Available online at www.ijournalse.org

Emerging Science Journal

(ISSN: 2610-9182)

Vol. 10, No. 3, June, 2026

Stroke Detection Using EEG and Deep Learning: A Comparative Study of Feature Engineering Techniques

May Issa Aldossary [1], Fatemah H. Alghamedy [2*], Dina A. Alabbad [3], Reem A. H. Alshami [3], Haya A. Alzahim [3], Renad A. Alnuaim [3], Maimonah S. Altaweel [3], Shahad F. Alotaibi [3], Sumayh S. Aljameel [4], Areej A. Almalki [2], Sunday O. Olatunji [5]

[1] *Department of Computer Information Systems, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia.*

[2] *Computer Science Department, Applied College, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia.*

[3] *Department of Computer Engineering, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam 34212, Saudi Arabia.*

[4] *Aramco Saudi Accelerated Innovation Lab (aramcoSAIL), Saudi Aramco, Dhahran 31311, Saudi Arabia.*

[5] *Department of Computer Science, Adekunle Ajasin University, Akungba Akoko, 342111, Ondo State, Nigeria.*

Abstract

Strokes remain one of the leading causes of disability and mortality worldwide, underscoring the need for effective early detection and intervention methods. Recently, researchers have shown a growing interest in harnessing bio-signals, natural indicators produced by the human body, as potential markers for stroke detection. Multiple types of bio-signals, such as electroencephalography (EEG), are currently being explored in stroke diagnostic studies. This approach is promising because it offers a non-invasive, cost-effective, accurate, and portable means of detecting strokes. The objectives of this research are to investigate the effectiveness of deep learning (DL) techniques, including Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), Recurrent Neural Networks (RNN), CNN-LSTM hybrid models, and CNN-Gated Recurrent Unit (CNN-GRU) models, for detecting early-stage strokes based on EEG data. In addition, the impact of diverse feature extraction techniques, including utilizing all features, selecting features with a Decision Tree (DT) based on different thresholds, Principal Component Analysis (PCA), and Independent Component Analysis (ICA), is analyzed to evaluate their influence on model performance. A comparative discussion is conducted across multiple experimental setups to identify the most effective DL and feature engineering combinations for stroke detection. Across 35 different experiments, the CNN-LSTM model with seven selected features using the DT method yields the best results, achieving 86% accuracy, 99% precision, 81% recall, and an F1-score of 89%.

Keywords:
Bio-Signals;
Deep Learning;
EEG;
Stroke Detection;
Feature Engineering.

Article History:

Received:	17	October	2025
Revised:	19	May	2026
Accepted:	25	May	2026
Published:	01	June	2026

1- Introduction

Each year, strokes affect more than 15 million individuals worldwide, causing death and long-term disability [1]. The brain requires blood and oxygen to function properly. During a stroke, this supply is blocked or disrupted, causing brain cells to begin dying within minutes. Strokes are generally classified into two types: ischemic and haemorrhagic. Ischemic strokes occur due to blockages in blood vessels, while haemorrhagic strokes are caused by

* **CONTACT**: falghamedy@iau.edu.sa

DOI: https://doi.org/10.28991/ESJ-2026-010-03-023

bleeding. Both types require fast treatment to reduce their consequences [2, 3]. The effects may include problems with speech or movement, depending on the severity of the stroke and its location [4]. Promptly recognizing a stroke enables medical intervention to reduce the associated harm. This highlights the urgent need for improved early detection and intervention methods.

Over the past few years, there has been a growing interest in studying bio-signals, such as electro-encephalography (EEG) and electrocardiography (ECG), to possibly forecast strokes and other neurological incidents [3, 5]. These signals provide a real-time examination of the biological processes that occur in the body, specifically in the brain and cardiovascular system. This examination facilitates the detection of irregularities that may indicate the onset of stroke [6]. EEG is considered vital for early stroke detection, as it measures brain activity by tracking voltage changes produced by neuronal firing [7]. Additionally, EEG aids in understanding walking intentions in chronic stroke patients [8, 9]. EEG's effectiveness lies in its portability, affordability, and precision, allowing it to be used in various healthcare settings. Compared with Computed Tomography (CT) scans, which use ionizing radiation and may not always be easily accessible, EEG provides a safer and more convenient option for early stroke detection and monitoring [10].

