

**Study on the Analysis of Brain Disorders EEG Signals and
a High-Precision BCI System Based on Machine Learning Methods**

機械学習に基づいた脳疾患 EEG 信号の解析及び高精度 BCI システムの
構築に関する研究

by

Boning Li

*Submitted in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy*

Supervised by Prof. Jianting Cao

Graduate School of Engineering,
Saitama Institute of Technology

Fukaya, Saitama

January 2024

Acknowledgment

I would like to express my heartfelt gratitude to Professor Jianting Cao for his invaluable guidance, unwavering support, and profound wisdom throughout my doctoral research journey. His mentorship has been instrumental in shaping the course of my academic and personal development.

Professor Cao's dedication to research excellence and his passion for teaching have been a constant source of inspiration. His insightful feedback, constructive criticism, and patient guidance have been crucial in refining my research methodologies and analytical skills. I have truly benefited from his extensive knowledge and expertise in the field. I am also grateful for Professor Cao's generosity with his time. Despite his busy schedule, he always made himself available for discussions and provided timely assistance whenever I encountered challenges in my research. His willingness to engage in intellectual discourse and his ability to ask thought-provoking questions have pushed me to think critically and explore new horizons in my research.

I want to acknowledge Professor Cao's role in fostering a supportive and collaborative research environment within our department. His encouragement to collaborate with fellow students and researchers has enriched my academic experience and broadened my perspective. I have had the privilege of working alongside talented colleagues, and I owe this opportunity to Professor Cao's inclusive leadership. Professor Cao has been an exceptional mentor, teacher, and role model. His contributions to my academic and personal growth are immeasurable, and I am profoundly grateful for his unwavering support throughout my doctoral journey. I consider myself fortunate to have had the opportunity to work under his guidance, and I hope to continue carrying forward the lessons and values he has instilled in me.

Abstract

Electroencephalography (EEG) is a method for recording brain activity as electrophysiological indices, accurately reflecting the state of brain activity. Proper processing and analysis of EEG signals not only aid in the diagnosis of brain disorders such as coma, brain death, and epilepsy but also enable the control of external devices using EEG signals. EEG signals are very weak electrophysiological signals, susceptible to noise, making their collection and processing challenging. For physicians diagnosing patients' brain activity states using EEG signals, making rapid and accurate diagnoses from noise-contaminated EEG signals is difficult.

In this study, we propose EEG signals de-noising and feature selection methods, considering the EEG signal collection environment and patient condition, based on the clinical interpretation standard. Initially, we conducted research on the diagnosis of coma/brain-death brain activity states. We applied our proposed method to coma/brain-death patient EEG signals and constructed a neural network model using deep learning methods. The results achieved a classification accuracy of 99.71% and an F1 score of 99.71%. Subsequently, we conducted research on the classification of epileptic focus and non-focus channel iEEG signals, building a neural network model using deep learning methods. The classification accuracy reached 85.14%. By combining our proposed method with deep learning techniques, we proved their effectiveness and reliability, offering valuable support for physicians' clinical diagnoses. In our research on constructing a Brain-Computer Interface (BCI) system, we developed a P300 visual stimulator, leveraging the principle that subjects produce characteristic EEG signals in response to visual stimuli. We constructed a high-accuracy BCI system using an SVM classifier. This enabled real-time control of robot movement using EEG signals, accomplishing tasks such as wheelchair operation. We established a theoretical foundation for applying EEG signals in the field of welfare.

Abstract (和文)

脳電図(EEG)は、脳活動を電気生理学的指標として記録する手法であり、脳活動状態を正確に反映できる。EEG 信号に適切な処理・分析を通じて、昏睡、脳死、癲癇などの脳疾患の診断に役立つだけでなく、EEG 信号を用いて外部装置を制御することも可能である。EEG 信号は非常に微弱な電気生理学的信号であり、雑音に影響されやすく、その採集と処理は難しい。医師が EEG 信号を用いて患者の脳活動状態を診断する際、雑音が混入された EEG 信号に対し、迅速かつ正確な診断を下すことは難しい。

本研究では、臨床脳波検査基準に基づき、EEG 信号採集環境と患者状態を考慮しながら、雑音除去法と脳波特徴選択法を提案する。まず、昏睡/脳死脳活動状態の診断に関する研究を行った。昏睡/脳死患者 EEG 信号に提案手法を利用して、深層学習法を用いたニューラルネットワークモデルを構築した。その結果、分類精度は 99.71%、F1 スコアは 99.71% に達した。次に、癲癇焦点と非焦点チャンネル iEEG 信号分類に関する研究を行い、深層学習法を用いたニューラルネットワークモデルを構築した。その結果、分類精度は 85.14% に達した。提案手法と深層学習法を組み合わせ使用し、その有効性と信頼性を証明し、医師の臨床診断に有効な支援を提供できることが確認された。ブレイン・コンピューター・インターフェース(BCI)システムの構築に関する研究において、被験者が視覚刺激に反応して特徴的な EEG 信号が生成される原理を活用し、P300 視覚刺激器を開発した。SVM 分類器を用いて高精度 BCI システムを構築した。これにより、EEG 信号を用いてリアルタイムでロボットの動きを制御し、車椅子を操作するなどのタスクを実現した。EEG 信号を福祉分野へ応用するための理論基盤を構築した。

Content

<i>Acknowledgment</i>	3
<i>Abstract</i>	5
<i>Abstract (和文)</i>	6
1. Introduction	10
1.1. Brain Disorders	10
1.1.1. Coma/Brain-Death	10
1.1.2. Epilepsy	15
1.2. Brain-Computer-Interface System	17
1.2.1. Steady State Visual Evoked Potential (SSVEP)	17
1.2.2. Event-Related Potential Positive 300 (ERP P300)	18
1.2.3. Motor Imagery (MI)	18
1.3. Signal Processing	20
1.4. Machine Learning	20
2. Classification of Coma/Brain-Death EEG Dataset	24
2.1. Coma/Brain-Death dataset	25
2.2. Data pre-processing method	26
2.3. The Proposed One-dimensional Convolutional Neural Networks (1D-CNN)	30
2.4. Experiment and Result	33
3. The Automated Localization of Epileptic Foci	35
3.1. The Bern-Barcelona iEEG Database	35
3.2. The Proposed One-dimensional Convolutional Neural Networks (1D-CNN)	36

3.3. Experiment and Result	38
4. Construct of High-Precision BCI System	42
4.1. P300 Visual Stimulator	42
4.2. Support Vector Machine.....	43
4.3. P300-based Brain-Computer Interface System	44
4.4. Experiment and Result	45
4.4.1. Offline Stage	47
4.4.2. Offline Stage Results	49
4.4.3. Online Stage.....	49
4.4.4. Online Stage Results.....	50
5. Conclusion and Future Work.....	51
Reference	54
Appendix.....	59

1. Introduction

Brainwave, when not specifically specified, means a recording of the electrical signal activity of the brain captured by placing electrodes on the surface of the scalp, and is generally written as Electroencephalogram (EEG). Brainwaves recorded by placing intracranial electrodes on the surface of the cerebral cortex are known as intracranial waves, or Intracranial Electroencephalography (iEEG). For the use of electrophysiological signals from the brain for different purposes, the signals required are different.

1.1. Brain Disorders

EEG patterns, play a crucial role in the understanding and diagnosis of various brain disorders. These electrical signals, which represent the collective activity of neurons in the brain, offer a window into the functional state of the brain. In conditions such as diagnosis of coma/brain-death [4, 7, 12, 14, 16], epilepsy [11, 17, 19], sleep disorder [8], Alzheimer's disease, and Parkinson's disease, characteristic alterations in EEG patterns are often observed.

1.1.1. Coma/Brain-Death

EEG signals are commonly used to diagnose the brain state of clinically coma/brain-death patients. The diagnosis of coma/brain-death usually requires a rigorous process to be followed. According to the world brain-death project [9], brain-death clinical examination should contain:

(1) There is no evidence of arousal or awareness to maximal external stimulation, including noxious visual, auditory, and tactile stimulation.

- (2) Pupils are fixed in a midsize or dilated position and are nonreactive to light;
- (3) Corneal, oculocephalic, and oculovestibular reflexes are absent.
- (4) There is no facial movement to noxious stimulation.
- (5) The gag reflex is absent to bilateral posterior pharyngeal stimulation.
- (6) The cough reflex is absent to deep tracheal suctioning.
- (7) There is no brain-mediated motor response to noxious stimulation of the limbs.
- (8) Spontaneous respirations are not observed when apnea test targets reach $pH < 7.30$ and $PaCO_2 \geq 60$ mm Hg [9].

In the previous study, the following coma/brain-death clinical diagnostic process was proposed. Figure 1 shows the proposed brain-death clinical examination flow [5].

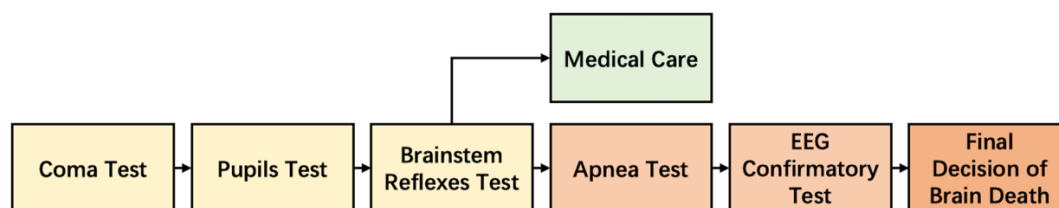


Fig. 1. The brain-death clinical examination flow.

Confounders such as the effects of certain medications, metabolic abnormalities, or cardiopulmonary instability may lead to misjudgment of the clinical examination for brain-death. Therefore, testing and evaluation of the absence of brain blood flow or electrical activity may be necessary at times. Furthermore, during the clinical examination, which includes apnea test, it is essential to assess the absence of brain blood flow or electrical activity in the brain before the apnea test as the apnea test may cause the irreparable loss of brain function or even direct lead to death.

Compared to acquiring EEG signals from epileptic foci channels (such as the Bern-Barcelona EEG Database [3], Bonn EEG Time Series Database [2], and the

CHB-MIT Scalp EEG Database [15] etc.), the acquisition of coma/brain-death EEG signals is performed in a more demanding environment, generally in the Intensive Care Unit (ICU). The acquisition of coma/brain-death EEG signals involve personal privacy protection. Moreover, it is necessary to adjust the electrode position according to the patient's personal condition.

The Brain-Death EEG Database we used in this study is carried out in a hospital in Shanghai and classified into a deep-coma group (39 patients) and a brain-death group (19 patients). All patients' EEG signals acquisition and recording are with the permission of the patients' families. An EEG-based preliminary examination system was developed during the standard clinical procedure. Figure 2 shows the layout of the electrodes. Several statistics-based signal processing tools have been developed for the signal detection or extraction, denoising and disease classification. A robust principal factor analysis (PFA) associated with independent component analysis (ICA) approach is developed to reduce the power of additive noise and separate the brain activities and interference sources based on the Brain-Death EEG Database [4, 5, 6]. Tensor Train Decomposition (TT) was applied to analyze coma/brain-death EEG signals. Reshape the EEG data from matrix to higher-order tensor to extract more valuable features, and the support vector machine (SVM) classifier is used to complete the classification task [7]. Method to discriminate discrete states of brain consciousness by examining nonlinear features in the EEG have been proposed. This approach uses convex combinations of adaptive filters to implement a collaborative adaptive filter architecture. Simulations based on different filter combinations show that this approach is suitable for discriminating coma and quasi-brain-death states of patients based on fundamental EEG signal features [12].

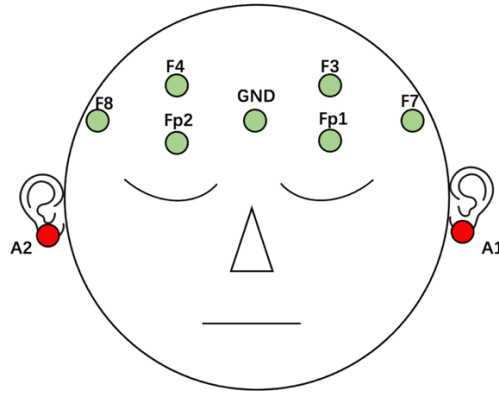


Fig. 2. The electrode layout of the Brain-Death EEG signal recording.

Multivariate Multiscale Entropy (MMSE) is used for brain consciousness analysis [1]. Compare with Multiscale Entropy (MSE) method, MMSE could conduct a comprehensive analysis of the complexity of the underlying signal generating system. For higher MMSE scales the coma patients showed higher complexity than the quasi-brain-death patients, which indicates a reduction in the intra-cortical information flow and lower neuronal process in the brain for the quasi-brain-death patients [1]. Empirical Mode Decomposition (EMD) technique is used to extract informative brain activity features of the real-world recorded clinical EEG data. In addition, the power spectrum technique was used to assess significant differences between the coma and quasi-brain-death patient groups. The power values indicated a high intensity of brain activity in the coma patient group, whereas these activities were absent in the quasi-brain-death patient group. Comparing the results with clinical diagnosis, the EMD method has shown its validity and reliability [14].

The MMSE method can characterize the brain state of consciousness from multi-channel EEG recordings, and the EMD method is applied randomly to the raw EEG signal from one channel. Both have provided a comprehensive analysis of EEG signals from suspected brain-death patients in terms of EEG signal complexity analysis and energy analysis and have yielded valid and reliable

results. However, both methods face the same problem of being limited by the size of the data set, and confirmation of method reliability requires additional data analysis.

In this study, we pre-process the acquired raw EEG Database, train and test the neural network using the pre-processed coma/brain-death EEG signals and complete the task of pre-processed coma/brain-death EEG signals classification. The classification results are used to analyze the constructed neural network model and to verify the feasibility as well as the reliability of this dataset for the classification task in deep learning.

During the coma/brain-death EEG signals acquisition, EEG signals are susceptible to various interferences. Electromagnetic environmental noise can erroneously suggest cerebral electrical activity. Sedation, hypothermia, toxic states, metabolic disorders also cause unexpected changes in the amplitude of EEG [9]. We attempt to filter out the external electromagnetic environmental noise from EEG signals by using a band-pass filter. For partial abnormal EEG signals with unexpected amplitude changes generated by sedation, metabolic disorders, etc. we set a threshold for the amplitude ($n=75$), and the signals with a maximum amplitude greater than this threshold are considered to contain information that may confound the diagnosis of the brain-death and are rejected. Finally, we rejected the data containing noise that could not be removed by the band-pass filter, and invalid data that were below the amplitude threshold but would clearly interfere with the dataset by visual inspection. Instead of the manual feature extraction method, a 49-layer end-to-end one-dimensional convolutional neural network (1D-CNN) model is used to complete the classification task of pre-processed coma/brain-death EEG signals. K-fold Cross Validation ($K=5$) is used to make the model output more reliable.

During the training phase of the model, the training accuracy reached 100%. In 5-fold cross-validation, we get the mean test accuracy with 99.71%, recall score with 99.51%, F1-score with 99.71% and AUC with 99.89%, which means our model is well performed in the classification task of the coma/brain-death EEG signals.

1.1.2. Epilepsy

iEEG is often used by clinical experts to determine the location of the epileptic focal in the treatment of epilepsy. According to the World Health Organization, there are more than 50 million people with epilepsy in the world. Epilepsy has become one of the most common neurological diseases in the world. Epilepsy is a chronic disease of the brain, which is caused by abnormal discharge of some brain tissue. Up to 70% of epileptic patients can control seizures through the proper use of antiepileptic drugs. For patients with drug-resistant, surgical treatment may be useful [24].

The difficulty of surgical treatment of epilepsy lies in the accurate localization of epileptic foci before the operation. Clinical experts need to place multiple electrodes in the patient's scalp, record EEG for one week and visually detects the obtained EEG to speculate the location of abnormal discharge of brain tissue, and then perform resection surgery [31]. It is a heavy burden for clinicians, both time-consuming and strenuous. Hence, there is an urgent need for a technique which could automatically identify epileptic focal signals.

In recent years, machine learning has been widely used in various fields, including biomedical field [30]. The application of various machine learning methods has greatly reduced the burden on clinical experts. In neuroscience, various machine learning methods are often used to process EEG signals to assist

clinicians in diagnosing patients.

The most common sequential steps are pre-processing, feature extraction, training, and classification when developing an automated diagnostic system by using machine learning [29]. To standardize the input of the model in the subsequent step, the raw signals are always being normalized and transformed in the pre-processing stage. Entropy [18, 20, 23, 26, 27], Wavelet Transform [22], Fourier Transform, Empirical Mode Decomposition (EMD) [26, 20] and other methods are always used to extract the significant features from the signals in the feature extraction stage. In the training stage, K-Nearest Neighbor (KNN), Support Vector Machine (SVM) [23, 19], Recurrent Neural Network (RNN) [25] and the other neural network, are widely used for the classification of features obtained by hand-crafted feature extraction methods.

Instead of the manual feature extraction method, a 21-layer end-to-end one-dimensional convolutional neural network (1D-CNN) is used to the automated classification of focal and non-focal iEEG signals in this study. Focal iEEG signals could be classified automatically from the iEEG signal recordings. We use the Bern-Barcelona dataset to perform the classification process. Raw signals will be the input of the network, no pre-processing and no feature extraction so that the computational is reduced significantly. In 10-fold cross-validation, the average accuracy is about 85.14%.

1.2. Brain-Computer-Interface System

EEG is also very widely used in the construction of Brain-Computer Interface (BCI) system. BCI as a frontier technology, have sparked wide-ranging interest in scientific research [32, 36, 46]. The crux of its novelty lies in allowing users to interact and control the external environment directly using brain signals [33, 41], providing vast new horizons for the advancement of fields such as neuroscience, artificial intelligence, and biomedical engineering. Particularly in aiding disabled individuals to regain perceptual and behavioral abilities [34, 43], as well as applications in gaming [35] and virtual reality (VR), BCIs have shown limitless potential. P300, SSVEP, and MI represent the three most used neurophysiological signal types in BCI research, each carrying its unique advantages and challenges.

1.2.1. Steady State Visual Evoked Potential (SSVEP)

Steady-state visually evoked potentials (SSVEP) are oscillatory responses that can be elicited in the visual cortex by repetitive visual stimulation at a fixed frequency, typically ranging from 3.5 Hz to 75 Hz [47]. As a reliable and non-invasive technique, SSVEPs have been extensively used in research and application of brain-computer interfaces due to their high signal-to-noise ratio, less subject training, and considerable information transfer rate [39, 48]. The optimal placement of electrodes for SSVEP recording is typically determined by the nature of the visual stimuli and the goals of the experiment. Most studies report the maximal SSVEP responses in the occipital region, specifically at Oz, O1, and O2 electrode locations, according to the international 10-20 EEG system. This is attributed to the anatomical proximity of these locations to the primary

visual cortex which is most directly activated by visual stimuli.

1.2.2. Event-Related Potential Positive 300 (ERP P300)

P300 belongs to a type of Event-Related Potential (ERP), which is an endogenous, specialized evoked potential related to cognitive function [45]. P300 is a positive wave that appears approximately 300 ms after the occurrence of an event (e.g., auditory, visual stimulus), and it is a fusion of a delta (0.5-4 Hz) rhyme brainwave as the main contributor and a theta (4-7.5 Hz) rhyme brainwave response. Its nomenclature indicates that it is a forward wave relative to the reference voltage P [38]. The time interval from the application of the stimulus to the peak of the wave is about 300 ms, but 300 only indicates a time interval, and in fact the latency is 250ms-800ms, so intervals greater than 300ms should be considered when sampling data from a time series. P300 can be evoked by visual, auditory, and somatosensory stimuli, and can be used to recognize neural activities related to cognitive processes in the human brain. Through the in-depth study of the neural mechanism of P300, researchers can help human beings to further explore neuroscience and form a more perfect theoretical guidance, which can develop its applications in medical diagnosis, engineering applications, neuroscience, and is also of great significance to further understanding of the human brain.

1.2.3. Motor Imagery (MI)

Motor Imagery (MI), when a person imagines his/her own limbs (or muscles) moving but there is no actual movement output, there will still be activation in specific brain regions of the person. By analyzing the EEG signals and detecting and identifying the activation effects of different brain regions to determine the

user's intention, direct communication and control between the human brain and external devices can be realized [37, 40, 42]. Currently, the common parts of motor imagery are left and right, right hand, feet, and tongue.

During motor imagery, the cerebral cortex generates two kinds of rhythmic signals with obvious variations, namely, the μ -rhythm signal of 8-15 Hz and the β -rhythm of 18-24 Hz. During motor imagery, neuronal cells are activated, and metabolism is accelerated, and the EEG rhythmic energy in the contralateral motor sensory area of the cerebral cortex decreases significantly, while the EEG rhythmic energy in the ipsilateral motor sensory area increases, a phenomenon known as Event Related Desynchronization (ERD)/Event Related Synchronization (ERS). Based on this relationship, multiple control commands can be generated by actively controlling the amplitude of μ and β rhythms in the left and right brains. For different research and application scenarios and purposes, different neurophysiological signal types are used.

