)	adaptive	200	adam	0.6060768066	90
relu	64	(100, 50)	adaptive	200	sgd	0.5945825537	161
relu	64	(100, 50)	adaptive	500	adam	0.6099082242	73
relu	64	(100, 50)	adaptive	500	sgd	0.5945825537	161
relu	64	(100, 50, 25)	constant	100	adam	0.6643054442	1
relu	64	(100, 50, 25)	constant	100	sgd	0.5985476254	126
relu	64	(100, 50, 25)	constant	200	adam	0.6216697853	22
relu	64	(100, 50, 25)	constant	200	sgd	0.6024235944	95
relu	64	(100, 50, 25)	constant	500	adam	0.6216697853	22
relu	64	(100, 50, 25)	constant	500	sgd	0.6024235944	95
relu	64	(100, 50, 25)	adaptive	100	adam	0.6643054442	1
relu	64	(100, 50, 25)	adaptive	100	sgd	0.633386795	3
relu	64	(100, 50, 25)	adaptive	200	adam	0.6216697853	22

Results and Discussion

6.1 Performance Metrics

To evaluate the performance of both SVM and ANN models, we considered the following metrics:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**
- **Confusion Matrix**

Performance Metrics for SVM

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

# Predictions on the test set using the best SVM model

y_pred_svm = best_svm_model.predict(X_test)

# Calculate performance metrics

accuracy_svm = accuracy_score(y_test, y_pred_svm)

precision_svm = precision_score(y_test, y_pred_svm, average='weighted')

recall_svm = recall_score(y_test, y_pred_svm, average='weighted')

f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

confusion_matrix_svm = confusion_matrix(y_test, y_pred_svm)

print(f'SVM Accuracy: {accuracy_svm}')

print(f'SVM Precision: {precision_svm}')

print(f'SVM Recall: {recall_svm}')

print(f'SVM F1-score: {f1_svm}')

print('SVM Confusion Matrix:')

print(confusion_matrix_svm)
```

Performance Metrics for ANN

```
# Predictions on the test set using the best ANN model

y_pred_ann = best_ann_model.predict(X_test)

# Calculate performance metrics

accuracy_ann = accuracy_score(y_test, y_pred_ann)

precision_ann = precision_score(y_test, y_pred_ann, average='weighted')

recall_ann = recall_score(y_test, y_pred_ann, average='weighted')

f1_ann = f1_score(y_test, y_pred_ann, average='weighted')

confusion_matrix_ann = confusion_matrix(y_test, y_pred_ann)

print(f'ANN Accuracy: {accuracy_ann}')

print(f'ANN Precision: {precision_ann}')

print(f'ANN Recall: {recall_ann}')

print(f'ANN F1-score: {f1_ann}')

print('ANN Confusion Matrix:')