Although strokes are most commonly detected using imaging techniques such as CT or Magnetic Resonance Imaging (MRI), this study aims to develop Deep Learning (DL) models that utilize EEG signals for early and accurate stroke detection. The study also seeks to lay the foundation for an accessible and scalable wearable mobile solution for real-world clinical settings. The significance of this development lies in enabling faster stroke detection, which can help save patients' lives, especially for older adults and individuals with disabilities whose symptoms may often go unnoticed. Imagine a wearable EEG device that tracks brain activity and immediately alerts doctors if a signal appears to be abnormal. Such devices could assist paramedics in deciding to transfer a patient to a specialized stroke center or the nearest hospital. Despite advances in utilizing DL for medical purposes, many studies have not thoroughly explored a range of DL methods or experimented with different feature extraction techniques. This research fills that gap by testing five different DL configurations and exploring feature extraction techniques such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Decision Tree (DT)-based selection. The aim of this study is to identify the most effective tools for stroke detection using EEG signals, providing practical solutions to enhance diagnosis in real-world scenarios.

The paper’s structure is designed to lead the reader logically through the key components of the research. Section 2 offers an overview of relevant prior work and literature, establishing the foundation for the study. Section 3 focuses on the proposed methodology and dataset description. Section 4 presents the results of the proposed methods. Section 5 provides a discussion that interprets the results. Finally, conclusions and suggestions for future work are presented in Section 6.

2- Related Works

2-1- Stroke Detection

This section presents a review of prior studies related to stroke detection using bio-signals. The reviewed papers are categorized based on the type of signal and their combinations. Additionally, a brief overview of the employed bio-signals is provided. EEG measures electrical activity in the brain and is commonly used to detect neural patterns and diagnose disorders such as epileptic seizures [11]. ECG measures the electrical activity of the heart, aiding in the diagnosis of cardiac disease and arrhythmias [12]. Electromyography (EMG) examines muscle electrical activity and assists in the identification of neuromuscular disorders [13]. Photoplethysmography (PPG) measures variations in blood volume using fingertip sensors to monitor heart rate and detect blood flow irregularities [14].

2-1-1- EEG Signals

A study by Choi et al. [15] aims to apply DL models using raw EEG data. The proposed system consists of a stroke prediction module that predicts the presence of a stroke using various DL models: including Convolutional Neural Network–Bidirectional Long Short-Term Memory (CNN–BiLSTM), Bidirectional Long Short-Term Memory (BiLSTM), and Convolutional Neural Network–Long Short-Term Memory (CNN–LSTM) models. The results indicate that the CNN-BiLSTM achieved the highest accuracy of 94.0% among all models implemented. In addition, in a paper by Kalahasty & Motati [16], the authors proposed a web application that takes a 60-second EEG record as input, then automatically provides a diagnosis and visualizes brain wave activities. The authors constructed three Artificial Neural Network (ANN) models. The first model was developed to predict the kind of stroke (hemorrhagic/ischemic/healthy), the second model was built to identify the stroke location (right hemisphere/left hemisphere), and the last model was built to determine the stroke severity (large/small). The proposed work achieved

remarkable results in predicting the type of stroke, determining the stroke location, and measuring the severity with an accuracy of 95%, 94.4%, and 100%, respectively. The study by Kumar & Sengupta [17] introduced models to categorize EEG signals as strokes or non-strokes. The study utilized VGG-16 and ResNet-50 DL model architectures, both of which achieved an accuracy of 90%.