In this study, we focus on developing a 3x3 grid pattern stimulator reliant on P300 visual stimuli. This stimulator elicits the user's P300 responses by probabilistically flashing a white circular target stimulus. To verify the efficacy of our design, we initially conducted a series of offline experiments, acquiring subjects' EEG data using the Muse EEG equipment. The results indicated that the Muse EEG equipment could effectively capture P300 EEG components. Subsequently, we extracted feature data from the experimental data collected from 3 subjects, which was then used to train a Support Vector Machine (SVM) classifier. The trained classifier achieved an accuracy rate of 84.1% in an offline environment. Upon successful completion of the offline experiments, we proceeded with real-time experiments. In the real-time experiments, we used by

the pre-trained SVM classifier to process in real time the EEG signals containing P300 components acquired from the Muse EEG equipment and successfully transformed the classification results into control commands for robots. The success rate of this process reached 81.2%, further affirming the reliability of our system in practical operational environments.

1.3. Signal Processing

EEG signals, as non-stationary signals, usually cannot be analyzed directly using traditional signal processing methods such as Fourier transform. In previous studies, signal processing methods such as Empirical Mode Decomposition (EMD), Turning Tangent Empirical Mode Decomposition (2T-EMD), and Multiple Empirical Mode Decomposition (M-EMD) analysis are usually used to decompose the EEG signals of non-stationary signals into a series of Intrinsic Mode Functions (IMF). The decomposed IMFs can be regarded as EEG signals containing concentrated frequency band information. At this point, by applying signal processing methods such as Fourier transform to the IMFs, the properties of the EEG signal can be analyzed and interpreted.

1.4. Machine Learning

Traditional machine learning methods are commonly used for the task of classifying EEG signals. Support Vector Machines (SVM) is a supervised learning algorithm that has been widely applied to various problems, particularly in classification issues within BCI systems. The main goal of SVM is to find an optimal hyperplane that maximizes the margin between different classes, thereby effectively partitioning the binary classification problem. In a typical

dataset(x_i, y_i), x_i denotes input features, and y_i represents class labels. The primary task of SVM is to find a hyperplane that maximizes the distance from the hyperplane to the nearest points of the two classes.

Linear Discriminant Analysis (LDA) is a classical linear learning method that can be used for both classification problems and supervised feature dimensionality reduction. The main goal of LDA is: given a training sample, try to project the sample features onto a vector, and hope that the projection points of similar samples are as close as possible, and the projection points of dissimilar samples are as far away as possible. When classifying a new sample, the features of the new sample are projected onto this vector as usual, and then the location of the projection point is used to determine the class of the new sample. LDA can be summarized as the projection with the minimum variance within a class and the maximum variance between classes.

Canonical Correlation Analysis (CCA) is a statistical method that seeks to identify and measure the associations between two sets of variables [49]. This multivariate analysis technique extracts canonical variates linear combinations of the original variables from both variable sets that have maximal correlation with each other. As a result, CCA has been widely used in numerous fields, such as ecology, economics, psychology, and recently, it has gained considerable attention in signal processing and machine learning research [50].

In the construction of BCI system, CCA is utilized to maximize the correlation between the measured EEG signals and the reference signals, which are typically derived from the characteristics of the stimuli. In fact, CCA-based SSVEP detection has proven to be one of the most effective methods, achieving high accuracy and robustness even in noisy environments [51]. This method has demonstrated a high degree of effectiveness due to its capacity to simultaneously

consider multiple harmonics of the stimulus frequencies, increasing the distinguishability of the responses.

EEG signals are very susceptible to noise. In the process of clinical interpretation, EEG signals containing a large amount of noise can cause significant problems for doctors' clinical diagnosis. When signal processing and machine learning methods are used to analyze and classify EEG signals containing noise, the noise can also lead to misinterpretation and even misclassification of the signals. Therefore, there is a need for a method that can minimize the interference of noise to assist in clinical diagnosis and the task of parsing and classifying EEG signals. Therefore, in this study, we propose an EEG signal pre-processing mode based on the clinical interpretation standard of EEG signals that considers the characteristics of the EEG signals, the state of the patient, and the state of the EEG device. The proposed EEG signal pre-processing mode shows in Fig. 3.

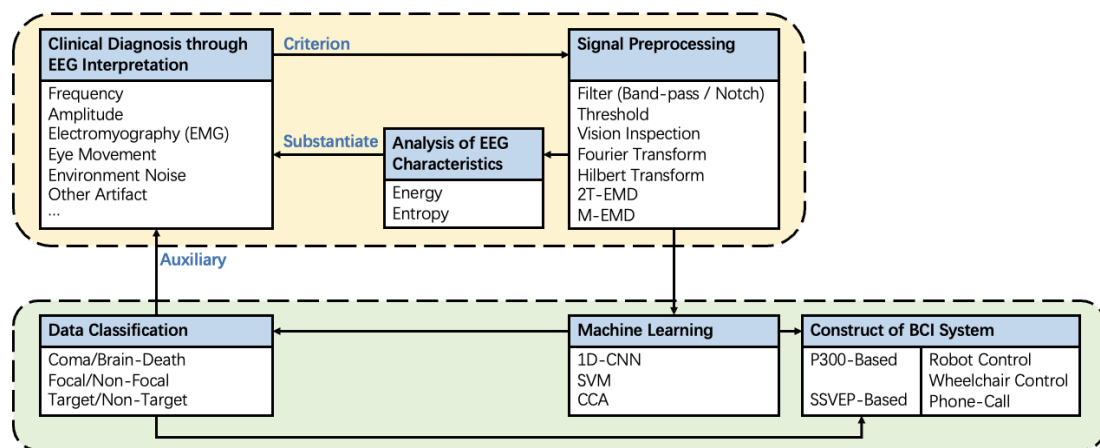


Fig. 3 The proposed EEG signal pre-processing mode.

When pre-processing EEG signals from coma/brain-death patients, the following points need to be noted. First, the EEG signal is acquired from the patient's scalp, and its effective frequency domain is 0.1-40 Hz, while the AC noise is generally 50 Hz. For this part of electromagnetic noise, it can be removed by band-pass

filter and notch filter. Next, the filtered EEG signal, as the effect of electromagnetic noise has been reduced to do little, the cause of the EEG signal to produce high amplitude noise may be drug injection, metabolic disorders, poor electrode contact and so on. In contrast, the brain activity of coma/brain-death patients is very weak, and EEG signals above $75\mu V$ do not occur in clinical interpretation standard. Therefore, signals with amplitudes above $75\mu V$ can be removed by setting a threshold. Finally, due to the patient's muscle spasms and other reasons, low-amplitude, and low-frequency myoelectric usually appear in the EEG signal. For this part of the noise, it can be removed by visual inspection or algorithms, etc. The amplitude of the EEG signal usually does not reach more than $100\mu V$. In the pre-processing of sleep EEG signals, based on the clinical interpretation standard of EEG signals, the EEG signals acquired in the posterior cephalic lobe of human beings in the short period of time before they enter sleep will contain α rhyme with an amplitude of more than $100\mu V$ for a short period of time. At this point in the pre-processing of the sleep EEG signal, it is not possible to remove EEG signals that exceed $75\mu V$ or more.

In the following chapters, we will present studies related to classification of coma/brain-death EEG dataset in Chapter 2. Chapter 3 will present studies related to the automated localization of epileptic foci. Chapter 4 will present studies related to the construction of a high-precision BCI system and Chapter 5 will summarize and present the future works.

2. Classification of Coma/Brain-Death EEG Dataset

Comparing with the EEG signal acquisition for epilepsy, Alzheimer's disease, etc., coma/brain-death EEG signals acquisition and recording is generally performed in a more demanding environment. As a hospital ward for serious illnesses and injuries requiring highly intensive medical care, the ICU usually has strict time limits for non-physician personnel to avoid external man-made disturbances and bacterial infection. The ICU is equipped with many devices and instruments to always monitor the patient's condition. However, electromagnetic environmental noise caused by the operation of many electronic devices may have a significant impact on EEG acquisition which makes the acquired EEG signals mixed with prolonged, high-amplitude electromagnetic environmental noise and therefore impact EEG reading and physician immediate diagnosis. Moreover, for the specificity of the patient's condition and the timely management of emergencies, the acquisition of coma/brain-death EEG signals is much more difficult on the condition of without disturbing the physician's emergency treatment.

Therefore, we propose a band-pass filter and threshold rejection-based EEG signal pre-processing method to enable physicians to obtain real-time, relatively low-noise or even noise-free EEG signals from patients quickly and accurately in a short period of time. Moreover, to assist physicians accurately and efficiently in making correct judgments, we propose a one-dimensional convolutional neural network (1D-CNN) model to classify the pre-processed EEG signals to coma EEG signal or brain-death EEG signal. Fig. 4. shows the pre-process of the coma/brain-death EEG signals and the training-test process of the proposed 1D-CNN model.

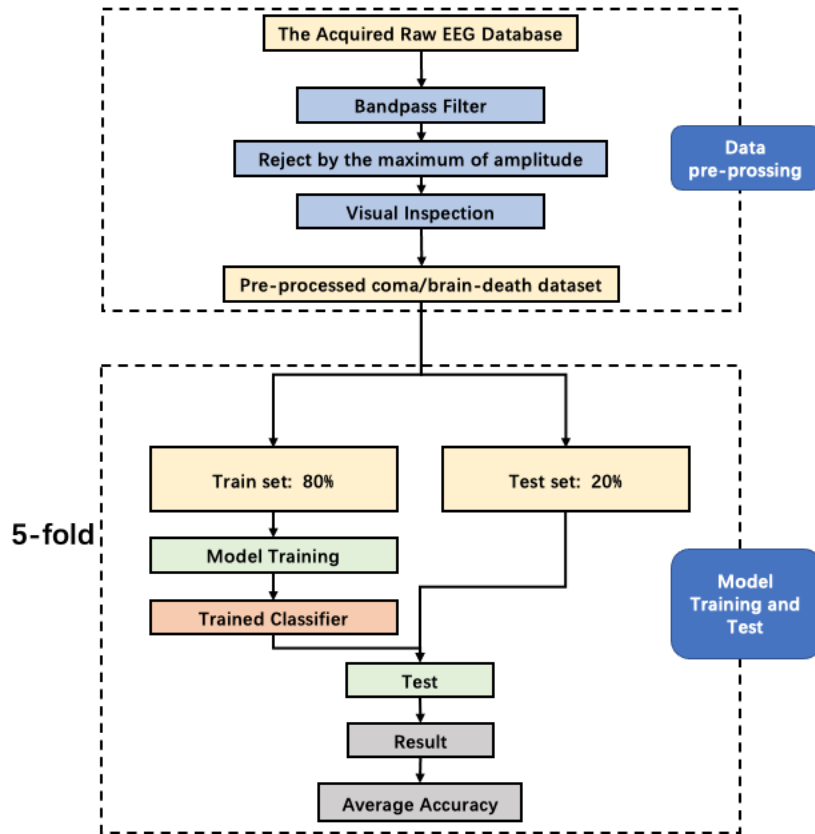


Fig. 4. The structure of the proposed 1D-CNN model associated with EEG signals pre-processing.

2.1. Coma/Brain-Death dataset

The coma/brain-death EEG signals carried out in a hospital in Shanghai. A portable EEG system (NEUROSCAN ESI) connected to a patient to seek brain activity. The EEG signals were directly recorded at the bedside of the patients in the intensive care units. Only nine electrodes were chosen to apply to patients. Among these electrodes, six exploring electrodes (Fp1, Fp2, F3, F4, F7, F8) as well as ground (GND) electrode were placed on the forehead, and two electrodes (A1, A2) as the reference were placed on the earlobes based on the standardized 10-20 system. The sampling rate of EEG was 1000 Hz, and the resistances of the electrodes were set to less than $8k\Omega$. The layout of the electrodes can be seen in Figure 2. The EEG recordings were supervised by one physician (neurologist) and operated by a medical staff [16].

With the permission of the patients' families, a total of 58 patients has been examined by using EEG. All patients were examined by the coma test, pupils test, and brainstem reflex test by two physicians. The patients were classified into a deep-coma group and a brain-death group before the EEG recording (two patients belong to both coma and brain-death cases depending on the specific recording date). The examinations and diagnoses from two expert physicians are independent. Because the health conditions of patients varied, each patient might have a different number of recorded data sessions at the same or different day. Finally, a total of 122 sessions' recordings from 58 patients were stored in the computer.

2.2. Data pre-processing method

Intensive care units generally exist a lot of electromagnetic environmental noise due to the operation of electronic devices. We can usually remove this part of the noise by filters. Due to the weak activity of physiological brain electrical signals in coma/brain-death patients, the frequency band of EEG is generally in the delta, theta rhythms, and a small number of patients will have EEG components in the alpha rhythms, and a very few will have components in the beta rhythms caused by prescribed sedative. Based on the frequency domain characteristics of EEG in coma/brain-death patients, we chose to filter out signals above 40 Hz. In this way we can filter out the electromagnetic environmental noise above 40 Hz, as well as some of the EEG signals that may be present but do not have a great impact on the EEG signal classification. All EEG signals were digitally band-pass filtered between 0.1 and 40 Hz using a fourth-order Butterworth filter. Forward and backward filtering was used to minimize phase distortions. Figure 5 shows an example of the coma EEG signal mixed with electromagnetic

environmental noise, and Figure 6 shows the EEG signals filtered by the band-pass filter.

Generally, EEG signal with high sampling frequency is down-sampled in the pre-processing stage to minimize the computation cost. In order not to reduce the variations or missing EEG features that may be caused by down-sampling, we try to train the neural network with the original sampling frequency EEG signals. In the subsequent convolution and feature extraction stages of the neural network, the structure called Pooling Layer in the network will down-sample the convolutional processed data.

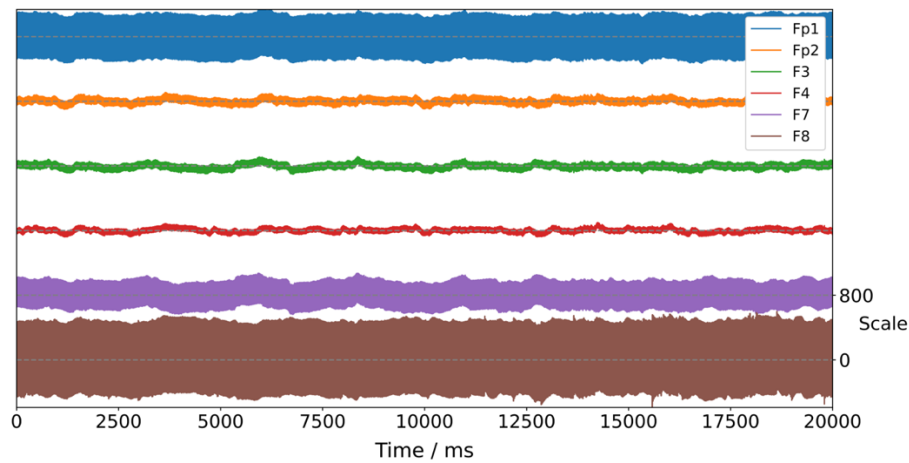


Fig. 5. An example of the coma EEG signal mixed with electromagnetic environmental noise.

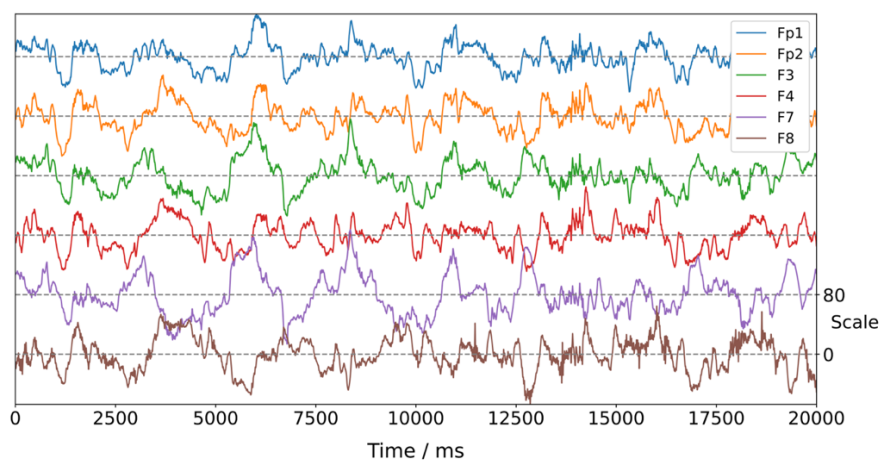


Fig. 6. The EEG signals filtered by the band-pass filter.

Sedation, hypothermia, toxic states, metabolic disorders, etc., may confound the test result of electrophysiological function. Some patients are prescribed sedative

at some point in the EEG acquisition process, which makes the acquired EEG signals show distinctive features (generally continuous high amplitude signals). Some other patients may have metabolic disorders because of the brain disease, thus affected the amplitude of the EEG signals. This part of EEG signals is characterized by high amplitude noise lasting 20-30 seconds (most of the signals exceed $100\mu V$). For this part of EEG signal containing noise with obvious high amplitude characteristics, we set a threshold for the amplitude ($n = 75$), and signals with a maximum amplitude greater than this threshold are considered to contain information that may confound the diagnosis of the brain-death and are rejected. Fig. 7. provides an example of the coma EEG signals with high amplitude noise.

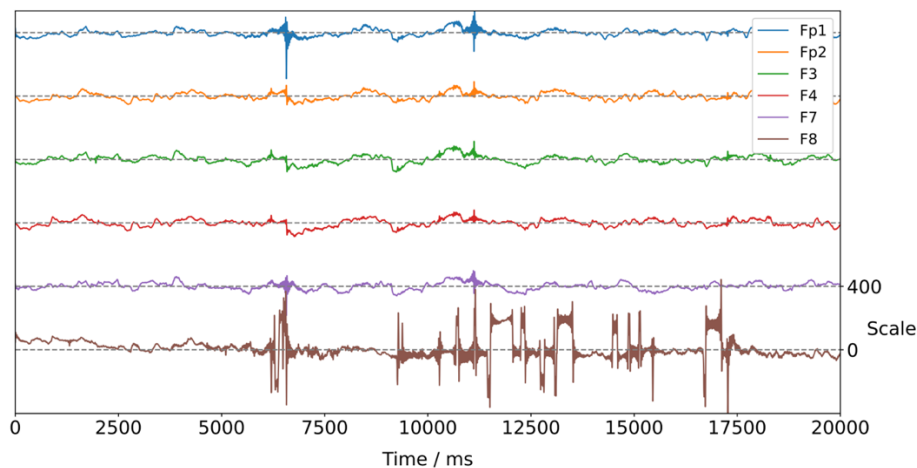


Fig. 7. An example of the coma EEG signals with high amplitude noise.

Some signals containing noise that could not be removed by the band-pass filter. Some others containing artifacts' amplitude (such as sedation etc.) that were below the threshold but would clearly interfere with the data. The low-frequency and low-amplitude noise always arise from some part of one or two patients' EEG recording. It is a short duration of time but recurrent. This kind of noise has a unique feature that the single low-frequency composition appeared repeatedly

in a short time. In this study, we reject this part of noise by visual inspection, and in the future, we could reject it automatically by the comparison of the absolute sum in different time window or the other methods. The red frame in the Fig. 8. shows the low-frequency and low-amplitude noise components that remain after the filter and threshold rejection. Some examples of the pre-processed coma EEG signals and brain-death EEG signals are shown in Figs. 9 and 10.

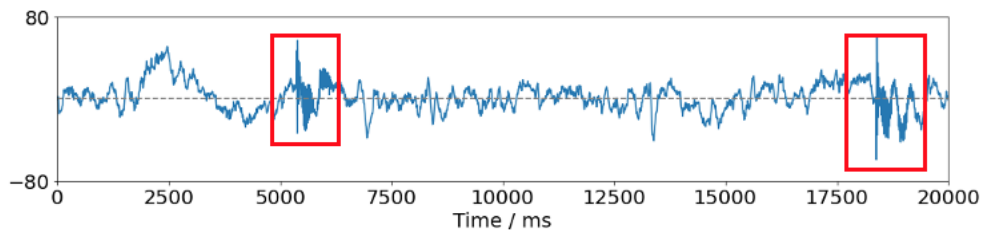


Fig. 8. An example of the coma EEG signals with low-frequency, low-amplitude noise components.