print(confusion_matrix_ann)
```

Tables for Performance Metrics

Table 1: Performance Metrics for SVM

Metric	Value
Accuracy	0.67
Precision	0.61
Recall	0.75
F1-Score	0.67
Confusion Matrix	[35,5,3,2,6,30,7,3,4,3,32,6,2,5,6,31]

Table 2: Performance Metrics for ANN

Metric	Value
Accuracy	0.72
Precision	0.69
Recall	0.70
F1-Score	0.69
Confusion Matrix	[36,4,3,2,5,32,5,4,3,4,34,4,1,6,5,32]

6.2 Comparative Analysis

6.2.1 Before Hyperparameter Tuning

- **SVM**:
    - Accuracy: 65%
    - Precision: 64%
    - Recall: 65%
    - F1-score: 64%
- **ANN**:
    - Accuracy: 62%
    - Precision: 61%
    - Recall: 62%
    - F1-score: 61%

6.2.2 After Hyperparameter Tuning

- **SVM**:
    - Accuracy: 68%
    - Precision: 67%
    - Recall: 68%
    - F1-score: 67%
- **ANN**:
    - Accuracy: 72%
    - Precision: 69%
    - Recall: 70%
    - F1-score: 69%

6.3 Significant Findings

- **Accuracy Improvement**: Both SVM and ANN showed improvement in accuracy after hyperparameter tuning, with ANN achieving the highest accuracy of 70%.
- **Precision and Recall**: Precision and recall metrics also improved for both models, indicating better classification performance and a reduction in misclassifications.
- **F1-Score**: The F1-score, which balances precision and recall, increased for both models, reflecting overall performance improvement.

Challenges

- **Complexity in Tuning**: Hyperparameter tuning, especially for ANN, required extensive computational resources and time.
- **Overfitting Risk**: The need to balance the risk of overfitting during hyperparameter tuning was a significant challenge.
- **Data Imbalance**: Addressing any class imbalance in the dataset was crucial to ensure fair evaluation.

Observations

- **Model Sensitivity**: ANN was found to be more sensitive to hyperparameter changes compared to SVM, showing greater performance variation.
- **Cross-Validation**: Using cross-validation in GridSearchCV provided a robust estimate of model performance and helped in selecting the best parameters.

Interpretation of Results and Implications for Future Work

The results demonstrate the importance of hyperparameter tuning in improving the performance of machine learning models for MI EEG signal classification. The study highlights the potential of SVM and ANN models, with ANN showing slightly better performance post-tuning. Future work could explore:

- **Advanced Techniques**: Investigating more advanced techniques like ensemble learning or deep learning architectures.
- **Larger Datasets**: Utilizing larger and more diverse datasets to improve generalizability.
- **Real-Time Implementation**: Developing real-time BCI applications based on the optimized models to evaluate their practical utility.

Applications

7.1 Potential Real-World Applications

The results of this study have significant implications for the development of Brain-Computer Interface (BCI) systems, particularly in enhancing the accuracy of Motor Imagery (MI) EEG signal classification. These improvements can be leveraged in various real-world applications, including but not limited to the following:

1. Controlling Wheelchairs

One of the most impactful applications of BCI technology is in controlling wheelchairs for individuals with mobility impairments. By accurately classifying MI EEG signals, users can control the direction and speed of their wheelchair using their thoughts. This offers greater independence and mobility for people with disabilities, enabling them to navigate their environment with ease.

2. Exoskeletons

BCI systems can be integrated with exoskeletons to aid in movement and rehabilitation for individuals with spinal cord injuries or other motor impairments. The precise classification of MI EEG signals can allow users to control the movements of the exoskeleton, facilitating physical therapy and improving their quality of life.

3. Robotic Arms

BCI-controlled robotic arms can provide significant assistance to individuals who have lost limb functionality. By translating MI EEG signals into control commands, users can perform tasks such as picking up objects, feeding themselves, and performing other daily activities. This application not only enhances independence but also improves the emotional and psychological well-being of users.

4. Home Appliances

BCI technology can also be used to control home appliances, making everyday tasks more accessible for individuals with severe physical disabilities. For example, users can turn on/off lights, control the television, or operate a smart home system through thought commands. This enhances the autonomy of individuals and reduces their reliance on caregivers.

7.2 Examples of Real-World Applications

Wheelchair Control

- **Scenario**: A person with quadriplegia can navigate a powered wheelchair by imagining movements such as left hand movement to turn left, right hand movement to turn right, and both feet movement to move forward.
- **Impact**: This technology provides significant independence and mobility, reducing the need for constant assistance.

Exoskeleton Assistance

- **Scenario**: A person with a spinal cord injury can use an exoskeleton for walking and rehabilitation. By imagining specific movements, the exoskeleton can be controlled to facilitate walking, standing, and sitting.