2-1-2- ECG Signals

A study by Anand Kumar et al. [18] provided a medical framework for detecting irregularities in ECG data associated with stroke using DL techniques. The proposed system employs various algorithms, including Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) networks. The models were assessed based on the accuracy score. Compared to other models, the experimental findings reveal that the LSTM model provided the best performance with an accuracy of 93.78%. Furthermore, a paper by Kunwar & Choudhary [19] developed a highly effective stacked model that detects stroke disease based on ECG signals. The proposed model architecture is composed of a stacking ensemble model with three 1D CNN sub-models; each sub-model consists of other 1D CNN layers that are mainly built for ECG feature extraction. The sub-models were stacked at the top of the meta-learner layers to combine the sub-models' results and predict stroke occurrence. The results of the stacked model demonstrate the effectiveness in diagnosing stroke with an accuracy of 99.7%, F1 of 99.69%, recall of 99.71%, and precision of 99.67%.

2-1-3- EMG Signals

In a recent study by Elbagoury et al. [20], the authors proposed a novel approach using a portable Artificial Intelligence (AI) system aimed at predicting cardiac and cerebral stroke-related emergencies. The framework of this study integrates the Internet of Things (IoT) with AI technologies. The mobile AI agent receives EMG signal inputs from the wearable sensor module, and the data was analyzed using DL methods such as stacked LSTM and Group Method of Data Handling (GMDH) models to predict the occurrence of stroke. The implementation of the proposed system achieved a precision of 99.9% in predicting stroke. Table 1 summarizes the studies on DL methods for stroke detection with different types of bio-signals.

Table 1. Summary of studies on DL methods for stroke detection

Ref.	Data Type	Dataset Size	DL Methods	Results
Choi et al. (2021) [15]	EEG	273 Stroke Patients and 212 Control Patients	CNN-BiLSTM, LSTM, BiLSTM, CNN-LSTM	Accuracy: CNN-BiLSTM scored 94.0%, LSTM scored 70.1%, BiLSTM scored 91.8%, CNN-LSTM scored 93.7%.
Kalahasty et al. (2022) [16]	EEG	20 Stroke Patients and 10 Control Patients	ANN	Type of stroke scored 95%, Stroke location scored 94.4%, ensuring the severity 100%.
Kumar & Sengupta (2022) [17]	EEG	33 Stroke Patients and Control Patients	VGG-16 and ResNet-50	Accuracy: VGG-16 and RESNET-50 scored 90%.
Anand Kumar et al. (2022) [18]	ECG	4068 ECG Records	CNN, RNN, and LSTM	Accuracy: CNN scored 89.25% , RNN scored 86.19%, and LSTM scored 93.78%.
Kunwar et al. (2023) [19]	ECG	35 Stroke Patients and 36 Control Patients	Stacked CNN	Accuracy (CNN): 99.7%.
Elbagoury et al. (2023) [20]	EMG	38 Stroke Patients	Stacked LSTM and GMDH	Precision: 99.9%

Gap Analysis and Contribution

Although previous studies utilized EEG signals to detect strokes, they differed in EEG data collection setups and dataset sizes, making performance comparisons challenging. In addition, most prior studies focused on achieving higher classification accuracy but were limited in the variety of DL architectures and feature selection approaches explored. Table 2 summarizes the comparisons based on the DL algorithms and feature engineering techniques used, highlighting the need to further investigate the effectiveness of multiple DL architectures combined with feature engineering methods for EEG-based stroke detection. In this study, we evaluated five different DL models and examined the impact of various feature extraction techniques, including PCA, ICA, and DT-based selection.

This study extends prior work on EEG-based stroke detection by:

- It evaluates a broader set of DL models (five different algorithms) compared to the limited approaches used in earlier studies.
- It systematically analyzes the impact of diverse feature extraction techniques, including PCA, ICA, and Decision Tree-based methods, on model performance.
- It provides a comprehensive comparative discussion across multiple experimental setups to identify the most effective DL–feature engineering combinations for EEG-based stroke detection.