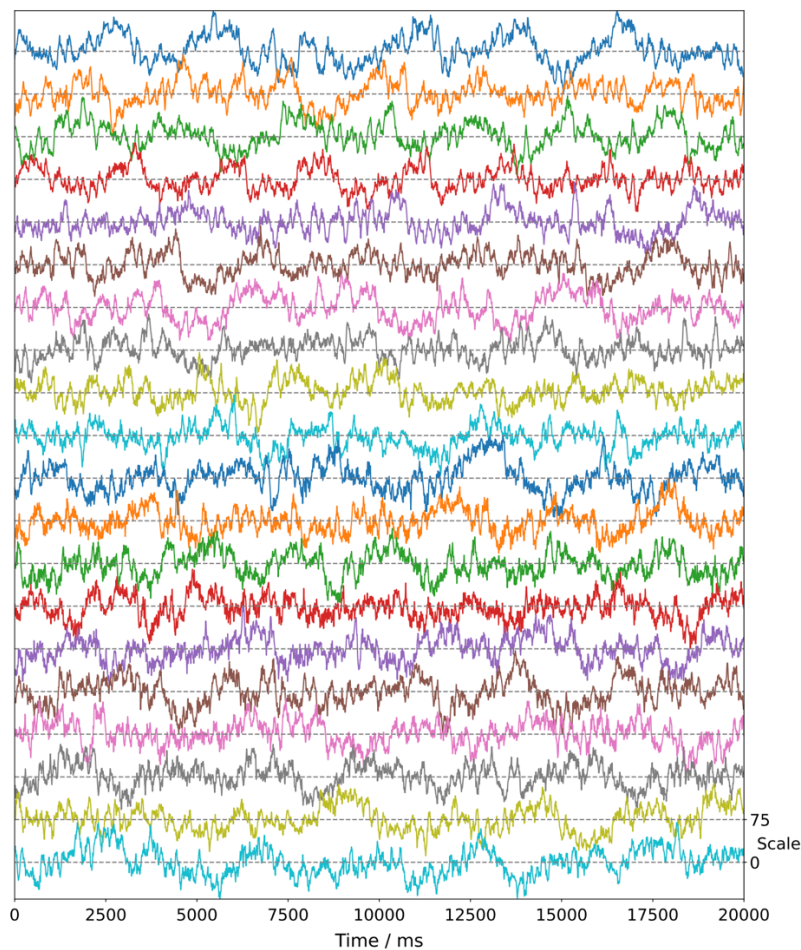


Fig. 9. An example of the pre-processed coma EEG signals.

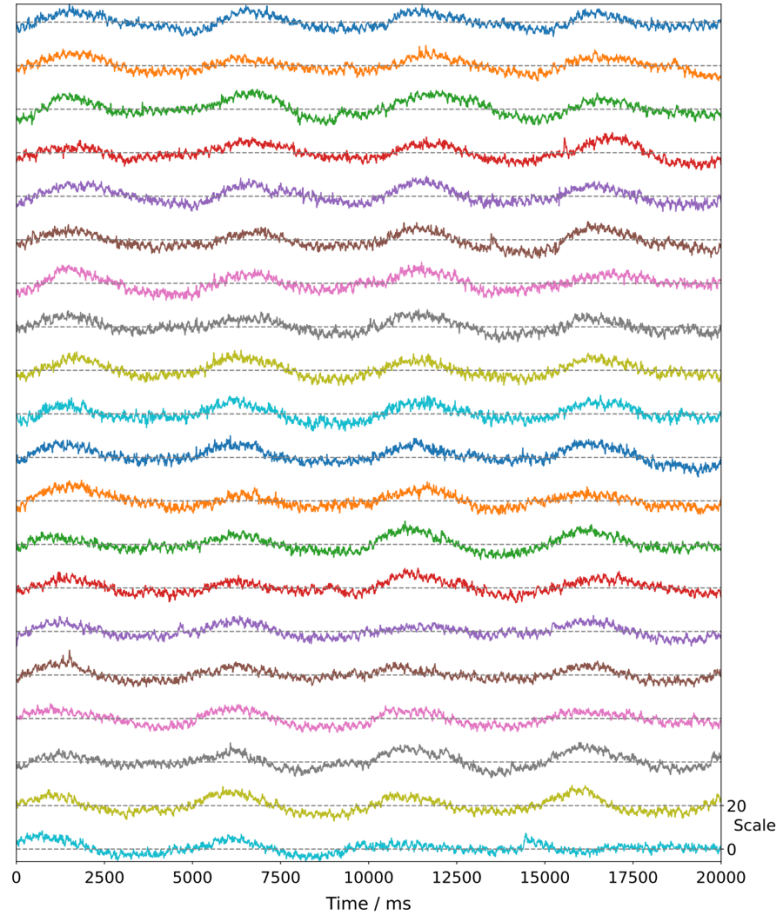


Fig. 10. An example of the pre-processed brain-death EEG signals.

2.3. The Proposed One-dimensional Convolutional Neural Networks (1D-CNN)

Convolutional Neural Network (CNN) as a class of Feedforward Neural Networks that include convolutional computation and deep structure, has artificial neurons which could respond to surrounding units within a portion of their coverage and perform well for large image processing. A CNN model consists of one or more convolutional layer and a top fully connected layer, and includes associative weights and pooling layers. This structure allows the CNN model to exploit the two-dimensional structure of the input data. Compared to other deep learning structures, the CNN model could provide better results in

image and speech recognition task. This model can also be trained by using back propagation algorithm. Compared to other deep, feed-forward neural networks, the CNN model requires less parameters to be considered. For one-dimensional data, such as single-channel EEG signals, one-dimensional convolutional neural networks (1D-CNN) also have a great advantage. Since the original one-dimensional EEG data does not have two-dimensional features in the structure, using 1D-CNN can directly extract the temporal feature information of one-dimensional data, which can greatly reduce the time consumed by the data in feature extraction, minimize the computational effort, and improve the computational efficiency. Fig. 11. shows the block representation of the proposed 1D-CNN model.

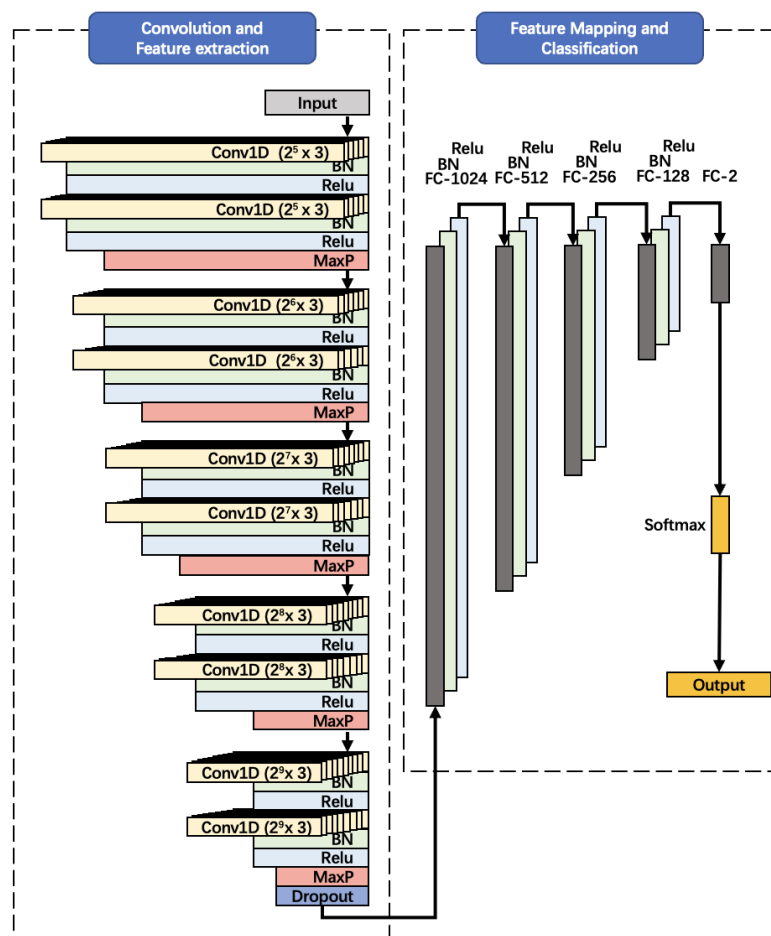


Fig. 11. Block representation of the proposed 1D-CNN model.

Each convolution filter of the convolution layer acts on the whole input signal and convolves the input signal. The result of convolution constitutes the feature map of the input signal, so that the local features of the signal are extracted after the convolution layer. We use convolutional kernels with a size of 3×1 in each layer to make the feature extraction stage will not have too much computation. In order not to miss any features, we set the stride of the convolutional kernel to 1. The number of filters is set to the N power of 2 ($5 \leq N \leq 9$).

The max pooling layer greatly reduces the neural network model's parameter amount and minimizes the computational effort and reduces the risk of overfitting the network to some extent. To avoid the loss of features in the signal, we do not down-sample the signal in the pre-processing stage. In the neural network model, we add a maximum pooling layer after every two convolutional layer processing to reduce the number of parameters of the network model, thus reducing the computational cost. We set the max-pooling layers which pool size as 4×1 and stride as 4 after every 2 convolutional layers.

Batch normalization layer [21] could accelerate the convergence rate of the neural network, effectively avoid vanishing gradient problem and improve the model generalization capability. We set the batch normalization layer after every convolutional layer and fully connected layer.

Relu activation function can make the training speed of neural network faster than sigmoid, tanh and other activation functions. Since it is a nonlinear function itself, adding it to the neural network can make the model fit the nonlinear mapping. We set the Relu activation function after every batch normalization layer.

Dropout layer [28] is used as a regularization method to prevent overfitting of

neural networks, which is of great help to the generalization capability of neural networks [10]. We set one dropout layer (Rate=0.5) before the first fully connect layer.

The fully connected layer acts as a classifier in the entire convolutional neural network. All the feature maps obtained from the dropout layer are flattened into a one-dimensional feature vector as the input of the first fully connected layer. To perform the classification process, we set the softmax layer as the last layer of the proposed 1D-CNN model. In this layer, input EEG signals are classified as coma signal or brain-death signal.

2.4. Experiment and Result

Fig. 12. shows the mean accuracy of train set and test set with 5-fold cross-validation. Fig. 13. shows the ROC (Receiver Operating Characteristic) curve of the test set. The summation of the confusion matrix obtained for the test data from this model in 5-fold cross-validation is shown in Tabel.1. With a 5-fold cross-validation, we obtained a mean accuracy of 100% of the training set and 99.71% of the test set. By calculating the AUC (Area Under the Curve) of the ROC, the value of the mean AUC of the model was obtained as 99.81%. In addition, we also obtained the recall score of 99.5% and the F1-score of 99.71% for the test set. The number of the total parameters in this model is 12,231,458, and the estimate total parameter size is 105.97MB. The time required for a 5-fold cross validation is about 3 hours. The model reached convergence at about the 125th epoch. We tried to use the raw coma/brain-death EEG signals directly to training and test the proposed 1D-CNN model with the same structure without the pre-processing mentioned previously. The results of the test are shown in the Table. 2. as a comparison.

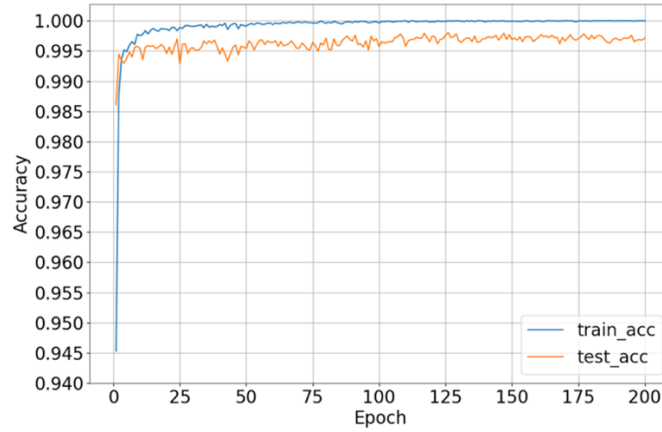


Fig. 12. The mean accuracy of the train set and the test set with 5-fold cross-validation.

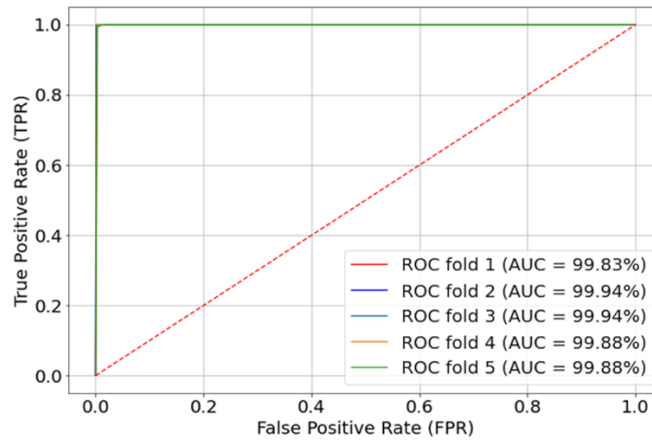


Fig. 13. The ROC curve of the test set with 5-fold cross-validation.

Table. 1. The summation of the confusion matrix obtained from this 1D-CNN model

Class	Pred Positive	Pred Negative
True Positive	TP = 4,346	FP = 4
True Negative	FN = 21	TN = 4,329

Table. 2. The performance measures of this 1D-CNN model

Preprocess	Classifier	Accuracy	TNR	Recall	Precision	F1-Score
No	1D-CNN	96.43	97.73	95.19	97.61	96.38
Yes	1D-CNN	99.71	99.91	99.51	99.52	99.71

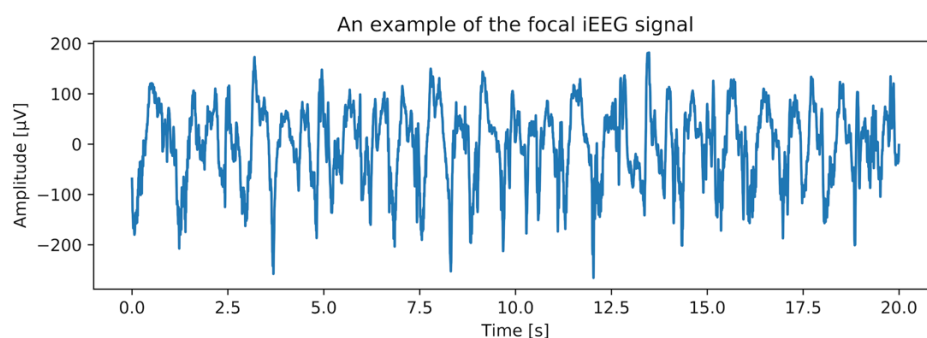
3. The Automated Localization of Epileptic Foci

3.1. The Bern-Barcelona iEEG Database

The iEEG signals from the Bern-Barcelona database provided by Andrzejak et al. at the Department of Neurology of the University of Bern, were obtained from the recordings of five epilepsy patients with focal epilepsy [3]. All patients suffered from long-standing pharmaco-resistant temporal lobe epilepsy and were candidates for epilepsy surgery patients. According to whether the signals were obtained from the focal channels, the dataset is divided into two categories. Each category contains 3,750 pairs sample with a duration of the 20 s sampled at a frequency of 512 Hz rendering 10,240 data points per sample. The dataset was processed by digitally band-pass filtered between 0.5 and 150 Hz by using a fourth-order Butterworth filter. All of the signals were labeled as focal signals or non-focal signals by clinical experts. The iEEG signals that during seizure activity and three hours after the last seizure were excluded.

An example of focal and non-focal iEEG signals is shown in Fig. 14, respectively.

The various attributes of the Bern-Barcelona dataset are provided in Tabel. 3.



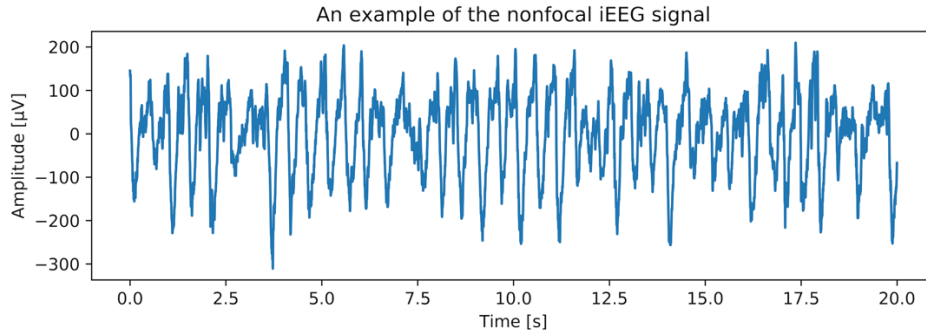


Fig. 14. An example of focal and non-focal epileptic iEEG signals.

Table. 3. Various attributes of the Bern-Barcelona dataset

No.	Attributes	Values
1	Dataset shape	15,000*10,240
2	The number of focal signals	3750*2
3	The number of non-focal signals	3750*2
4	Sampling time	20 s
5	Sampling frequency	512 Hz
6	Frequency band	0.5-150 Hz

3.2. The Proposed One-dimensional Convolutional Neural Networks (1D-CNN)

Developed one-dimensional convolutional neural network (1D-CNN) model was used for the classification of focal and non-focal iEEG signals in this study. This model allows the raw iEEG signals to be entirely classified directly without any feature extraction stage.

Five different types of layers are used in the developed 1D-CNN model: convolutional layer, pooling layer, fully connected layer, dropout layer, and batch normalization layer. Fig. 15. shows the details and output shape of every layer of the developed 1D-CNN model.

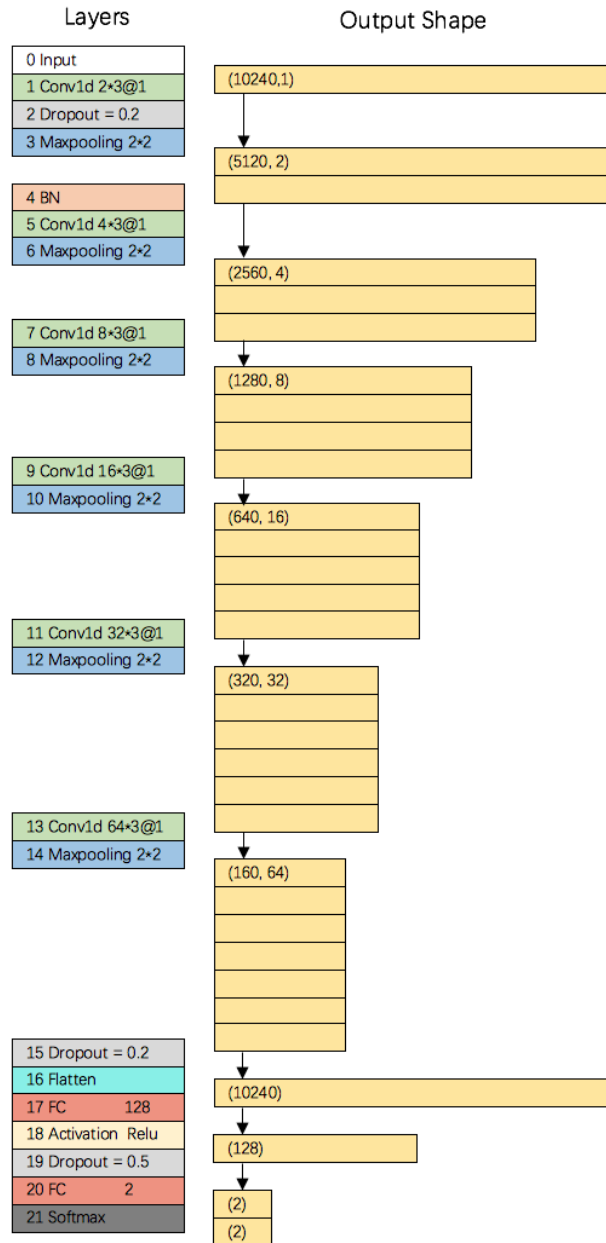


Fig. 15. Block representation of the developed 1D-CNN model.

We use convolutional kernels with a size of 3×1 in each layer to make the feature extraction stage will not have too much computation. In order not to miss any features, we set the stride of the convolutional kernel to 1. Relu activation function is used in all the convolutional layers. The number of filters is set to the N power of 2 ($N \leq 6$). We set the max-pooling layers which pool size as 2×1 and stride as 2 after every convolutional layer so that we could reduce the computation of the whole model and won't miss the pivotal features. All the

feature maps obtained from layer 16 are flattened into a one-dimensional feature vector as the input of layer 17. The output obtained from the first fully connected layer will be nonlinear by using Relu activation function and dropout with a rate of 0.5 and then as the input of layer 20. To perform the classification process, we set the softmax layer as the last layer of the developed 1D-CNN model. In this layer, input iEEG signals are classified as focal and non-focal.

One of the biggest challenges in this developed model is overfitting. We added the dropout layer [28] and Batch Normalization layer [21] in various positions to reduce the effect of overfitting. Batch Normalization layer in layer 4 performed the normalization for the output obtained from the upper max-pooling layer in this study.

3.3. Experiment and Result

The Bern-Barcelona iEEG dataset used in the study was recorded by Andrzejak et al. We split the iEEG database into three parts: train set (80%), validate set (10%) and test set (10%). The training dataset and the validation data were used during the learning stage, and test data was used during the testing stage. Thus, 12,000 out of a total of 15,000 samples were used for training, 1,500 were used for the validation, and the remaining 1,500 were used for the test. We used 10-fold cross-validation to ensure the results more dependable. A detailed illustration of the data sets used for this study is shown in Fig. 16.

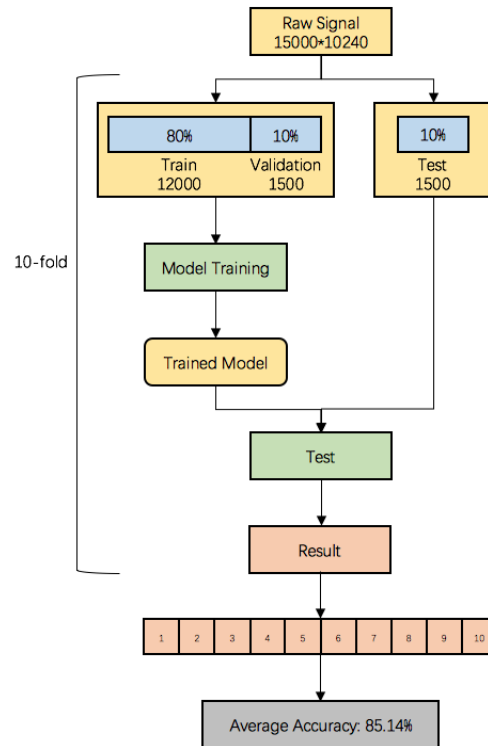


Fig. 16. The illustration of the data set used to develop this model.