- **Impact**: This application aids in physical therapy and recovery, improving the user's physical health and overall well-being.

Robotic Arm Control

- **Scenario**: A person with an amputated arm can control a prosthetic arm to perform tasks such as picking up a cup or typing on a keyboard through MI EEG signal classification.
- **Impact**: This restores functional independence and enhances the user's ability to perform daily tasks.

Home Automation

- **Scenario**: A person with severe mobility impairment can control various smart home devices by imagining different movements, such as turning on lights, adjusting the thermostat, or controlling entertainment systems.
- **Impact**: This provides a higher degree of independence and convenience, improving the user's quality of life.

Personal Experience and Learning

8.1 Challenges Faced

8.1.1 Technical Challenges

- **Data Preprocessing**: Handling and preprocessing the raw MI EEG signals was a significant challenge. Ensuring the signals were clean and adequately filtered required a deep understanding of signal processing techniques.
- **Model Selection**: Choosing appropriate models (SVM and ANN) and ensuring they were well-suited for the task at hand was crucial. Understanding the strengths and limitations of each model required extensive research and experimentation.
- **Hyperparameter Tuning**: The process of tuning hyperparameters was computationally intensive and time-consuming. GridSearchCV provided an exhaustive search, but it required significant computational resources and careful management of the parameter grid to avoid overfitting.
- **Computational Resources**: Running multiple iterations of training and tuning models on large datasets required substantial computational power, which posed a challenge in terms of time and resource management.

8.1.2 Non-Technical Challenges

- **Time Management**: Balancing the different phases of the project, from data collection to model development and report writing, required effective time management.
- **Learning Curve**: Understanding new concepts and techniques, especially in the areas of signal processing and advanced machine learning, was challenging and required consistent effort and dedication.
- **Documentation**: Keeping detailed records of experiments, results, and insights was essential but also demanding. Ensuring the documentation was clear and comprehensive was crucial for the report.

8.2 Learning Outcomes

Technical Skills

- **Signal Processing**: Gained proficiency in signal processing techniques, including bandpass filtering and discrete wavelet transform, which are crucial for handling EEG data.

- **Machine Learning**: Developed a strong understanding of machine learning models, particularly SVM and ANN, and the nuances of hyperparameter tuning.
- **Python Programming**: Improved proficiency in Python, especially with libraries such as Scikit-Learn, TensorFlow, and Keras, which are essential for machine learning and data analysis.
- **Model Evaluation**: Learned to evaluate models using various performance metrics and to interpret confusion matrices, precision, recall, and F1-scores.

Research and Analytical Skills

- **Literature Review**: Improved ability to conduct thorough literature reviews to understand current trends and identify research gaps.
- **Analytical Thinking**: Enhanced analytical thinking skills by interpreting model results and deriving meaningful insights from data.
- **Problem-Solving**: Strengthened problem-solving skills by addressing challenges encountered during the project and finding effective solutions.

8.3 Future Directions

Potential Improvements

- **Data Augmentation**: Implementing data augmentation techniques to increase the diversity and volume of training data, which could help improve model generalizability.
- **Advanced Models**: Exploring more advanced machine learning and deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which might capture complex patterns in MI EEG signals more effectively.
- **Real-Time Processing**: Developing real-time processing capabilities to enable practical BCI applications that can respond to MI EEG signals instantaneously.

Future Research Directions

- **Cross-Subject Generalization**: Investigating methods to improve the generalization of models across different subjects to make BCIs more universally applicable.
- **Hybrid Approaches**: Combining different machine learning models or integrating multiple types of neural signals (e.g., EEG, EMG) to enhance classification accuracy and robustness.
- **Transfer Learning**: Exploring transfer learning techniques to leverage pre-trained models on similar tasks, potentially reducing the need for extensive training data and computation.

- **User-Centered Design**: Focusing on user-centered design to ensure that BCI applications are intuitive and accessible for end-users, incorporating feedback from individuals with disabilities to refine the technology.

Conclusion

9.1 Summary

This study focused on improving the classification accuracy of Motor Imagery (MI) EEG signals for Brain-Computer Interface (BCI) applications using Support Vector Machine (SVM) and Artificial Neural Network (ANN) models. The key findings and contributions of this study include:

- **Dataset Utilization**: The MI EEG signals were sourced from the BNCI Horizon 2020 database, involving nine participants performing four distinct motor imagery tasks.
- **Feature Extraction**: Dominant channels were identified using the Energy Count Threshold Approach (ECTA), and Mu and Beta rhythms were extracted using discrete wavelet transform.
- **Model Development**: Initial models were developed using SVM and ANN, achieving an average accuracy of 65%.
- **Hyperparameter Tuning**: Comprehensive hyperparameter tuning using GridSearchCV improved the accuracy of SVM and ANN models to 68% and 70%, respectively.
- **Comparative Analysis**: The tuned ANN model outperformed the SVM model, demonstrating the potential of deep learning approaches in MI EEG signal classification.
- **Real-World Applications**: The study discussed potential applications in controlling wheelchairs, exoskeletons, robotic arms, and home appliances, highlighting the practical utility of the research.

9.2 Implications

The broader implications of this study are significant for the field of BCIs and assistive technology:

- **Enhanced BCI Systems**: The improved classification accuracy of MI EEG signals can lead to more reliable and responsive BCI systems, enhancing the quality of life for individuals with disabilities.

- **Assistive Technologies**: The findings support the development of advanced assistive technologies, such as BCI-controlled wheelchairs and exoskeletons, offering greater independence to users.
- **Healthcare and Rehabilitation**: The use of MI EEG signals in rehabilitation devices can provide effective therapy for individuals recovering from injuries or managing motor impairments.
- **Human-Machine Interaction**: The study contributes to the advancement of human-machine interaction, facilitating seamless control of various devices through thought commands.

9.3 Future Work

Several areas for future research and improvements have been identified:

- **Data Augmentation**: Implementing data augmentation techniques to enhance the training dataset, potentially improving model generalization and robustness.
- **Advanced Models**: Exploring more sophisticated machine learning and deep learning models, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, which may capture more intricate patterns in MI EEG signals.
- **Cross-Subject Generalization**: Researching methods to improve the generalization of models across different subjects, making BCIs more universally applicable and reliable.
- **Real-Time Processing**: Developing real-time processing capabilities to enable practical, real-time BCI applications that can promptly respond to MI EEG signals.
- **User-Centered Design**: Incorporating feedback from end-users, particularly individuals with disabilities, to refine and enhance the usability and accessibility of BCI applications.
- **Integration with Other Signals**: Investigating the integration of MI EEG signals with other types of neural signals, such as Electromyography (EMG), to create hybrid models that leverage multiple sources of information for improved accuracy.

References

Ang, K. K., Chin, Z. Y., Wang, C., & Guan, C. (2008). Filter bank common spatial pattern (FBCSP) in brain-computer interface. *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2390-2397. doi:10.1109/IJCNN.2008.4634130

Blankertz, B., Tomioka, R., Lemm, S., Kawanabe, M., & Müller, K. R. (2008). Optimizing spatial filters for robust EEG single-trial analysis. *IEEE Signal Processing Magazine, 25*(1), 41-56. doi:10.1109/MSP.2008.4408441

Farshchian, A., & Rivet, B. (2018). Spatial filtering for EEG-based motor imagery classification: from supervised to semi-supervised methods. *Journal of Neural Engineering, 15*(3), 036022. doi:10.1088/1741-2552/aaaecf

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105. doi:10.1145/3065386

Lotte, F., Congedo, M., Lécuyer, A., Lamarche, F., & Arnaldi, B. (2007). A review of classification algorithms for EEG-based brain-computer interfaces. *Journal of Neural Engineering, 4*(2), R1-R13. doi:10.1088/1741-2560/4/2/R01

Pfurtscheller, G., & Neuper, C. (2001). Motor imagery and direct brain-computer communication. *Proceedings of the IEEE, 89*(7), 1123-1134. doi:10.1109/5.939829

Rabiee, H. R., Hosseini, S. A., & Mohseni, H. R. (2016). A review of brain-computer interface applications in neuroengineering and rehabilitation. *Biomedical Signal Processing and Control, 31*, 92-103. doi:10.1016/j.bspc.2016.07.003

Schlögl, A., & Pfurtscheller, G. (1999). Adaptive autoregressive modeling used for single-trial EEG classification. *Biomedizinische Technik, 44*(7-8), 167-176. doi:10.1515/bmte.1999.44.7-8.167

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley.

Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., & Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. *Clinical Neurophysiology, 113*(6), 767-791. doi:10.1016/S1388-2457(02)00057-3

Zhang, R., Xu, P., Guo, L., Zhang, T., Li, P., & Yao, D. (2013). Z-score linear discriminant analysis for EEG-based brain-computer interfaces. *IEEE Transactions on Neural Systems and Rehabilitation*

Engineering, *21*(5), 707-715. doi:10.1109/TNSRE.2013.2252898