Table 2. Comparison of prior studies in DL methods and feature engineering techniques for stroke prediction using EEG signals

Model/Ref.	Choi et al. [15]	Kalahasty et al. [16]	Kumar et al. [17]	Present Study
CNN	×	×	×	✓
LSTM	✓	×	×	✓
CNN-LSTM	✓	×	×	✓
CNN-GRU	×	×	×	✓
RNN	×	×	×	✓
ANN	×	✓	×	×
Bi LSTM	✓	×	×	×
VGG-16	×	×	✓	×
RESNET-50	×	×	✓	×
ICA technique	×	×	×	✓
PCA technique	×	×	×	✓
DT technique	×	×	×	✓
EEG	✓	✓	✓	✓

3- Material and Methods

This study developed binary classification models for early-stage stroke detection using EEG signals, implemented in Python via Google Colab. Four experimental setups were employed, as shown in Figure 1. The first setup used all features, the second used features selected by DT with three different thresholds, and the third and fourth setups applied PCA and ICA, respectively. For all setups, the dataset was split into 65% for training, 15% for validation, and 20% for testing. Five DL models were developed: CNN, RNN, LSTM, CNN–LSTM, and CNN–GRU (Convolutional Neural Network–Gated Recurrent Unit). Optimal hyperparameters for all models were determined using manual tuning due to memory constraints in the online environment. Model performance was evaluated using accuracy, precision, recall, and F1-score, and the confusion matrix was reported for the best-performing models.

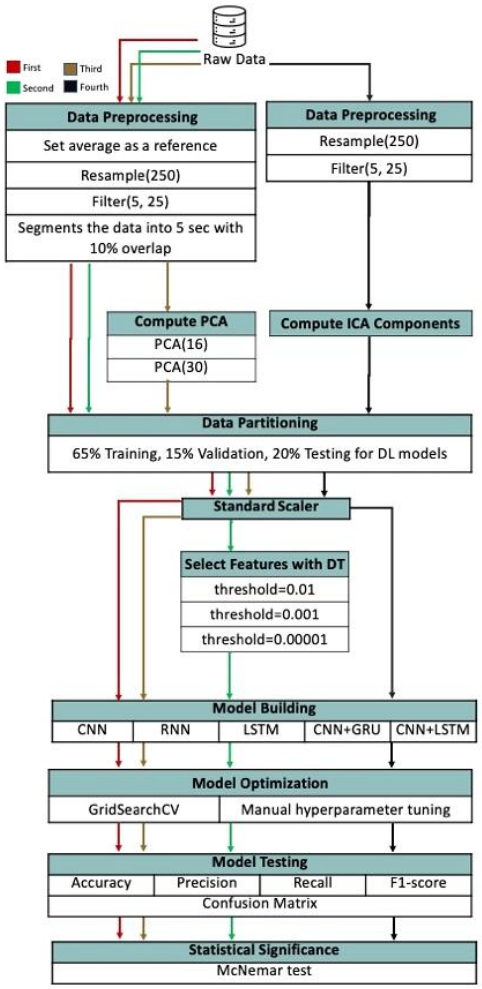


Figure 1. Overview of the four experimental setups with different feature selection techniques (first: all features (no feature selection), second: DT-based, third: PCA, fourth: ICA)

3-1- Dataset Description

This study employed the Stroke EIT (Electrical Impedance Tomography) dataset [21], which originally consisted of 48 EEG records from 33 individuals (10 healthy and 23 stroke patients). Due to technical issues, three records were excluded from the experiments. Table 3 lists the 34 EEG features (channels) present in the dataset, along with their corresponding positions in Figure 2.