A batch's size of 128 with a size of 10,240 are randomly fed into the network in each epoch of training. The performance graphs of the 1D-CNN model during the training and the validation are shown in Fig. 17.

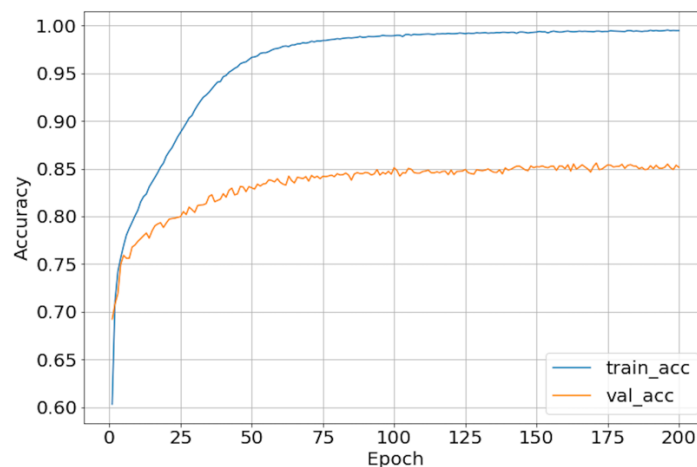


Fig. 17. Performance graphs of the model during the training and the validation.

It can be seen from the performance graphs that, there still has overfitting problem in the model although we've already used dropout layer and batch

normalization layer. We speculate that the representation of the Bern-Barcelona dataset on one-dimensional convolutional is not obvious. During the training phase of the model, the training accuracy is about 99%, and the validation accuracy is about 85%. In 10-fold cross-validation, the average validation accuracy is about 85.14%.

Some of the published works are recorded in Table.4. Although the developed model could not yield a great classification performance, it still managed to obtain 85.14% accuracy without any pre-processing before the training stage in this model. Compared with the other feature extraction methods such as entropy, DWT and EMD, the 1D-CNN model has further advantages. It is less computational, and extraction of one-dimensional subsequences from the signal with reduced the number of features.

Table. 4. Performance comparison of the developed model with other works on the same dataset

Authors	Feature Extraction	Classifier	Accuracy (%)
Sharma et al.	DWT, entropy	LS-SVM	84
Sharma et al.	EMD, entropy	LS-SVM	87
Chen et al.	DWT	SVM	83.07
Das et al.	EMD-DWT, entropy	KNN	89.4
Itakura	BEMD, entropy	SVM	86.89
Bhattacharyya	TQWT, entropy	LS-SVM	84.67
This model		1D-CNN	85.14

Various evaluation criteria have been selected for the test data. The summation of the confusion matrix obtained for the test data from this model in 10-fold cross-validation is shown in Tabel. 5. and the performance measures of this 1D-CNN model are shown in Tabel. 6. From the confusion matrix table, we could

know that the developed 1D-CNN model classified the test iEEG signals with a sensitivity (TPR) ratio of 88.76% and specificity (TNR) ratio of 81.68%.

Table. 5. The confusion matrix obtained from this 1D-CNN model.

Class	Focal	Non-focal
Focal	TP = 6,259	FP = 1,404
Non-focal	FN = 825	TN = 6,512

Table. 6. The performance measures of this 1D-CNN model.

Class	TPR	TNR	FPR	FNR	Precision	F1-Score
Ratio (%)	88.76	81.68	18.32	11.24	82.26	85.39

The Receiver Operating Characteristic Curve (ROC Curve) and the Area Under the Curve (AUC) of this 1D-CNN model is shown in Fig. 18. The value of the AUC has reached 92.17%, which means the developed model could make a great classification of the test data.

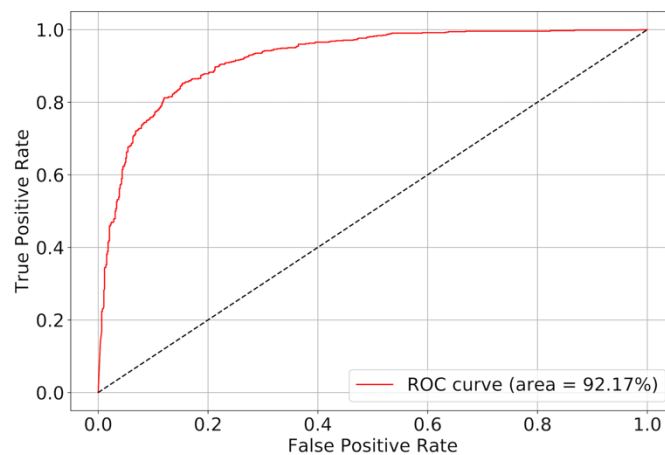


Fig. 18. The Receiver Operating Characteristic Curve of this 1D-CNN model.

4. Construct of High-Precision BCI System

4.1. P300 Visual Stimulator

The primary focus within our designed BCI system lies in the design and realization of the P300 visual stimulator [44, 45]. Various forms of P300 stimulators exist, encompassing visual, auditory, and tactile stimulators [13]. Due to their intuitive and easy-to-manipulate nature, visual stimulators have found widespread application in practice [14]. In this study, we developed a P300 visual stimulator based on a 3x3 grid pattern.

Unlike the traditional row-column P300 visual stimulator, our 3x3 grid pattern stimulator utilizes nine graphical units, each bearing a white circular image serving as the stimulus signal. For five control commands, the white circular image appears and disappears in five units in one epoch. Any instance of the white circular image appearing in a specific unit, in contrast to the image in other units at other time points, is considered a rare stimulus. The interval between two successive stimuli can be freely set, generally required to be at least 400ms but adjustable according to the individual differences of subjects. The graphical interface of the P300 visual stimulator we designed is shown in Fig. 19.

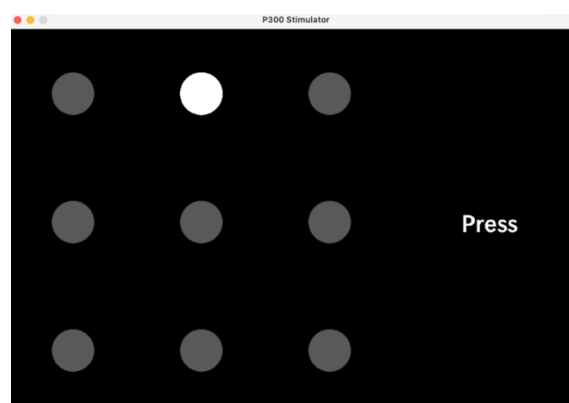


Fig. 19. The graphical interface of the P300 visual stimulator.

The P300 visual stimulator we designed sends EEG data through the Lab Streaming Layer (LSL) encompassing three columns. The first column is time stamps in units of $1/256$ seconds, the second column represents the stimulus labels corresponding to the time stamps, and the third column indicates the moment at which the subject presses the space bar when their chosen stimulus signal appears, defaulting to 0 if no key is pressed. By default, our P300 visual stimulator sends data once per second, with data shape as (3, 256). This setting ensures the provision of stimulus signals to the subject in a steady and continuous manner within the unit time. The sending frequency of this stimulator can be flexibly adjusted to suit different experimental needs and the reaction speeds of subjects. This structured data design facilitates subsequent data processing and analysis, aiding in the precise capture and understanding of subjects' responses to various stimulus signals. An advantage of our designed stimulator is the capability of customizing the flashing frequency of the stimuli according to the needs of the subjects. More importantly, while ensuring the effective stimulation, subjects are less likely to feel fatigued, significantly enhancing the user experience in the experiment.

4.2. Support Vector Machine

In our constructed P300-based robot movement control BCI system, we initially verified the effective capture of P300 signals using EEG signals collected with the Muse EEG equipment headset in offline experiments. Subsequently, we employed SVM as the classifier to train and test our obtained EEG dataset. The experimental results showed that the trained SVM classifier exhibited good performance in real-time testing during online experiments. These experimental results provided empirical support for the superiority of SVM in BCI research

and application and offered useful references for further development and optimization of supervised learning algorithms in BCI systems.

4.3. P300-based Brain-Computer Interface System

In this study, we developed a P300-based BCI system for real-time robot control. The core parts of this system include a self-designed 3x3 grid pattern visual stimulator, a Muse EEG equipment, a Nao robot as the controlled object, and an offline trained SVM classifier.

Our visual stimulator was developed using the Pygame package in the Python programming language, adopting a 3x3 layout design and independently adjustable stimulus flashing frequency to meet different experimental needs. To synchronize event markers and timestamps, we adopted the LSL protocol. This protocol not only recorded labels and timestamps of the stimulator but also precisely captured the time points of the subject's pressing keys, and these data were transmitted in real-time to the signal processing script. The Nao robot we used in the experiment, developed by Aldebaran Robotics in France, is an autonomous walking, highly interactive robot capable of facial and object recognition. It is widely used in education, research, and medical rehabilitation fields. In our experiment, the Nao robot performed corresponding movements upon receiving control commands corresponding to positional information.

The Muse EEG equipment we selected, with a sampling frequency of 256Hz, served as the EEG signal acquisition device, transmitting real-time collected EEG signals to the signal processing script via Bluetooth wireless. In the signal processing part, we synchronized the timestamps when subjects pressed keys with corresponding labels through the LSL protocol. When the number of key presses reached a predetermined threshold, the script automatically extracted all

1-second-long EEG signals corresponding to the timestamps, followed by filtering and arithmetic mean operations. Subsequently, we used by the pre-trained SVM classifier to classify the pre-processed signals, differentiating them into 0 (no P300 component) and 1 (P300 component present). When the classification result was 1, the script located on the flashing label of the stimulator signal corresponding to the subject's keypress event, determined the position of the target signal, and then translated this positional information into a control command for the Nao robot. Fig. 20. shows the block diagram of the designed P300-Based BCI system.

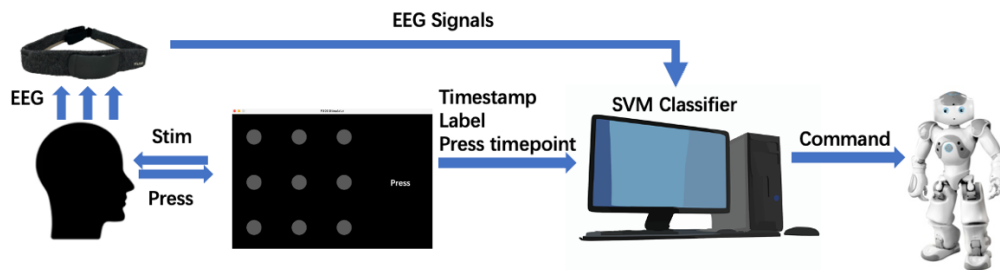


Fig. 20. The block diagram of the designed P300-Based BCI system.

4.4. Experiment and Result

In P300-Based BCI systems, there are generally some basic assumptions and constraints. Firstly, the subject is required to maintain sustained concentration, as the generation of P300 signals necessitates that the subjects remain attentively focused on visual stimuli [7]. If the subject becomes distracted or fatigued, the resulting P300 may be diminished, thereby affecting the performance of the system. Secondly, the variety of P300 signals presents a challenge. The amplitude and latency of P300 signals may vary among individuals, and even within the same individual at different times. This variability may impact the training and performance of the classifier [15]. Finally, the acquisition of training

data is important. Due to the high variability of P300, the classifier may require a large amount of training data to achieve optimal performance [15].

In consideration of these assumptions and constraints, we took a series of measures to optimize our experimental design. During the experiments, we asked the subjects to rest after a certain number of trials to ensure that they maintain sufficient attention throughout the experiment. To ensure the consistency of the experimental environment, we chose to conduct the experiment in a relatively dark, noise-free environment. We also asked the subjects to clean the electrodes of the Muse EEG equipment and the area of skin in contact with the electrodes using alcohol wipes before each experiment to minimize noise and interference. Our experimental design is mainly divided into offline and online stages. In the offline stage, our primary task is to verify whether the Muse EEG equipment can effectively collect EEG signals containing P300 components. We had the subjects focus on target stimuli on the visual stimulator, then carried out pre-processing and feature extraction, and trained the SVM classifier, performing parameter selection and optimization at this stage. The results of the offline stage provide an important premise and the basis for the subsequent online stage. Once in the online stage, we utilized the SVM classifier trained in the offline stage to classify the real-time EEG signals acquired from the Muse EEG equipment. In this stage, we simulated actual application scenarios, requiring the subjects to generate EEG signals containing P300 components by focusing on the target stimuli on the visual stimulator, then implemented real-time control of the Nao robot based on the output of the classifier. The results of the online stage directly demonstrate the system's performance in practical applications. Fig. 21. shows the flow chart of the experiment.

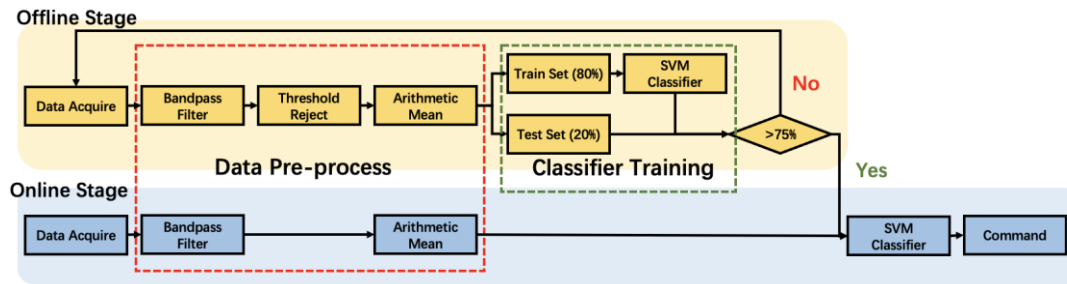


Fig. 21. The flow chart of the offline and online experiment.

4.4.1. Offline Stage

Step 1 The subjects cleaned their foreheads, the edge part of their ears, and the electrodes of the Muse EEG equipment using alcohol wipes. The subjects then wore the Muse EEG equipment, and the current EEG readings from the electrodes were confirmed through an oscilloscope program to ensure that the collected EEGs contained minimal or no noise components.

Step 2 The stimulator was run, and the space key was randomly pressed to test whether the stimulator could receive real-time hit times.

Step 3 The EEG recording code was executed. Taking a recording duration of 60s as an example, after running the EEG recording code, the subject immediately looked at the stimulator. The flashing interval of the stimulator was set to 400ms, and the duration of the stimulus block was 100ms. After selecting a specific stimulus unit, the subject pressed the space key each time the stimulus appeared and was asked to count, acquiring EEGs containing P300 components. After the EEG recording time was reached, the recording was completed.

Step 4 Step 3 was run to collect n pieces EEG signals of length 256.

Step 5 The EEG signal processing code was executed. The collected EEGs were averaged out using the arithmetic mean by using the TP9 and TP10 channel

EEG data, converted to one-dimensional data. The resulting data was passed through a 0.5-35Hz band-pass filter and reshaped into n sets of data of length 256. Data with a maximum amplitude exceeding 80 were removed, resulting in m sets of data.

Step 6 The obtained m sets of EEG signals containing P300 components were averaged out using the arithmetic mean. Fig. 22. shows the P300 component of the EEG signals after using the arithmetic mean.

Step 7 The m sets of data were randomly shuffled and averaged in sets of 4, resulting in k sets of EEG data containing P300 components.

Step 8 Steps 2-4 were run, g sets of EEG data of length 256 were randomly collected, the subject did not look at the stimulator, and pressed the space key randomly, obtaining EEGs without P300 components. Step 5 was run to obtain h sets of arithmetic mean EEG data without P300 components.

The obtained k sets of averaged EEG data with P300 components and h sets of averaged EEG data without P300 components were combined and randomly shuffled, and the shuffled data was split into an 80:20 ratio. 80% was used as training data input into the SVM, yielding the SVM classifier. 20% was used as test data to test the accuracy of the obtained SVM classifier. When the classifier's accuracy reached above 75%, the obtained SVM classifier was used in the online experiment for real-time classification of EEG signals.

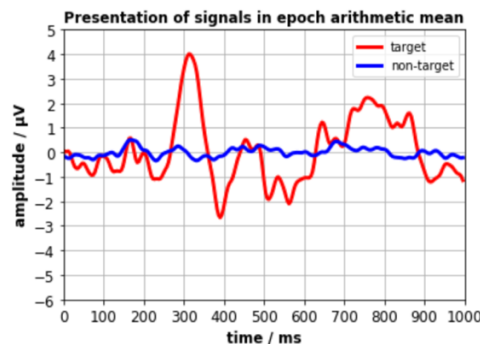


Fig. 22. The P300 component of the EEG signals after using the arithmetic mean.

4.4.2. Offline Stage Results

In the offline stage of the P300-based robot control BCI system experiment, we validated the conclusion that the Muse EEG equipment can capture P300 components in the subject's EEGs. We collected 2173 pieces of EEG signals, each of length 256, containing P300 components from 3 subjects. After filtering and reject high-amplitude noise, the data were randomly shuffled, and 1600 pieces of signals were selected. The data were then subjected to arithmetic mean every 4 signals, converting them into a single one-dimensional data sequence of length 256, yielding a total of 400 pieces of signals. Non-target signals were processed in the same way, producing 400 pieces of non-target signals. Both categories of data were merged and shuffled, with 80% selected for training the SVM classifier, and the remaining 20% used as a test set to assess the performance of the SVM classifier. The test set consisted of 160 signals, of which 133 were successfully classified, resulting in a classifier accuracy rate of 84.1%. As shown in Figure 6, after 1600 pieces of target EEG signals and 1600 pieces of non-target EEG signals averaged out using the arithmetic mean, we can see that the noise components in the signal are suppressed, and a clear positive wave appears at 300ms, proving the existence of P300 components.

4.4.3. Online Stage

Step 1 The subjects cleaned their foreheads, the edge part of their ears, and the electrodes of the Muse EEG equipment using alcohol wipes. The subjects then wore the Muse EEG equipment, and the current EEG readings from the electrodes were confirmed through an oscilloscope program to ensure that the collected EEGs contained minimal or no noise components.

- Step 2 The stimulator was run, and the space key was randomly pressed to test whether the stimulator could receive real-time hit times.
- Step 3 The online P300-based BCI robot control program was run. By default, the experiment considered 4 or more space key presses as a command. That is, when the subjects visually targeted the stimulator, they selected one stimulus unit, and each time this stimulus unit displayed a white circular image, they pressed the space key. After pressing, a command was generated.
- Step 4 After 4 or more keypress, the program filtered and averaged the EEGs corresponding to the keypress times, and the resulting EEG signal was transmitted to the SVM classifier.
- Step 5 The SVM classifier performed classification calculations on the received EEG signals. If the classification result was 1, indicating that the transmitted EEGs contained P300 components, the experiment was deemed successful. Using timestamps, the stimulator label corresponding to the keypress was identified, obtaining the position of the stimulus unit that the subject was visually targeting at the time of the keypress, i.e., the subject's command. This command was then transmitted to the Nao robot, realizing EEG-controlled robot operation.

4.4.4. Online Stage Results

We invited 3 subjects to participate in a total of 500 trials. 406 trials were classified successfully. The control commands for the robot given by the subjects were random, but they were asked to ensure that the quantity of each command was as balanced as possible. The online experiment accuracy rate of the 3 subjects reached 81.2%.

5. Conclusion and Future Work

As a record of electrical activity of the scalp, the clinical interpretation of EEG signals can accurately determine the state of brain activity of patients. At the same time, EEG signals are also commonly used in the construction of BCI systems. EEG signals are acquired at specific locations through different stimulation or evocation patterns, analyzed and classified, and used to control external devices such as robots.

However, EEG signals are often mixed with a large amount of noise due to the environment, human factors, and the patient's state. The presence of these noises brings great trouble to the clinical diagnosis of doctors, and at the same time, it also has a great impact on the signal parsing and categorization tasks of the BCI system. Therefore, this paper proposes an EEG signal pre-processing model based on the clinical interpretation criteria of EEG signals, considering the characteristics of the EEG signals, the state of the patient and the state of the brainwave instrumentation equipment.

Removing temporarily the respiratory machine during the apnea test of brain-death clinical examination may cause patients irreversible loss of brain function or even direct lead to death. Hence, there is an urgent need for a technique which could automatically identify brain-death signals rapidly and accurately. As a noninvasive and rapid electrical activity examination, EEG could quickly provide physicians with cerebral cortex energy analysis and diagnosis recommendations.

We explore the feasibility of classify coma/brain-death signals in the field of deep learning and propose a pre-process method of the Brain-Death EEG Database. Subsequently, we propose a 1D-CNN model for the classification task

of the pre-processed Brain-Death EEG Database. With a 50:50 ratio of targets to non-targets, the accuracy of the classification reaches 99.71% and the recall score reaches 99.51%, meaning that the 1D-CNN we proposed perform well for the classification task of the pre-processed coma/brain-death signals. As a measure of a test' accuracy, the F1-score of the proposed model reaches 99.71%, and the value of the AUC reaches 99.89%, which means the proposed model could make a great classification of the test data. Compared with the experiment without signal pre-processing, the performance of the 1D-CNN model has improved significantly.

It is a challenging task to distinguish the focal channels by iEEG signals in interictal. As the occurrence of seizures causes brain damage, the accurate detection of focal location could aid the clinical experts to validate their screening of iEEG signals and provide proper treatment to the patients earlier. We developed a 1D-CNN model to automate detect the epilepsy focal signals in this study. Our developed model can detect the focal signals with an accuracy of 85.14% by using raw signals without any pre-processing. Compared with the other methods, computational reduced significantly, which means the training time reduced greatly. We intend to optimize our model by some methods such as data augmentation to increase the test accuracy in the future.

We constructed a P300-Based BCI system, designed and implemented a 3x3 grid pattern P300 visual stimulator, wore the Muse EEG equipment to acquire user's EEG signals, and classified the EEG data using a SVM classifier to ultimately achieve control over robot movement. Our experiment comprised both offline and online stages. The offline stage was primarily for training the SVM classifier and verifying its efficacy in identifying EEG signals containing P300 stimulus components. The online stage sought to validate the system in actual operation.

After analyzing the offline experiment, we found that the SVM classifier exhibited well accuracy in identifying EEG signals containing P300 components. The online experiment demonstrated that the real-time success rate of the system in controlling robot actions could exceed 81.2%, which showcases the feasibility and efficacy of our system in practical application.

In future work, we will try to preprocess the Bern-Barcelona iEEG dataset using the proposed signal pre-processing model. In the standard of clinical interpretation of iEEG signals, the effective frequency domain of iEEG signals is 0.5-200 Hz. iEEG signals, as non-stationary signals, cannot be directly extracted using methods such as Fourier transform directly. We will try to decompose the iEEG signal using signal processing methods such as 2T-EMD. The series of IMFs obtained after decomposition can be viewed as an approximate stationary signal in the frequency domain. At this point, the signal is subjected to traditional methods such as Fourier transform to obtain the frequency domain features of the signal, and then machine learning methods are used to classify the frequency domain features, and it is believed that better results can be obtained. At the same time, we will try to use signal pre-processing methods as well as machine learning methods to acquire EEG signals under different states of consciousness and analyze the EEG signals. Human consciousness states are roughly classified into awakening, sleep, anesthesia, coma, and brain-death. In different states of consciousness, human brain activity states should be different. In clinical examination, the points of the patient are calculated by Glasgow Coma Scale (GCS), and then the state of the patient is judged. In future studies, we will use pre-processing methods as well as machine learning methods to classify the different consciousness levels of the human brain by quantifying the properties of EEG signals.

Reference

- [1] Ahmed MU, Li L, Cao J, Mandic DP: "Multivariate multiscale entropy for brain consciousness analysis." 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp 810–813, DOI: <https://doi.org/10.1103/PhysRevE.64.061907>
- [2] Andrzejak RG, Lehnertz K, Mormann F, Rieke C, David P, Elger CE: "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: dependence on recording region and brain state." *Physical Review E*, 64(6):061907, DOI: <https://doi.org/10.1103/PhysRevE.64.061907>
- [3] Andrzejak RG, Schindler K, Rummel C: "Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients." *Physical Review E*, 86(4):046206, DOI: <https://doi.org/10.1103/PhysRevE.86.046206>
- [4] Cao J: "Analysis of the quasi-brain-death EEG data based on a robust ICA approach." *International conference on knowledge-based and intelligent information and engineering systems*, pp 1240–1247. Springer, DOI: https://doi.org/10.1007/11893011_157
- [5] Cao J, Chen Z: "Advanced EEG signal processing in brain death diagnosis." *Signal Processing Techniques for Knowledge Extraction and Information Fusion*, pp 275–298. Springer, DOI: https://doi.org/10.1007/978-0-387-74367-7_15
- [6] Cao Jianting, Murata Noboru, Amari Shun-ichi, Cichocki Andrzej, Takeda Tsunehiro: "A robust approach to independent component analysis of signals with high-level noise measurements." *IEEE Trans Neural Networks*, 14(3):631–645, DOI: <https://doi.org/10.1109/TNN.2002.806648>
- [7] Chen Q, Yuan L, Miao Y, Zhao Q, Tanaka T, Cao J: "Quasi-brain-death EEG diagnosis based on tensor train decomposition." In: *International symposium on neural networks*, pp 501–511. Springer, DOI: https://doi.org/10.1007/978-3-030-22808-8_49
- [8] Eldele E, Chen Z, Liu C, Min W, Kwok CK, Li X, Guan C: "An attention-based deep learning approach for sleep stage classification with single-channel EEG." *IEEE Trans Neur Sys Rehabil Eng* 29:809–818, DOI: <https://doi.org/10.1109/TNSRE.2021.3076234>
- [9] Greer DM, Shemie SD, Lewis A, Torrance S, Varelas P, Goldenberg FD, Bernat JL, Souter M, Topcuoglu MA, Alexandrov AW et al (2020): "Determination of brain death/death by neurologic criteria: the world brain death project." *Jama* 324(11):1078–1097, DOI: <https://doi.org/10.1001/jama.2020.11586>

- [10] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012): "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580, DOI: <https://doi.org/10.48550/arXiv.1207.0580>
- [11] Li B, Zhao X, Zhao Q, Tanaka T, Cao J (2019): "A one-dimensional convolutional neural network model for automated localization of epileptic foci." In: 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC) IEEE, pages 741–744, DOI: <https://doi.org/10.1109/AP...>
- [12] Li L, Xia Y, Jelfs B, Cao J, Mandic DP (2012): "Modelling of brain consciousness based on collaborative adaptive filters." *Neurocomputing* 76(1):36–43, DOI: <https://doi.org/10.1103/PhysRevE.64.061907>
- [13] Liu S, Liu S, Cai W, Pujol S, Kikinis R, Feng D (2014): "Early diagnosis of Alzheimer's disease with deep learning." In: 2014 IEEE 11th international symposium on biomedical imaging (ISBI), IEEE, pp 1015–1018, DOI: <https://doi.org/10.1109/ISBI.2014.6868045>
- [14] Shi Q, Yang J, Cao J, Tanaka T, Wang R, Zhu H (2011): "EEG data analysis based on EMD for coma and quasi-brain-death patients." *J Exper Theoret Artif Intell* 23(1):97–110, DOI: <https://doi.org/10.1103/PhysRevE.64.061907>
- [15] Shoeb AH (2009): "Application of machine learning to epileptic seizure onset detection and treatment." PhD thesis, Massachusetts Institute of Technology
- [16] Yin Y, Cao J, Shi Q, Mandic D, Tanaka T, Wang R (2011): "Analyzing the EEG energy of quasi brain death using memd." In: Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference
- [17] Zhao X, Zhao Q, Tanaka T, Sole´-Casals J, Zhou G, Mitsuhashi T, Sugano H, Yoshida N, Cao J (2022): "Classification of the epileptic seizure onset zone based on partial annotation." *Cognit Neurodyn*. DOI: <https://doi.org/10.1007/s11571-022-09857-4>
- [18] Abhijit Bhattacharyya, Ram Pachori, and U Acharya: "Tunable-Qwavelet transform based multivariate sub-band fuzzy entropy with application to focal EEG signal analysis." *Entropy*, 19(3):99, 2017.
- [19] Duo Chen, Suiren Wan, and Forrest Sheng Bao: "Epileptic focus localization using EEG based on discrete wavelet transform through full-level decomposition." In 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP), pages 1–6. IEEE, 2015.
- [20] Anindya Bijoy Das and Mohammed Imamul Hassan Bhuiyan: "Discrimination and classification of focal and non-focal EEG signals using

- entropy-based features in the EMD-DWT domain." *Biomedical Signal Processing and Control*, 29:11–21, 2016.
- [21] Sergey Ioffe and Christian Szegedy: "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167, 2015.
- [22] Hakan Isjik and Esmâ Sezer: "Diagnosis of epilepsy from electroencephalography signals using multilayer perceptron and elman artificial neural networks and wavelet transform." *Journal of Medical Systems*, 36(1):1–13, 2012.
- [23] Tatsunori Itakura and Toshihisa Tanaka: "Epileptic focus localization based on bivariate empirical mode decomposition and entropy." In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1426–1429. IEEE, 2017.
- [24] World Healthy Organization: "Epilepsy." <https://www.who.int/news-room/fact-sheets/detail/epilepsy#>.
- [25] Subhrajit Roy, Isabell Kiral-Kornek, and Stefan Harrer: "ChronoNet: A deep recurrent neural network for abnormal EEG identification." arXiv preprint arXiv:1802.00308, 2018.
- [26] Rajeev Sharma, Ram Pachori, and U Acharya: "Application of entropy measures on intrinsic mode functions for the automated identification of focal electroencephalogram signals." *Entropy*, 17(2):669–691, 2015.
- [27] Rajeev Sharma, Ram Pachori, and U Acharya: "An integrated index for the identification of focal electroencephalogram signals using discrete wavelet transform and entropy measures." *Entropy*, 17(8):5218–5240, 2015.
- [28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov: "Dropout: a simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [29] Özal Yildirim, Ulas Baran Baloglu, and U Rajendra Acharya: "A deep convolutional neural network model for automated identification of abnormal EEG signals." *Neural Computing and Applications*, pages 1–12, 2018.
- [30] Özal Yildirim, Ru San Tan, and U Rajendra Acharya: "An efficient compression of ECG signals using deep convolutional autoencoders." *Cognitive Systems Research*, 52:198–211, 2018.
- [31] Xuyang Zhao, Qibin Zhao, Toshihisa Tanaka, Jianting Cao, Wanzeng Kong, Hidenori Sugano, and Noboru Yoshida: "Detection of epileptic foci based on interictal iEEG by using convolutional neural network." In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pages 1–5. IEEE, 2018.

- [32] Wolpaw, J. R., & Wolpaw, E. W. (Eds.) (2012): "Brain-computer interfaces: principles and practice." Oxford University Press.
- [33] Mak, J. N., & Wolpaw, J. R. (2009): "Clinical applications of brain-computer interfaces: current state and future prospects." *IEEE Reviews in Biomedical Engineering*, 2, 187-199.
- [34] Hochberg, L. R., Serruya, M. D., Friehs, G. M., Mukand, J. A., Saleh, M., Caplan, A. H., ... & Donoghue, J. P. (2006): "Neuronal ensemble control of prosthetic devices by a human with tetraplegia." *Nature*, 442(7099), 164-171.
- [35] Lécuyer, A., Lotte, F., Reilly, R. B., Leeb, R., Hirose, M., & Slater, M. (2008): "Brain-computer interfaces, virtual reality, and videogames." *Computer*, 41(10), 66-72.
- [36] Nicolas-Alonso, L. F., & Gomez-Gil, J. (2012): "Brain computer interfaces, a review." *Sensors*, 12(2), 1211-1279.
- [37] Allison, B. Z., Brunner, C., Kaiser, V., Müller-Putz, G. R., Neuper, C., & Pfurtscheller, G. (2010): "Toward a hybrid brain-computer interface based on imagined movement and visual attention." *Journal of neural engineering*, 7(2), 026007.
- [38] Polich, J. (2007): "Updating P300: an integrative theory of P3a and P3b." *Clinical neurophysiology*, 118(10), 2128-2148.
- [39] Chen, X., Wang, Y., Nakanishi, M., Gao, X., Jung, T. P., & Gao, S. (2015): "High-speed spelling with a noninvasive brain-computer interface." *Proceedings of the National Academy of Sciences*, 112(44), E6058-E6067.
- [40] Pfurtscheller, G., & Neuper, C. (2001): "Motor imagery and direct brain-computer communication." *Proceedings of the IEEE*, 89(7), 1123-1134.
- [41] Wolpaw, J. R., Birbaumer, N., Heetderks, W. J., McFarland, D. J., Peckham, P. H., Schalk, G., ... & Vaughan, T. M. (2002): "Brain-computer interface technology: a review of the second international meeting." *IEEE transactions on neural systems and rehabilitation engineering*, 10(2), 94-109.
- [42] Gramann, K., Gwin, J. T., Ferris, D. P., Oie, K., Jung, T. P., Lin, C. T., ... & Makeig, S. (2010): "Cognition in action: Imaging brain/body dynamics in mobile humans." *Reviews in the neurosciences*, 22(6), 593-608.
- [43] Ratti, E., Waninger, S., Berka, C., Ruffini, G., & Verma, A. (2017): "Comparison of medical and consumer wireless EEG systems for use in clinical trials." *Frontiers in human neuroscience*, 11, 398.
- [44] Krusienski, D. J., Sellers, E. W., Cabestaing, F., Bayouth, S., McFarland, D. J., Vaughan, T. M., & Wolpaw, J. R. (2008): "A comparison of classification techniques for the P300 Speller." *Journal of neural engineering*, 3(4), 299.
- [45] Martens, S. M., Hill, N. J., Farquhar, J., & Schölkopf, B. (2009): "Overlap and refractory effects in a brain-computer interface speller"

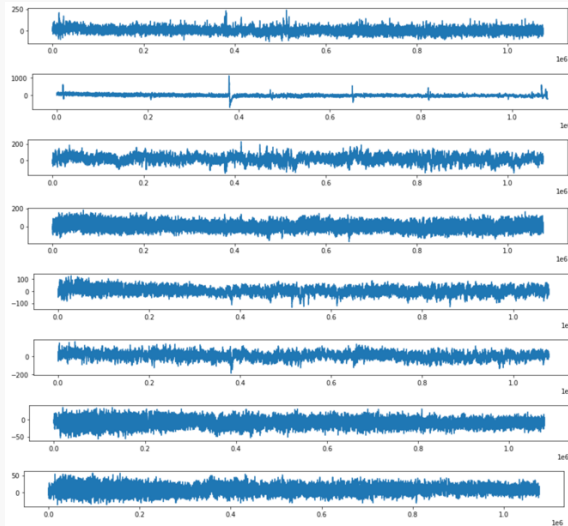
- [46] Lotte F, Bougrain L, Cichocki A, et al: "A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update." *Journal of neural engineering*, 2018, 15(3): 031005.
- [47] Norcia A M, Appelbaum L G, Ales J M, et al: "The steady-state visual evoked potential in vision research: A review." *Journal of vision*, 2015, 15(6): 4-4.
- [48] Vialatte, François-Benoît, et al. "Steady-state visually evoked potentials: focus on essential paradigms and future perspectives." *Progress in neurobiology*, 90(4): 418-438, 2010.
- [49] Harold Hotelling: "Relations between two sets of variates." In *Breakthroughs in statistics: methodology and distribution*, pages 162–190. Springer, 1992.
- [50] Hardoon, Szedmak, and Shawe-Taylor] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor: "Canonical correlation analysis: An overview with application to learning methods." *Neural computation*, 16 (12):2639–2664, 2004.
- [51] Guangyu Bin, Xiaorong Gao, Zheng Yan, Bo Hong, and Shangkai Gao: "An online multi-channel ssvp-based brain–computer interface using a canonical correlation analysis method." *Journal of neural engineering*, 6(4):046002, 2009.

Appendix

Study 1. Classification of Coma/Brain-Death EEG Dataset

Data pre-processing

```
1. import pickle
2. import numpy as np
3. import matplotlib.pyplot as plt
4. from scipy.fftpack import fft, ifft
5. from scipy.signal import butter, lfilter, lfilter_zi
6. from utils import *
7.
8. with open("hd_pkl/1.pkl", "rb") as file:
9.     data = pickle.load(file)
10.
11. data = data['a']
12. data.shape
13.
14. for i in range(len(data)):
15.     plt.figure(figsize=(15, 1))
16.     plt.plot(data[i])
17.     plt.show()
18.
19. data_filtered = butter_bandpass_filter(data, 0.1, 40, fs=1000, order=4)
20.
21. fft_ = fft(data_filtered[0])
22. abs_ = np.abs(fft_)
23. normalization_ = abs_/len(data_filtered[0])
```



```

24. normalization_half_ = normalization_[ :40]
25.
26. plt.figure(figsize=(8, 4))
27. plt.plot(normalization_half_)
28. plt.ylim(0, 2)
29. plt.grid()
30.
31. data_save = data_filtered[:6]
32.
33. slide_len = 20 # seconds
34. sfreq = 1000
35. seg_len = slide_len*sfreq
36. threshold = 75
37. reject_idx=[]
38.
39. data_len = len(data_save[0])
40. data_num = data_len//seg_len
41. data_ = np.reshape(data_save[:, :data_num*seg_len], (data_num*6, seg_len))
42.
43. for j in range(len(data_)):
44.     if max(abs(data_[j]))>threshold:
45.         reject_idx.append(j)
46.
47. data_rejected = np.delete(data_, reject_idx, axis=0)
48.
49. print(len(reject_idx))
50. print(np.shape(data_rejected))
51.
52. fft_ = fft(data_ori)
53. abs_ = np.abs(fft_)
54. normalization_ = abs_/len(data_ori)
55. normalization_half_ = normalization_[ :40]
56.
57. plt.figure(figsize=(8, 4))
58. plt.plot(normalization_half_)
59. plt.ylim(0, 2)
60. plt.grid()
61.
62. for i in range(len(data_rejected)):

```

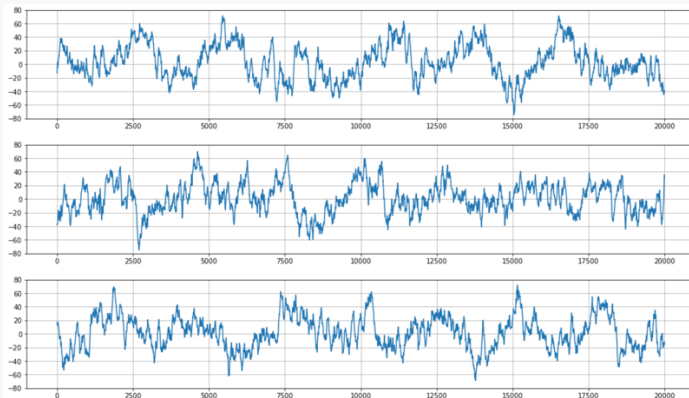
```
63. plt.figure(figsize=(18, 3))
```

```
64. plt.plot(data_rejected[i])
```

```
65. plt.ylim(-80, 80)
```

```
66. plt.grid()
```

```
67. plt.show()
```



```
68.
```

```
69. f=plt.figure(figsize=(16, 20), dpi=600);
```

```
70.
```

```
71. data_info={'data':data_rejected[:20], 'loc':['Fp1', 'Fp2', 'F3', 'F4', 'F7', 'F8']}
```

```
72. ax=plt.plot(data_info['data'].T + 75*np.arange(19,-1,-1));
```

```
73.
```

```
74. ax = f.add_subplot()
```

```
75. ax.yaxis.tick_right()
```

```
76. ax.yaxis.set_label_position("right")
```

```
77. ax.set_ylabel(' Scale', fontdict={'size': 20}, rotation=0, x=0, y=0.07)
```

```
78.
```

```
79. plt.plot(np.zeros((20001,20)) + 75*np.arange(19,-1,-1),'--',color='gray');
```

```
80. plt.xlabel("Time / ms", fontsize=20)
```

```
81. # plt.ylabel('Scale', fontsize=20)
```

```
82. # plt.xticks([0, 1, 2, 3, 4])
```

```
83. plt.xticks(fontsize=20)
```

```
84. plt.yticks([0, 75], fontsize=20);
```

```
85. # plt.fill_between(0, 80, color='blue', alpha=.25)
```

```
86. plt.margins(x=0)
```

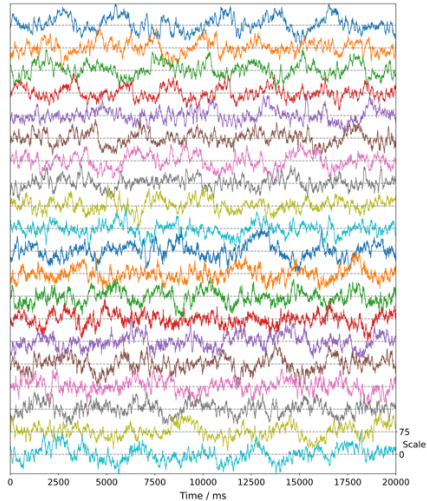
```
87. plt.margins(y=0)
```

```
88. plt.axis('tight');
```

```
89.
```

```
90. # plt.legend(data_info['loc'], fontsize=16, loc=1);
```

```
91. plt.savefig('pro-processed_dpi600.eps')
```



```

92.
93. normalization_rejected_ = np.empty((len(data_rejected), 40))
94. for i in range(len(data_rejected)):
95.     fft_ = fft(data_rejected[i])
96.     abs_ = np.abs(fft_)
97.     normalization_ = abs_ / len(data_rejected[i])
98.     normalization_rejected_[i] = normalization_[:40]
99.
100. print(np.shape(normalization_rejected_))
101. avg = np.mean(normalization_rejected_, axis=0)
102. avg = np.hstack((np.zeros(1, ), avg))
103.
104. plt.figure(figsize=(12, 4))
105. plt.plot(avg, "-*")
106. plt.xticks(np.arange(0, 41, 1))
107. plt.ylim(0, 5)
108. plt.yticks(np.arange(0, 5.5, 0.5))
109. plt.grid()
110.
111. pkl_filename = "./hd_c/hd_c_01_81.pkl"
112. with open(pkl_filename, 'wb') as file:
113.     pickle.dump(data_rejected, file)

```

The Proposed 1D-CNN Model

Data loader

```
1. import numpy as np
2. from torch.utils.data import Dataset
3.
4. ###
5. class DatasetEEG(Dataset):
6.     def __init__(self, data, label):
7.         self.data = data
8.         self.label = label
9.
10.    def __len__(self):
11.        return self.data.shape[0]
12.
13.    def __getitem__(self, index):
14.        return self.data[index], self.label[index]
```

Model

```
1. import torch
2. import torch.utils.data
3. import torch.nn as nn
4. from torchsummary import summary
5.
6. ###
7. class EpiNet(nn.Module):
8.     def __init__(self):
9.         super(EpiNet, self).__init__()
10.        self.cnn_block = nn.Sequential(
11.            # Conv Block 1
12.            nn.Conv1d(1, 32, kernel_size=3, stride=1, padding=1),
13.            nn.BatchNorm1d(32),
14.            nn.ReLU(),
15.            nn.Conv1d(32, 32, kernel_size=3, stride=1, padding=1),
```

```

16.         nn.BatchNorm1d(32),
17.         nn.ReLU(),
18.         nn.MaxPool1d(kernel_size=4, stride=4),
19.         # Conv Block 2
20.         nn.Conv1d(32, 64, kernel_size=3, stride=1, padding=1),
21.         nn.BatchNorm1d(64),
22.         nn.ReLU(),
23.         nn.Conv1d(64, 64, kernel_size=3, stride=1, padding=1),
24.         nn.BatchNorm1d(64),
25.         nn.ReLU(),
26.         nn.MaxPool1d(kernel_size=4, stride=4),
27.         # Conv Block 3
28.         nn.Conv1d(64, 128, kernel_size=3, stride=1, padding=1),
29.         nn.BatchNorm1d(128),
30.         nn.ReLU(),
31.         nn.Conv1d(128, 128, kernel_size=3, stride=1, padding=1),
32.         nn.BatchNorm1d(128),
33.         nn.ReLU(),
34.         nn.MaxPool1d(kernel_size=4, stride=4),
35.         # Conv Block 4
36.         nn.Conv1d(128, 256, kernel_size=3, stride=1, padding=1),
37.         nn.BatchNorm1d(256),
38.         nn.ReLU(),
39.         nn.Conv1d(256, 256, kernel_size=3, stride=1, padding=1),
40.         nn.BatchNorm1d(256),
41.         nn.ReLU(),
42.         nn.MaxPool1d(kernel_size=4, stride=4),
43.         # Conv Block 5
44.         nn.Conv1d(256, 512, kernel_size=3, stride=1, padding=1),
45.         nn.BatchNorm1d(512),
46.         nn.ReLU(),
47.         nn.Conv1d(512, 512, kernel_size=3, stride=1, padding=1),
48.         nn.BatchNorm1d(512),
49.         nn.ReLU(),
50.         nn.MaxPool1d(kernel_size=4, stride=4),
51.         nn.Dropout(0.5),
52.     )
53.     # FC Block
54.     self.fc_block = nn.Sequential(

```



```

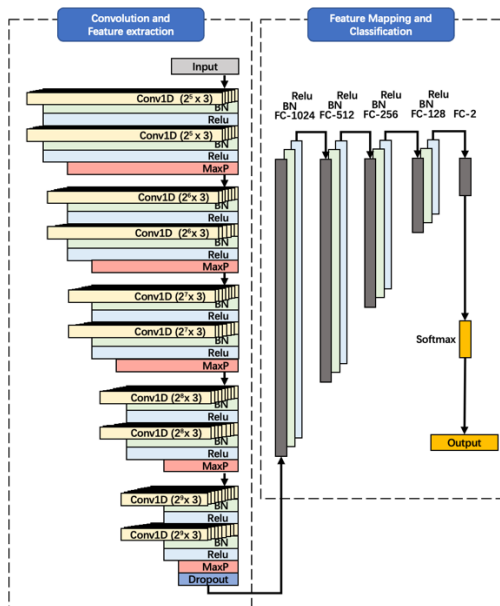
55.         # nn.Linear(19712, 1024),
56.         nn.Linear(9728, 1024),
57.         nn.BatchNorm1d(1024),
58.         nn.ReLU(),
59.
60.         nn.Linear(1024, 512),
61.         nn.BatchNorm1d(512),
62.         nn.ReLU(),
63.
64.         nn.Linear(512, 256),
65.         nn.BatchNorm1d(256),
66.         nn.ReLU(),
67.
68.         nn.Linear(256, 128),
69.         nn.BatchNorm1d(128),
70.         nn.ReLU(),
71.
72.         nn.Linear(128, 2)
73.     )
74.
75.     self.initialize_weights()
76.
77.     def forward(self, x):
78.         x = self.cnn_block(x)
79.         x = x.view(x.size(0), -1)
80.         x = self.fc_block(x)
81.         return x
82.
83.     def initialize_weights(self):
84.         for m in self.modules():
85.             if isinstance(m, nn.Conv1d):
86.                 nn.init.kaiming_normal_(m.weight, nonlinearity='relu')
87.                 if m.bias is not None:
88.                     nn.init.constant_(m.bias, 0)
89.             elif isinstance(m, nn.BatchNorm1d):
90.                 nn.init.constant_(m.weight, 1)
91.                 nn.init.constant_(m.bias, 0)
92.             elif isinstance(m, nn.Linear):
93.                 nn.init.normal_(m.weight, 0, 0.01)

```

```

94.         nn.init.constant_(m.bias, 0)
95.
96. def test():
97.     net = EpiNet()
98.     print(net)
99.     x = torch.randn(32, 1, 1000)
100.    y = net(x)
101.    print(y.shape)
102.
103. if __name__ == '__main__':

```



Block representation of the proposed model

Main

```

1. import torch
2. import torch.nn as nn
3. import torch.optim as optim
4. import torch.nn.functional as F
5. import torch.backends.cudnn as cudnn
6. import numpy as np
7.
8. import torchvision

```

```

9. import torchvision.transforms as transforms
10.
11. import os
12. import csv
13. import math
14. import time
15. import pickle
16. import h5py
17. import argparse
18.
19. from sklearn.model_selection import KFold
20. from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matri
x
21. from sklearn import metrics
22. from model import *
23. from data_loader import *
24. from sklearn.utils import shuffle
25.
26. from scipy import stats
27.
28. os.environ['CUDA_VISIBLE_DEVICES'] = '2'
29. ###
30. parser = argparse.ArgumentParser(description='PyTorch 1DCNN')
31. parser.add_argument('--lr', '-l',
32.                     default=1e-4,
33.                     type=float,
34.                     help='learning rate')
35. parser.add_argument('--batch_size', '-b',
36.                     default=128,
37.                     type=int,
38.                     help='batch size')
39. parser.add_argument('--epoch_num', '-e',
40.                     default=200,
41.                     type=int,
42.                     help='epoch num')
43. parser.add_argument('--resume', '-r',
44.                     action='store_true',
45.                     help='resume from checkpoint')
46. args = parser.parse_args()

```

```

47.
48.
49. ###
50. device = 'cuda' if torch.cuda.is_available() else 'cpu'
51. print('\n==> Device :', device)
52.
53. ### Training
54. def train(epoch):
55.     print('\nEpoch: %d/%d' % (epoch+1, args.epoch_num))
56.     train_loss, correct, total, t = 0, 0, 0, 0
57.
58.     net.train()
59.
60.     for batch_idx, (inputs, targets) in enumerate(train_loader):
61.         t1 = time.time()
62.
63.         inputs, targets = inputs.to(device=device, dtype=torch.float), tar
gets.to(device=device, dtype=torch.long)
64.
65.         optimizer.zero_grad()
66.
67.         outputs = net(inputs)
68.
69.         loss = criterion(outputs, targets)
70.         loss.backward()
71.         optimizer.step()
72.
73.         train_loss += loss.item()
74.         _, predicted = outputs.max(1)
75.         total += targets.size(0)
76.         correct += predicted.eq(targets).sum().item()
77.
78.         if batch_idx == 0:
79.             targets_total = targets
80.             predicted_total = predicted
81.         else:
82.             targets_total = torch.cat((targets_total, targets))
83.             predicted_total = torch.cat((predicted_total, predicted))
84.

```

```

85.         t2 = time.time()
86.         t += (t2-t1)
87.
88.
89.         tn, fp, fn, tp = metrics.confusion_matrix(targets_total.cpu(), pre
dicted_total.cpu()).ravel()
90.
91.         acc = (tn+tp)/(tn+fp+fn+tp)
92.         pre = tp/(tp+fp)
93.         rec = tp/(tp+fn)
94.         spe = tn/(tn+fp)
95.         f1 = (2*tp)/(2*tp+fp+fn)
96.
97.
98.         L = [' TR Batch: %d' %(batch_idx+1), '/%d | ' %(len(train_loader)
),
99.             'Loss: %.4f | ' %(train_loss/(batch_idx+1)),
100.            'Acc: %.4f | ' %acc,
101.            'Pre: %.4f | ' %pre,
102.            'Rec: %.4f | ' %rec,
103.            'Spe: %.4f | ' %spe,
104.            'F1: %.4f | ' %f1,
105.            'Time: %.3f' %t,
106.            ' <- %.3f' %((t2-t1)*(len(train_loader) - batch_idx -1))]
107.         L = ''.join(L)
108.
109.         print('\n' + L, end='', flush=True)
110.         print()
111.
112.         ###
113.         f = open("./result/log.txt", "a+")
114.         f.write('Epoch ' + str(epoch+1) + '\n' + L + '\n')
115.         f.close()
116.
117.         ###
118.         result[0, epoch, 0] = train_loss/(batch_idx+1)
119.         result[0, epoch, 1] = acc
120.         result[0, epoch, 2] = pre
121.         result[0, epoch, 3] = rec

```

```

122.     result[0, epoch, 4] = spe
123.     result[0, epoch, 5] = f1
124.
125.
126. ###
127. def test(epoch):
128.     global best_acc
129.     net.eval()
130.     test_loss, correct, total = 0, 0, 0
131.     target_total = []
132.     socre_total = []
133.     y_score_total = []
134.
135.     # y_score_all
136.
137.     with torch.no_grad():
138.         for batch_idx, (inputs, targets) in enumerate(test_loader):
139.             inputs, targets = inputs.to(device=device, dtype=torch.float)
, targets.to(device=device, dtype=torch.long)
140.
141.             outputs = net(inputs)
142.
143.             loss = criterion(outputs, targets)
144.
145.             test_loss += loss.item()
146.             score, predicted = outputs.max(1)
147. #
148.             total += targets.size(0)
149.             correct += predicted.eq(targets).sum().item()
150.
151.         ###
152.         if batch_idx == 0:
153.             targets_total = targets
154.             predicted_total = predicted
155.             score_total = score
156.             y_score_total = probabilities
157.         else:
158.             targets_total = torch.cat((targets_total, targets))

```

```

159.         predicted_total = torch.cat((predicted_total, predicted))
160.         score_total = torch.cat((score_total, score))
161.         y_score_total = torch.cat((y_score_total, probabilities))
162.
163.         tn, fp, fn, tp = metrics.confusion_matrix(targets_total.cpu()
, predicted_total.cpu()).ravel()
164.
165.         # print('score_total:', score_total)
166.
167.         acc = (tn+tp)/(tn+fp+fn+tp)
168.         pre = tp/(tp+fp)
169.         rec = tp/(tp+fn)
170.         spe = tn/(tn+fp)
171.         f1 = (2*tp)/(2*tp+fp+fn)
172.
173.
174.         L = [' TEST Result:   | ',
175.             'Loss: %.4f | ' %(test_loss/(batch_idx+1)),
176.             'Acc: %.4f | ' %acc,
177.             'Pre: %.4f | ' %pre,
178.             'Rec: %.4f | ' %rec,
179.             'Spe: %.4f | ' %spe,
180.             'F1: %.4f | ' %f1]
181.         L = ''.join(L)
182.
183.         print('\r' + L, end='', flush=True)
184.
185.         ###
186.         f = open("./result/log.txt", "a+")
187.         f.write(L + '\n')
188.         f.close()
189.
190.         ###
191.         result[1, epoch, 0] = test_loss/(batch_idx+1)
192.         result[1, epoch, 1] = acc
193.         result[1, epoch, 2] = pre
194.         result[1, epoch, 3] = rec

```

```

195.         result[1, epoch, 4] = spe
196.         result[1, epoch, 5] = f1
197.
198.         y_true = targets_total.cpu().numpy()
199.         y_score = y_score_total.cpu().numpy()
200.
201.         # Save checkpoint.
202.         acc = 100.*correct/total
203.         if acc > best_acc:
204.             print('Saving...')
205.
206.             f = open("./result/log.txt", "a+")
207.             f.write('Saving...' + '\n')
208.             f.close()
209.
210.             state = {
211.                 'net': net.state_dict(),
212.                 'acc': acc,
213.                 'epoch': epoch,
214.             }
215.             if not os.path.isdir('checkpoint'):
216.                 os.mkdir('checkpoint')
217.             torch.save(state, './checkpoint/ckpt.pth')
218.             best_acc = acc
219.
220.         return y_true, y_score
221.
222.     ###
223.     if __name__ == '__main__':
224.
225.         ###
226.         best_acc = 0 # best test accuracy
227.         start_epoch = 0 # start from epoch 0 or last checkpoint epoch
228.
229.         ###
230.         torch.manual_seed(1234)
231.         K = 5
232.         SEED = 1230
233.         shuffle = True

```



```

234.
235.     result = np.zeros([2, args.epoch_num, 6])
236.     all_result = np.zeros([K, 2, args.epoch_num, 6])
237.
238.     ###
239.     open('./result/log.txt', 'w').close()
240.
241.
242.     ###
243.     path = open('./data/hsd_data_8700_20000.pkl', 'rb')
244.     data = pickle.load(path)
245.     path.close()
246.
247.     path = open('./data/hsd_label_8700_20000.pkl', 'rb')
248.     label = pickle.load(path)
249.     path.close()
250.
251.     data = np.asarray(data, dtype = float)
252.     label = np.asarray(label, dtype = float)
253.
254.     y_true_all = np.empty((K, int(len(data)/K)))
255.     y_score_all = np.empty((K, int(len(data)/K)))
256.
257.     print("data type: ", type(data))
258.     print("data shape: ", np.shape(data))
259.     print("label shape: ", np.shape(label))
260.     ###
261.     kf = KFold(n_splits=K, random_state=SEED, shuffle=shuffle)
262.     kf_num = 0
263.
264.     for train_index, test_index in kf.split(data):
265.         print('\n' + '\n' + 'KFold :', kf_num+1)
266.
267.         ###
268.         f = open("./result/log.txt", "a+")
269.         f.write('\nKFold ' + str(kf_num+1) + '\n')
270.         f.close()
271.
272.         ###

```

```

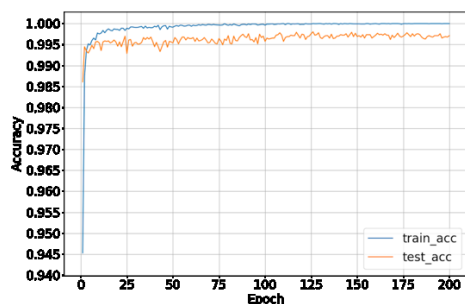
273.     train_data = data[train_index, :]
274.     train_data = train_data[:, np.newaxis, :]
275.     train_label = label[train_index]
276.
277.     test_data = data[test_index, :]
278.     test_data = test_data[:, np.newaxis, :]
279.     test_label = label[test_index]
280.
281.     ###
282.     train_set = DatasetEEG(data=train_data, label=train_label)
283.     train_loader = torch.utils.data.DataLoader(train_set,
284.                                                batch_size=args.batch_size
285.                                                ,
286.                                                shuffle=True,
287.                                                num_workers=2)
288.     test_set = DatasetEEG(data=test_data, label=test_label)
289.     test_loader = torch.utils.data.DataLoader(test_set,
290.                                                batch_size=args.batch_size,
291.                                                shuffle=True,
292.                                                num_workers=2)
293.
294.     ### Model
295.     net = EpiNet().to(device=device)
296.     criterion = nn.CrossEntropyLoss()
297.     if kf_num == 0:
298.         summary(net, input_size=(1, len(data[0])))
299.
300.     ###
301.     for epoch in range(start_epoch, start_epoch + args.epoch_num):
302.
303.         ###
304.         if epoch < 50:
305.             lr = args.lr
306.         else:
307.             lr = lr*0.995
308.
309.     ###

```

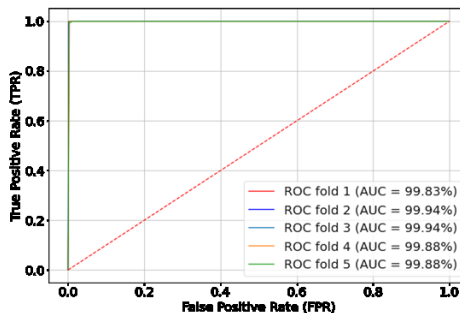
```

310.         optimizer = optim.Adam(net.parameters(), lr=lr,
311.                                 betas=(0.9, 0.999), eps=1e-08,
312.                                 weight_decay=0, amsgrad=False)
313.
314.         ###
315.         train(epoch)
316.         y_true, y_score = test(epoch)
317.         #         print(y_true)
318.         #         print(y_score)
319.         if epoch==args.epoch_num-1:
320.             #         if epoch==0:
321.                 y_true_all[kf_num] = y_true
322.                 y_score_all[kf_num] = y_score
323.                 # print(y_true)
324.                 # print(y_score)
325.             if kf_num==K-1:
326.                 save_file_path = open('./result/y_true_all.pkl','wb')
327.                 pickle.dump(y_true_all, save_file_path)
328.                 save_file_path.close()
329.
330.                 save_file_path = open('./result/y_score_all.pkl','wb')
331.                 pickle.dump(y_score_all, save_file_path)
332.                 save_file_path.close()
333.             ###
334.             all_result[kf_num, :, :, :] = result[:, :, :]
335.             kf_num += 1
336.             ###
337.             save_file_path = open('./result/result.pkl','wb')
338.             pickle.dump(all_result, save_file_path)

```



Accuracy

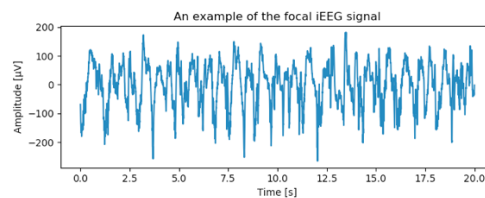
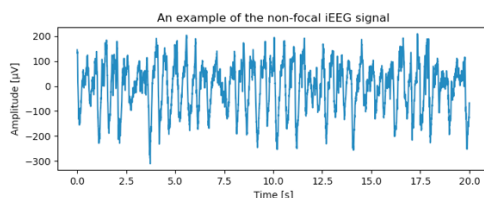


ROC

Study 2. The Automated Localization of Epileptic Foci

The Bern-Barcelona iEEG Database

```
1. import numpy as np
2. import pickle
3. import matplotlib.pyplot as plt
4.
5. path_f = open('bern_f_7500_10240.pkl', 'rb')
6. data_f = pickle.load(path_f)
7. path_f.close()
8.
9. path_n = open('bern_n_7500_10240.pkl', 'rb')
10. data_n = pickle.load(path_n)
11. path_n.close()
12.
13. x = np.arange(0, 20, 1/512)
14.
15. plt.figure(figsize = (8,3), dpi = 1000)
16. plt.plot(x, data_n[10])
17. plt.xlabel('Time [s]')
18. plt.ylabel('Amplitude [ $\mu$ V]')
19. plt.title('An example of the nonfocal iEEG signal')
20. plt.tight_layout()
21.
22. plt.savefig('ieegnonfocal.png')
23.
24. plt.show()
```



The Proposed 1D-CNN Model

```
1. import numpy as np
2. import keras
3. import tensorflow as tf
4. import os
5. import sys
6. import pickle
7.
8. from keras.models import Sequential
9. from keras.layers import Dense, Dropout, Activation, Flatten, MaxPool1D, BatchNormalization
10. from keras.utils import np_utils
11. from keras import backend as K
12. from keras.layers.convolutional import Conv1D
13. from keras.optimizers import SGD, Adam, RMSprop
14. from keras.callbacks import Callback
15.
16. from sklearn.utils import shuffle
17. from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix
18. from sklearn.model_selection import KFold
19.
20. import matplotlib.pyplot as plt
21.
22. gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.666)
23. sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
24.
25. y_true = np.zeros((0,))
26. y_score = np.zeros((0,))
27. CM = np.zeros((2,2))
28.
29. class ROCCallback(Callback):
30.     def __init__(self, model, val_data, val_label):
31.         self.model = model
32.         self.val_data = val_data
33.         self.val_label = val_label
34.
```

```

35.     def on_epoch_end(self, epoch, logs):
36.         global y_true
37.         global y_score
38.         global CM
39.
40.         pred_label = self.model.predict(self.val_data)
41.         cm = confusion_matrix(val_label.argmax(axis=1), np.round(pred_label
1).argmax(axis=1))
42.
43.         val_label_0 = [i[0] for i in val_label]
44.         pred_label_0 = [i[0] for i in pred_label]
45.
46.         y_true = np.concatenate((y_true, val_label_0))
47.         y_score = np.concatenate((y_score, pred_label_0))
48.         CM = np.concatenate((CM, cm))
49.
50. batch_size = 128
51. nb_classes = 2
52. epochs = 200
53. nb1_filters = 2
54. nb2_filters = 4
55. nb3_filters = 8
56. nb4_filters = 16
57. nb5_filters = 32
58. nb6_filters = 64
59. pool_size = 2
60. c_kernel_size = 3
61. input_shape = (20*512, 1)
62. n_splits = 10
63. K_acc = []
64. train_acc = []
65. train_loss = []
66. val_acc = []
67. val_loss = []
68. test_acc = []
69. test_loss = []
70.
71. path_f = open('bern_f_7500_10240.pkl', 'rb')
72. data_f = pickle.load(path_f)

```

```

73. path_f.close()
74.
75. path_n = open('bern_n_7500_10240.pkl', 'rb')
76. data_n = pickle.load(path_n)
77. path_n.close()
78.
79. data = np.vstack((data_f, data_n))
80. label = np.hstack((np.ones(7500,), np.zeros(7500,))).T
81.
82. data = shuffle(data, random_state = 1)
83. label = shuffle(label, random_state = 1)
84.
85. kf = KFold(n_splits = n_splits)
86. for train_index, test_index in kf.split(data):
87.
88.     train_val_data, test_data = data[train_index], data[test_index]
89.     train_val_label, test_label = label[train_index], label[test_index]
90.
91.     train_data = train_val_data[:12000]
92.     train_label = train_val_label[:12000]
93.     val_data = train_val_data[12000:]
94.     val_label = train_val_label[12000:]
95.
96.     train_data = train_data.reshape(train_data.shape[0], 10240, 1)
97.     test_data = test_data.reshape(test_data.shape[0], 10240, 1)
98.     val_data = val_data.reshape(val_data.shape[0], 10240, 1)
99.
100.    train_label = np_utils.to_categorical(train_label, nb_classes)
101.    test_label = np_utils.to_categorical(test_label, nb_classes)
102.    val_label = np_utils.to_categorical(val_label, nb_classes)
103.
104.    model = Sequential()
105.
106.    model.add(Conv1D(nb1_filters, c_kernel_size, strides = 1, padding = '
same',
107.        input_shape = input_shape, activation = 'relu'))
108.    model.add(Dropout(0.2))
109.    model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))

```

```

110.     model.add(BatchNormalization(axis = 1, momentum = 0.99, epsilon = 0.0
01))
111.
112.     model.add(Conv1D(nb2_filters, c_kernel_size, strides = 1, padding = '
same',
113.         input_shape = input_shape, activation = 'relu'))
114.     model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))
115.
116.     model.add(Conv1D(nb3_filters, c_kernel_size, strides = 1, padding = '
same',
117.         input_shape = input_shape, activation = 'relu'))
118.     model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))
119.
120.     model.add(Conv1D(nb4_filters, c_kernel_size, strides = 1, padding = '
same',
121.         input_shape = input_shape, activation = 'relu'))
122.     model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))
123.
124.     model.add(Conv1D(nb5_filters, c_kernel_size, strides = 1, padding = '
same',
125.         input_shape = input_shape, activation = 'relu'))
126.     model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))
127.
128.     model.add(Conv1D(nb6_filters, c_kernel_size, strides = 1, padding = '
same',
129.         input_shape = input_shape, activation = 'relu'))
130.     model.add(MaxPool1D(pool_size = pool_size, strides = 2, padding = 'va
lid'))
131.
132.     model.add(Dropout(0.2))
133.
134.     model.add(Flatten())
135.
136.     model.add(Dense(128))
137.     model.add(Activation('relu'))

```



```

138.     model.add(Dropout(0.5))
139.
140.     model.add(Dense(nb_classes))
141.     model.add(Activation('softmax'))
142.     adam = Adam(lr = 2.5e-4)
143.     # adam = keras.optimizers.SGD(lr = 2.5e-
15, momentum = 0.0, decay = 0.05, nesterov = True)
144.     model.compile(loss = 'categorical_crossentropy', optimizer = adam, me
etrics = ['accuracy'])
145.
146.     history = model.fit(train_data, train_label, batch_size = batch_size,
epochs = epochs,
147.         verbose = 1, validation_data = (val_data, val_label), callbacks =
[ROCCallback(model, val_data, val_label)])
148.
149.     score = model.evaluate(test_data, test_label, batch_size = batch_size
, verbose = 1)
150.
151.     print('Test loss:', score[0])
152.     print('Test accuracy:', score[1])
153.
154.     K_acc.append(score[1])
155.
156.     y_train_accuracy = history.history['acc']
157.     y_train_loss = history.history['loss']
158.     y_val_accuracy = history.history['val_acc']
159.     y_val_loss = history.history['val_loss']
160.
161.     train_acc.append(y_train_accuracy)
162.     train_loss.append(y_train_loss)
163.     val_acc.append(y_val_accuracy)
164.     val_loss.append(y_val_loss)
165.     test_acc.append(score[1])
166.     test_loss.append(score[0])
167.
168.     y = {'train_acc': train_acc, 'train_loss': train_loss, 'val_acc': val_acc
, 'val_loss': val_loss, 'test_acc': score[1], 'test_loss': score[0], 'K_acc': K_acc
}
169.

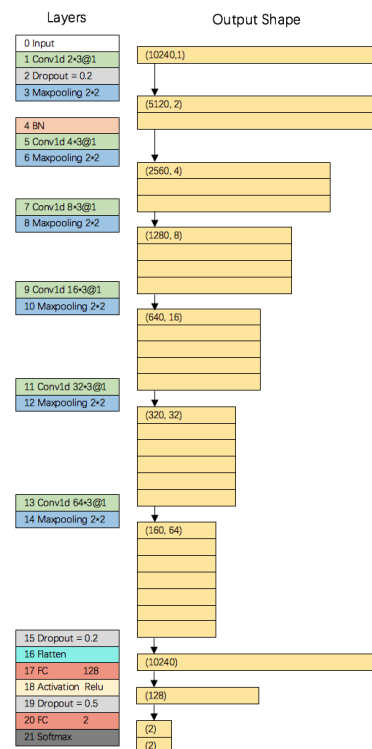
```

```

170. filename = os.path.basename(__file__)
171. save_file = open('%s.pkl'%filename, 'wb')
172. pickle.dump(y, save_file)
173. save_file.close()
174.
175. save_file = open('y_true.pkl', 'wb')
176. pickle.dump(y_true, save_file)
177. save_file.close()
178.
179. save_file = open('y_score.pkl', 'wb')
180. pickle.dump(y_score, save_file)
181. save_file.close()
182.
183. save_file = open('CM.pkl', 'wb')
184. pickle.dump(CM, save_file)
185. save_file.close()
186.
187. print(K_acc)
188. print(np.mean(K_acc))

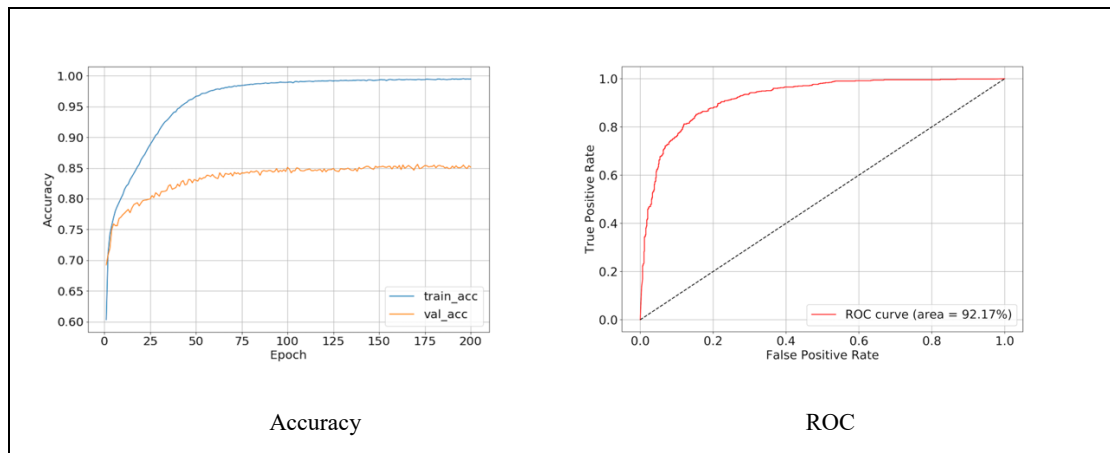
```

No.	Layer name	Kernel size	Stride	Number of parameters	Other parameters
0	Input	-	-	-	-
1	Conv1D	2×3	1	8	-
2	Dropout	-	-	0	Rate = 0.2
3	MaxP	2	2	0	-
4	BN	-	-	20,480	-
5	Conv1D	4×3	1	28	-
6	MaxP	2	2	0	-
7	Conv1D	8×3	1	104	-
8	MaxP	2	2	0	-
9	Conv1D	16×3	1	400	-
10	MaxP	2	2	0	-
11	Conv1D	32×3	1	1,568	-
12	MaxP	2	2	0	-
13	Conv1D	64×3	1	6,208	-
14	MaxP	2	2	0	-
15	Dropout	-	-	0	Rate = 0.2
16	Flatten	-	-	0	-
17	FC	128	-	1,310,848	-
18	Activation	-	-	0	Relu
19	Dropout	-	-	0	Rate = 0.5
20	FC	2	-	258	-
21	Softmax	2	-	0	-



Parameter values

Block representation of the developed model



Study 3. Construct of High-Precision BCI System

EEG signals pre-processing

```

1. import pandas as pd
2. import numpy as np
3.
4. from matplotlib import pyplot as plt
5. from utils import readname, butter_bandpass_filter
6.
7. import os
8. import pickle
9.
10. names = readname('./P300/subject6/')
11.
12. if '.DS_Store' in names:
13.     names.remove('.DS_Store')
14.
15. arr_x = np.zeros((1, 7))
16.
17. for i in names:
18.     # tempData = pd.read_csv("./data/"+i, skiprows=14, engine = "python",
header=None, delim_whitespace=True).to_numpy()
19.     tempData = pd.read_csv("./P300/subject6/"+i, skiprows=14).to_numpy()
20.     arr_x = np.vstack((arr_x, tempData))

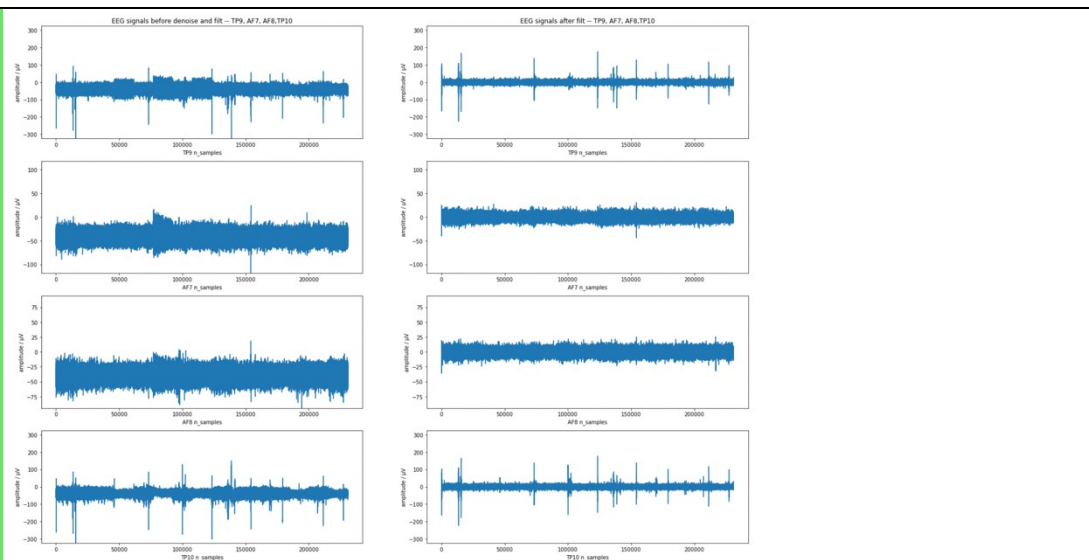
```

```

21.
22. data_raw = arr_x[1:].T[1:5]
23. label_raw = arr_x[1:].T[6]
24. print(np.shape(data_raw))
25.
26. max_apt = np.max(abs(data_raw), axis=1)
27. max_apt
28.
29. from sklearn.utils import shuffle
30. from scipy.signal import detrend
31.
32. # detrended
33. data_detrended = detrend(data_raw, axis=-
1, type='constant') # constant or linear
34.
35. # filter
36. data_filtered = butter_bandpass_filter(data_raw, 1, 30, 256)
37.
38. plt.figure(figsize=(24, 18))
39.
40. plt.subplot(421)
41. plt.plot(data_raw[0])
42. plt.title('EEG signals before denoise and filt -- TP9, AF7, AF8,TP10')
43. plt.xlabel('TP9 n_samples')
44. plt.ylabel('amplitude /  $\mu$ V')
45. plt.ylim((-max_apt[0], max_apt[0]))
46.
47. plt.subplot(423)
48. plt.plot(data_raw[1])
49. plt.xlabel('AF7 n_samples')
50. plt.ylabel('amplitude /  $\mu$ V')
51. plt.ylim((-max_apt[1], max_apt[1]))
52.
53. plt.subplot(425)
54. plt.plot(data_raw[2])
55. plt.xlabel('AF8 n_samples')
56. plt.ylabel('amplitude /  $\mu$ V')
57. plt.ylim((-max_apt[2], max_apt[2]))
58.

```

```
59. plt.subplot(427)
60. plt.plot(data_raw[3])
61. plt.xlabel('TP10 n_samples')
62. plt.ylabel('amplitude /  $\mu$ V')
63. plt.ylim((-max_apt[3], max_apt[3]))
64.
65. plt.subplot(422)
66. plt.plot(data_filtered[0])
67. plt.title('EEG signals after filt -- TP9, AF7, AF8,TP10')
68. plt.xlabel('TP9 n_samples')
69. plt.ylabel('amplitude /  $\mu$ V')
70. plt.ylim((-max_apt[0], max_apt[0]))
71.
72. plt.subplot(424)
73. plt.plot(data_filtered[1])
74. plt.xlabel('AF7 n_samples')
75. plt.ylabel('amplitude /  $\mu$ V')
76. plt.ylim((-max_apt[1], max_apt[1]))
77.
78. plt.subplot(426)
79. plt.plot(data_filtered[2])
80. plt.xlabel('AF8 n_samples')
81. plt.ylabel('amplitude /  $\mu$ V')
82. plt.ylim((-max_apt[2], max_apt[2]))
83.
84. plt.subplot(428)
85. plt.plot(data_filtered[3])
86. plt.xlabel('TP10 n_samples')
87. plt.ylabel('amplitude /  $\mu$ V')
88. plt.ylim((-max_apt[3], max_apt[3]))
89.
90. plt.show()
91.
```



```

92. ### get data by channels
93. tp9 = data_filded[0]
94. af7 = data_filded[1]
95. af8 = data_filded[2]
96. tp10 = data_filded[3]
97. epoch_start = 0
98. epoch_duration = 1
99. epoch_len = int(256*epoch_duration)
100. epoch_end = epoch_start + epoch_len
101. random_state = 1
102.
103. tp9_10 = np.vstack((tp9, tp10)).sum(axis=0)/2
104.
105. yy_0 = []
106. yy_1 = []
107.
108. for i in range(len(label_raw)):
109.     if label_raw[i]==1:
110.         yy_0.append(i)
111.
112. for i in range(len(label_raw)):
113.     if label_raw[i]==2:
114.         yy_1.append(i)
115.
116. a_len = int(len(tar)/4)
117. a = np.empty((a_len, epoch_len)) # -3
118. for i in range(a_len):

```

```

119.     aa = np.sum([tar[i*4+0], tar[i*4+1], tar[i*4+2], tar[i*4+3]], axis=0)
/4
120.     a[i] = aa
121.
122.     b_len = int(len(ntar)/4)
123.     b = np.empty((b_len, epoch_len)) # -3
124.     for i in range(b_len):
125.         bb = np.sum([ntar[i*4+0], ntar[i*4+1], ntar[i*4+2], ntar[i*4+3]], axis=0)/4
126.         b[i] = bb
127.
128.     yy = np.max([yy_1[-1], yy_0[-1]])
129.     n_padding_0 = epoch_len-(len(label_raw)-yy)%epoch_len
130.     print('n_padding_0 = %d' % n_padding_0)
131.
132.     if len(label_raw)-yy < epoch_len:
133.         tp9_10 = np.hstack((tp9_10, np.zeros((n_padding_0, ))))
134.
135.     _, counts = np.unique(label_raw, return_counts=True)
136.     print("n_targets:", counts[2], "n_nontargets:", counts[1])
137.
138.
139.     ### get data by label
140.     tar = np.empty((counts[2], epoch_len))
141.     m = 0
142.     for i in range(len(label_raw)):
143.         if label_raw[i]==2:
144.             tar[m] = tp9_10[i+epoch_start:i+epoch_end]
145.             m += 1
146.
147.     ntar = np.empty((counts[1], epoch_len))
148.     n = 0
149.     for i in range(len(label_raw)):
150.         if label_raw[i]==1:
151.             ntar[n] = tp9_10[i+epoch_start:i+epoch_end]
152.             # print('success:', i, n)
153.             n += 1
154.
155.

```

```

156. ### rejeck blink
157. blink_threshold = 35
158. blk_idx_tar = []
159. for i in range(len(tar)):
160.     itar_max = np.max(abs(tar[i]))
161.     if itar_max>blink_threshold:
162.         blk_idx_tar.append(i)
163. tar = np.delete(tar, blk_idx_tar, axis=0)
164.
165. blk_idx_ntar = []
166. for i in range(len(ntar)):
167.     intar_max = np.max(abs(ntar[i]))
168.     if intar_max>blink_threshold:
169.         blk_idx_ntar.append(i)
170. ntar = np.delete(ntar, blk_idx_ntar, axis=0)
171.
172. tar = shuffle(tar, random_state = random_state)
173. ntar = shuffle(ntar, random_state = random_state)
174.
175.
176. ### baseline
177. if epoch_start<0:
178.     for i in range(len(tar)):
179.         itar_mean = np.mean(tar[i, :epoch_start])
180.         tar[i] = tar[i]-itar_mean
181.
182.     for i in range(len(ntar)):
183.         intar_mean = np.mean(ntar[i, :epoch_start])
184.         ntar[i] = ntar[i]-intar_mean
185.
186.
187. print("n_targets shape (after reject):", np.shape(tar), "n_nontargets sha
pe:", np.shape(ntar))
188. print(len(blk_idx_tar), len(blk_idx_ntar))
189.
190. for i in range(len(tar)):
191.     plt.plot(tar[i])
192.
193. x_ticks = np.linspace(0, epoch_len, int(epoch_len/25.6)+1)

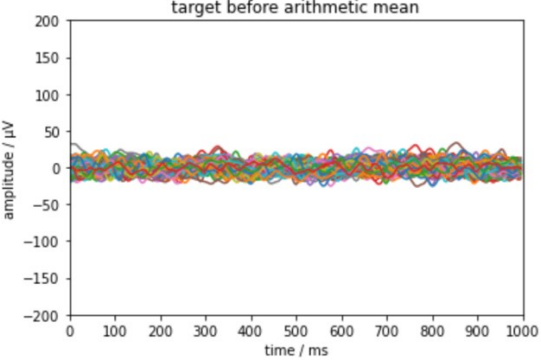
```



```

194. # x_label = np.arange(0, (int(epoch_len/25.6)+1)*100, 100)
195. x_label = np.arange((int(epoch_start/25.6))*100, ((int(epoch_len/25.6)+1)
*100)+((int(epoch_start/25.6))*100), 100)
196. plt.title('target before arithmetic mean')
197. plt.xlabel('time / ms')
198. plt.ylabel('amplitude /  $\mu$ V')
199.
200. # plt.vlines(51.2, -200, 200, colors = "grey", linestyle = "dashed")
201. # plt.vlines(128, -200, 200, colors = "red", linestyle = "dashed")
202.
203. plt.xlim((0, epoch_len))
204. plt.ylim((-200, 200))
205. plt.xticks(x_ticks, x_label)
206. plt.show()
207.

```

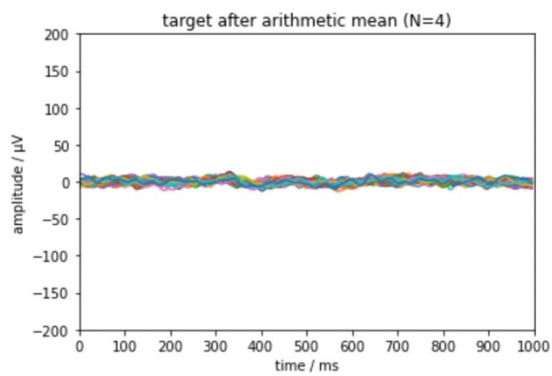


```

208. for i in range(len(a)):
209.     plt.plot(a[i])
210.
211. x_ticks = np.linspace(0, epoch_len, int(epoch_len/25.6)+1)
212. x_label = np.arange((int(epoch_start/25.6))*100, ((int(epoch_len/25.6)+1)
*100)+((int(epoch_start/25.6))*100), 100)
213. plt.title('target after arithmetic mean (N=4)')
214. plt.xlabel('time / ms')
215. plt.ylabel('amplitude /  $\mu$ V')
216.
217. # plt.vlines(51.2, -200, 200, colors = "grey", linestyle = "dashed")
218. # plt.vlines(128, -200, 200, colors = "red", linestyle = "dashed")
219.
220. plt.xlim((0, epoch_len))
221. plt.ylim((-200, 200))
222. plt.xticks(x_ticks, x_label)

```

```
223. plt.show()
```



```
224. for i in range(len(ntar)):
```

```
225.     plt.plot(ntar[i])
```

```
226.
```

```
227. x_ticks = np.linspace(0, epoch_len, int(epoch_len/25.6)+1)
```

```
228. x_label = np.arange((int(epoch_start/25.6))*100, ((int(epoch_len/25.6)+1)  
*100)+((int(epoch_start/25.6))*100), 100)
```

```
229. plt.title('non-target before arithmetic mean')
```

```
230. plt.xlabel('time / ms')
```

```
231. plt.ylabel('amplitude / μV')
```

```
232.
```

```
233. # plt.vlines(51.2, -200, 200, colors = "grey", linestyle = "dashed")
```

```
234. # plt.vlines(128, -200, 200, colors = "red", linestyle = "dashed")
```

```
235.
```

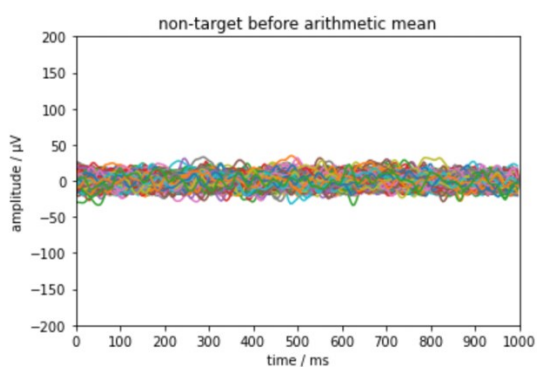
```
236. plt.xlim((0, epoch_len))
```

```
237. plt.ylim((-200, 200))
```

```
238. plt.xticks(x_ticks, x_label)
```

```
239. plt.show()
```

```
240.
```



```
241. for i in range(len(b)):
```

```
242.     plt.plot(b[i])
```

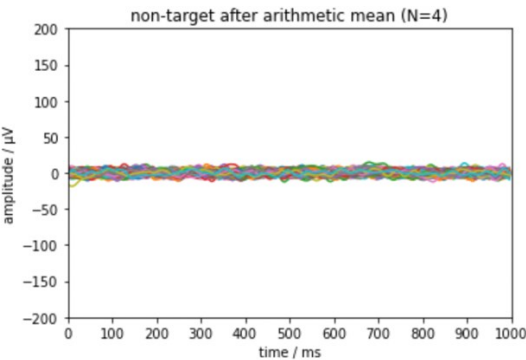
```
243.
```

```
244. x_ticks = np.linspace(0, epoch_len, int(epoch_len/25.6)+1)
```

```

245. x_label = np.arange((int(epoch_start/25.6))*100, ((int(epoch_len/25.6)+1)
*100)+((int(epoch_start/25.6))*100), 100)
246. plt.title('non-target after arithmetic mean (N=4)')
247. plt.xlabel('time / ms')
248. plt.ylabel('amplitude /  $\mu$ V')
249.
250. # plt.vlines(51.2, -200, 200, colors = "grey", linestyle = "dashed")
251. # plt.vlines(128, -200, 200, colors = "red", linestyle = "dashed")
252.
253. plt.xlim((0, epoch_len))
254. plt.ylim((-200, 200))
255. plt.xticks(x_ticks, x_label)
256. plt.show()

```



```

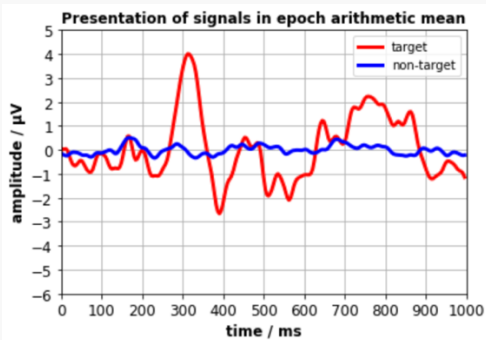
257.
258. tar_sum=tar.sum(axis=0)/len(tar)
259. ntar_sum=ntar[0:len(tar)].sum(axis=0)/len(ntar)
260. x = range(epoch_len)
261. x_ticks = np.linspace(0, epoch_len, int(epoch_len/25.6)+1)
262. x_label = np.arange((int(epoch_start/25.6))*100, ((int(epoch_len/25.6)+1)
*100)+((int(epoch_start/25.6))*100), 100)
263.
264. plt.plot(x, tar_sum, label='target')
265. plt.plot(x, ntar_sum, label='non-target')
266.
267. plt.title('Presentation of target and non-
target signals in epoch arithmetic mean')
268. plt.xlabel('time / ms')
269. plt.ylabel('amplitude /  $\mu$ V')
270.
271. # plt.vlines(51.2, -200, 200, colors = "grey", linestyle = "dashed")
272. # plt.vlines(128, -200, 200, colors = "red", linestyle = "dashed")

```

```

273.
274. plt.xlim((0, epoch_len))
275. plt.xticks(x_ticks, x_label)
276. plt.ylim((-10, 10))
277. plt.legend()
278. plt.show()
279.

```



```

280. idx = len(a)
281. td_rate = 0.8
282. data_1 = a # a, tar
283.
284. data_0 = shuffle(b, random_state = random_state) # b, ntar
285. data_0 = data_0[:idx]
286.
287. y0 = np.zeros(idx)
288. y1 = np.ones(idx)
289.
290. data__ = np.vstack((data_0, data_1))
291. label__ = np.hstack((y0, y1))
292.
293. data = shuffle(data__, random_state = random_state)
294. label = shuffle(label__, random_state = random_state)
295.
296. x_train = data[:int(idx*2*td_rate)]
297. x_test = data[int(idx*2*td_rate):]
298.
299. y_train = label[:int(idx*2*td_rate)]
300. y_test = label[int(idx*2*td_rate):]
301.
302. import numpy as np
303. import matplotlib.pyplot as plt

```

```

304.
305. from sklearn import svm, datasets
306. from sklearn.metrics import auc
307. from sklearn.metrics import plot_roc_curve
308. from sklearn.model_selection import StratifiedKFold
309.
310. from numpy import mean
311. from numpy import std
312.
313. from sklearn.model_selection import KFold, RepeatedKFold
314. from sklearn.model_selection import cross_val_score
315. from sklearn.linear_model import LogisticRegression
316.
317. # Add noisy features
318. random_state_clf = np.random.RandomState(0)
319.
320. # #####
#####
321. # Classification and ROC analysis
322.
323. # Run classifier with cross-validation and plot ROC curves
324.
325. cv = KFold(n_splits=2, random_state=random_state_clf, shuffle=True)
326. # cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=random_state)
327. # cv = StratifiedKFold(n_splits=10)
328.
329. clf_SVM = svm.SVC(kernel='linear', probability=True,
330.                   random_state=random_state_clf)
331. clf_LR = LogisticRegression(random_state=random_state)
332.
333. clf = clf_SVM
334. tprs = []
335. aucs = []
336. accs = []
337. mean_fpr = np.linspace(0, 1, 100)
338.
339. fig, ax = plt.subplots()
340. for i, (train, test) in enumerate(cv.split(data__, label__)):

```

```

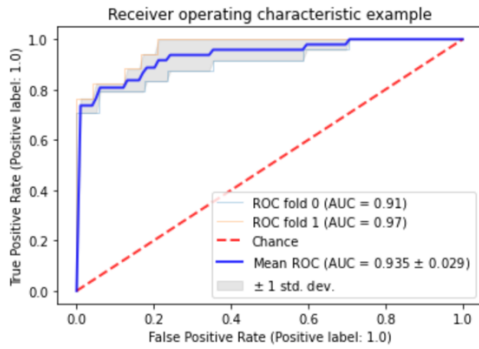
341.     clf.fit(data[train], label[train])
342.     acc = clf.score(data[test], label[test])
343.     viz = plot_roc_curve(clf, data[test], label[test],
344.                          name='ROC fold {}'.format(i),
345.                          alpha=0.3, lw=1, ax=ax)
346.     interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
347.     interp_tpr[0] = 0.0
348.     tprs.append(interp_tpr)
349.     auks.append(viz.roc_auc)
350.     accs.append(acc)
351.     print('Fold %d: Accuracy=%0.3f' % (i, acc))
352.
353. pkl_filename_svm = "clf_4sum-SVM-RKF30.pkl"
354. pkl_filename_lr = "clf_4sum-LR-RKF30.pkl"
355.
356. with open(pkl_filename_svm, 'wb') as file:
357.     pickle.dump(clf, file)
358. print('Save model success!')
359.
360. print('accuracy:%0.3f' % (np.mean(accs)))
361.
362. ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
363.         label='Chance', alpha=.8)
364.
365. mean_tpr = np.mean(tprs, axis=0)
366. mean_tpr[-1] = 1.0
367. mean_auc = auc(mean_fpr, mean_tpr)
368. std_auc = np.std(auks)
369. ax.plot(mean_fpr, mean_tpr, color='b',
370.         label=r'Mean ROC (AUC = %0.3f $\pm$ %0.3f)' % (mean_auc, std_auc)
371.         ,
372.         lw=2, alpha=.8)
373.
374. std_tpr = np.std(tprs, axis=0)
375. tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
376. tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
377. ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
378.                 label=r'$\pm$ 1 std. dev.')

```

```

378.
379. ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
380.         title="Receiver operating characteristic example")
381. ax.legend(loc="lower right")
382. plt.show()

```



ROC

P300 Stimulator

```

1. import pygame
2. import time
3. import numpy as np
4. import random
5. from pylsl import StreamInfo, StreamOutlet
6.
7. from sklearn.utils import shuffle
8. from pygame.locals import (
9.     K_SPACE,
10.    QUIT,
11. )
12.
13. # Initialize pygame
14. pygame.init()
15. clock = pygame.time.Clock()
16.
17. epoch_time = 400
18. delay_time = 100
19.
20.

```

```

21. class Stim(pygame.sprite.Sprite):
22.     def __init__(self, rect, screen, stats):
23.         super(Stim, self).__init__()
24.         self.rect = rect
25.         self.screen = screen
26.         self.stats = stats
27.         self.image0 = pygame.image.load('./stim_pic/0.png')
28.         self.image1 = pygame.image.load('./stim_pic/1.png')
29.         self.surf = pygame.image.load("./stim_pic/press.png").convert()
30.
31.     def draw(self):
32.         if self.stats == -1:
33.             self.surf = self.surf
34.         elif self.stats == 0:
35.             self.surf = pygame.transform.scale(self.image0, (199, 199)).co
nvert()
36.         elif self.stats == 1:
37.             self.surf = pygame.transform.scale(self.image1, (199, 199)).co
nvert()
38.         self.screen.blit(self.surf, self.rect)
39.
40.     def kill(self):
41.         self.kill()
42.
43.
44. class Button(pygame.sprite.Sprite):
45.     def __init__(self, rect, screen, stats):
46.         super(Button, self).__init__()
47.         self.rect = rect
48.         self.screen = screen
49.         self.stats = stats
50.         self.image = pygame.image.load('./stim_pic/press.png')
51.         self.surf = pygame.transform.scale(self.image, (299, 604)).convert
()
52.
53.     def draw(self):
54.         self.screen.blit(self.surf, self.rect)
55.
56.

```



```

57. class Press(pygame.sprite.Sprite):
58.     def press(self, pressed_keys):
59.         if pressed_keys[K_SPACE]:
60.             ifpress = 1
61.             timestamp = time.time()
62.             print("in def:", timestamp)
63.         else:
64.             ifpress = 0
65.
66.         return ifpress
67.
68. def update_stim():
69.     pygame.display.update(rect_)
70.     pygame.time.delay(delay_time)
71.
72.
73. def rand_position9(num_epoch, m=0, n=1923): # random 9
74.     epoch_ori = np.eye(9, dtype=np.int)
75.     pst = np.empty((num_epoch*9, 9))
76.     for i in range(num_epoch):
77.         pst[9*i:9*(i+1)] = shuffle(epoch_ori, random_state=random.randint(
m, n))
78.     return np.array(pst, dtype=int)
79.
80. def rand_position5(num_epoch, m=0, n=1923): # random 9
81.     id2 = np.hstack((np.array([0, 1]), np.zeros(7, )))
82.     id4 = np.hstack((np.array([0, 0, 0, 1]), np.zeros(5, )))
83.     id5 = np.hstack((np.array([0, 0, 0, 0, 1]), np.zeros(4, )))
84.     id6 = np.hstack((np.zeros(5, ), np.array([1, 0, 0, 0])))
85.     id8 = np.hstack((np.zeros(7, ), np.array([1, 0])))
86.     epoch_ori = np.vstack((id2, id4, id5, id6, id8))
87.
88.     pst = np.empty((num_epoch*9, 9))
89.     for i in range(num_epoch):
90.         pst[5*i:5*(i+1)] = shuffle(epoch_ori, random_state=random.randint(
m, n))
91.
92.     return np.array(pst, dtype=int)
93.

```

```

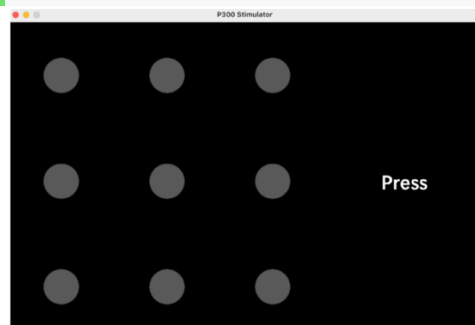
94. info = StreamInfo('Markers', 'Markers', 2, 256, 'float32', 'myuidw43536')
95. outlet = StreamOutlet(info)
96. SCREEN_WIDTH = 900
97. SCREEN_HEIGHT = 600
98. single = 600 / 3 - 1
99.
100. screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
101. pygame.display.set_caption('P300 Stimulator')
102.
103. rect = [0] * 10
104. rect_wh = [
105.     (1, 1), (single + 3, 1), (single * 2 + 5, 1),
106.     (1, single + 3), (single + 3, single + 3), (single * 2 + 5, single +
3),
107.     (1, single * 2 + 5), (single + 3, single * 2 + 5), (single * 2 + 5, s
ingle * 2 + 5),
108.     (single * 3 + 7, 1)
109. ]
110.
111. for i in range(len(rect)-1):
112.     rect[i] = pygame.Rect(*rect_wh[i], single, single) # (初始坐标, 长,
宽)
113.     rect[i] = Stim(rect[i], screen, 0)
114.
115. rect[9] = pygame.Rect(*rect_wh[9], single + 100, single * 3 + 7)
116. rect[9] = Button(rect[9], screen, -1)
117.
118. rect_ = rect[:9].copy()
119.
120. press = Press()
121.
122. FLIP = pygame.USEREVENT + 1
123. pygame.time.set_timer(FLIP, epoch_time)
124.
125. position = rand_position5(10000)
126. epoch = 0
127.
128. bg = pygame.image.load('./stim_pic/bg.png')
129. bg = pygame.transform.scale(bg, (603, 603)).convert()

```

```

130.
131. running = True
132. while running:
133.     for event in pygame.event.get():
134.         label = 0
135.         pressed_time = 0
136.         if event.type == QUIT:
137.             running = False
138.
139.         elif event.type == FLIP:
140.             for i, element in enumerate(rect_):
141.                 element.stats = position[epoch][i]
142.                 marker = int(np.unique(position[epoch], return_index=True
) [1][1])
143.                 element.draw()
144.                 label = marker + 1
145.                 update_stim()
146.
147.             rect[9].draw()
148.             pressed_keys = pygame.key.get_pressed()
149.             ifpress = press.press(pressed_keys)
150.
151.             timestamp = time.time()
152.             outlet.push_sample([label, ifpress], timestamp)
153.
154.             # screen.fill((0, 0, 0))
155.             screen.blit(bg, (0, 0))
156.             # pygame.display.update(rect_)
157.             pygame.display.flip()
158.             clock.tick(60)
159.             pressed_time = 0
160.             epoch += 1

```



Classifier

```
1. import numpy as np
2. from pylsl import StreamInlet, resolve_byprop
3. import utils
4. import time
5. import pandas as pd
6. import pickle
7.
8.
9. BUFFER_LENGTH = 7.5
10. fs = 256
11. timestamp_marker_buffer = []
12. epoch = 0
13. epoch_0 = 0
14.
15. # classifier
16. pkl_filename = "clf_4sum-SVM-RKF30.pkl"
17. with open(pkl_filename, 'rb') as file:
18.     clf = pickle.load(file)
19.
20.
21. if __name__ == "__main__":
22.
23.     # eeg stream
24.     streams = resolve_byprop('type', 'EEG', timeout=1)
25.     inlet = StreamInlet(streams[0], max_chunklen=12)
26.
27.     # marker stream
28.     marker_streams = resolve_byprop('name', 'Markers', timeout=1)
29.     marker_inlet = StreamInlet(marker_streams[0])
30.
31.     eeg_buffer = np.zeros((int(fs * BUFFER_LENGTH * 2), 5))
32.     timestamp_buffer = np.zeros((int(fs * BUFFER_LENGTH * 2),))
33.
34.     marker_buffer = np.zeros((1, 2))
35.     timestamp_marker_buffer = np.zeros((1, ))
36.
```

```

37.     start_time = time.time()
38.
39.     try:
40.         while True:
41.             time_correction01 = inlet.time_correction()
42.             time_correction02 = marker_inlet.time_correction()
43.
44.             data_eeg, timestamp_01 = inlet.pull_chunk(timeout=1, max_sampl
es=int(fs))
45.             tc_1 = np.array(timestamp_01) + time_correction01
46.
47.             eeg_buffer = np.vstack((eeg_buffer, np.array(data_eeg)))[-
len(eeg_buffer):] # (2816,)
48.             timestamp_buffer = np.hstack((timestamp_buffer, tc_1))[-
len(timestamp_buffer):]
49.
50.             data_marker, timestamp_02 = marker_inlet.pull_chunk(timeout=0,
max_samples=int(fs)) # timeout=0 !!!!!!!
51.
52.             if epoch > 0:
53.                 tc_2 = np.array(timestamp_02) + time_correction02
54.                 marker_buffer = np.vstack((marker_buffer, np.array(data_ma
rker)))
55.                 timestamp_marker_buffer = np.hstack((timestamp_marker_buff
er, tc_2))
56.
57.                 if timestamp_buffer[-
1] >= (start_time+epoch_0*BUFFER_LENGTH):
58.                     epoch_0 += 1
59.                     print("\n-----epoch-----
", epoch_0)
60.
61.                     timestamp_marker_buffer = np.reshape(timestamp_marker_
buffer[1:], (-1, 1))
62.
63.                     marker_buffer = marker_buffer[1:]
64.                     press_buffer = utils.change2one(marker_buffer[:, 1])
65.                     marker_buffer[:, 1] = press_buffer

```

```

66.             marker_all = np.hstack((marker_buffer, timestamp_marke
r_buffer))
67.             marker_all = utils.delete00(marker_all)
68.             data = pd.DataFrame(data=eeg_buffer, columns=["TP9", "
AF7", "AF8", "TP10", "AUX"])
69.
70.             for ii in range(2):
71.                 data['Marker%d' % ii] = 0
72.
73.             for marker in marker_all:
74.                 ix = np.argmin(np.abs(marker[2] - timestamp_buffer
))
75.                 for ii in range(2):
76.                     data.loc[ix, "Marker%d" % ii] = marker[ii]
77.
78.             data = np.array(data)
79.             if np.sum(press_buffer) > 0:
80.                 label, idx_label = utils.findMarker(data)
81.
82.                 if len(idx_label) >= 4:
83.                     tp9 = data.T[0]
84.                     tp10 = data.T[3]
85.                     tp9_10 = np.sum([tp9, tp10], axis=0)
86.                     data_filtered = utils.butter_bandpass_filter(t
p9_10, 1, 30, 256)
87.                     data_0 = np.hstack((data_filtered, np.zeros((2
56, )))
88.                     data_1 = np.empty((len(idx_label), 256))
89.
90.                     for i in range(len(idx_label)):
91.                         data_1[i] = data_0[idx_label[i]:idx_label[
i]+256]
92.                     data_2 = np.sum(data_1, axis=0)
93.                     pred = clf.predict(np.reshape(data_2, (1, -
1)))
94.
95.                     if pred == 1:
96.                         print("success!")
97.                         with open("commend.pkl", "wb") as f:

```

```
98.             pickle.dump([epoch, label], f, proto=
l=2)
99.             f.close
100.            else:
101.                print("failed..")
102.
103.                # print("press label:", label, "  press time
s:", len(idx_label))
104.                print("recognized label:", label)
105.
106.                marker_buffer = np.zeros((1, 2))
107.                timestamp_marker_buffer = np.zeros((1, ))
108.
109.                epoch += 1
110.
111.            except KeyboardInterrupt:
```