Table 3. Features description of the Stroke EIT dataset [21]

#	Feature	Description	Type	#	Feature	Description	Type
1	**Time**	**Time of recording in seconds**	**float**	**18**	**Status**	**Stimulus value**	**Float**
2	A1	EEG channel contains the voltage value	float	19	A17	EEG channel contains the voltage value	float
3	A2		float	20	A18		float
4	A3		float	21	A19		float
5	A4		float	22	A20		float
6	A5		float	23	A21		float
7	A6		float	24	A22		float
8	A7		float	25	A23		float
9	A8		float	26	A24		float
10	A9		float	27	A25		float
11	A10		float	28	A26		float
12	A11		float	29	A27		float
13	A12		float	30	A28		float
14	A13		float	31	A29		float
15	A14		float	32	A30		float
16	A15		float	33	A31		float
17	A16		float	34	A32		float

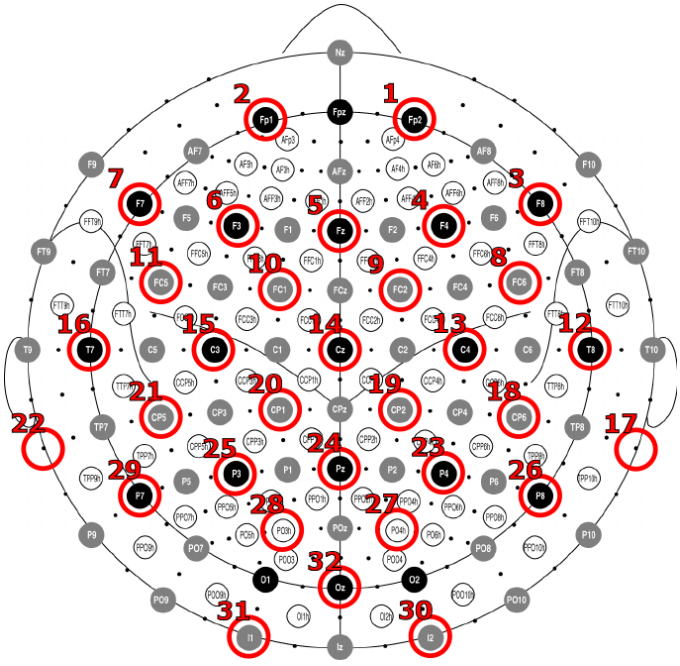


Figure 2. Electrodes (channels) Position [21]

3-2- Raw Dataset Statistical Analysis

Statistical analysis is essential for identifying patterns and trends within a dataset. The average record duration was 12.17 minutes. Due to limited RAM (Random Access Memory) capacity, it was not feasible to process the entire dataset. To address this, 90-second random samples were extracted from each record. Table 4 summarizes this sampling, showing the minimum value, first quartile (Q1), median, third quartile (Q3), and maximum value for each channel.

Table 4. Statistical analysis of the Stroke EIT dataset samples [21]

Ch	Name	Type	Unit	Min	Q1	Median	Q3	Max
0	A1	EEG	µV	-540772.34	-7165.31	889.31	10180.18	548666.96
1	A2	EEG	µV	-997640.90	-7420.81	2180.94	8759.56	984472.03
2	A3	EEG	µV	-587817.44	-4527.69	1051.94	7098.18	602170.56
3	A4	EEG	µV	-985137.53	-5830.44	1361.44	8777.68	984839.90
4	A5	EEG	µV	-781349.49	-5948.69	-603.19	8445.18	791082.36
5	A6	EEG	µV	-998695.40	-5517.06	1190.19	6775.81	987206.28
6	A7	EEG	µV	-999004.02	-6584.68	2166.19	9195.81	985117.40
7	A8	EEG	µV	-704407.89	-3726.94	2758.94	10263.18	714484.76
8	A9	EEG	µV	-919604.55	-6283.81	-782.44	5717.06	964747.53
9	A10	EEG	µV	-998424.77	-7140.31	-398.94	6128.43	984827.40
10	A11	EEG	µV	-997963.65	-1752.94	4989.69	11483.18	983868.65
11	A12	EEG	µV	-581323.82	-1725.94	3368.56	9176.68	578787.57
12	A13	EEG	µV	-911947.06	-2241.44	2245.81	8086.56	907275.68
13	A14	EEG	µV	-854364.46	-7382.31	-391.31	4904.06	881783.94
14	A15	EEG	µV	-998001.77	-3952.19	1126.31	9788.81	983733.53
15	A16	EEG	µV	-998925.77	-1802.94	3888.31	11234.56	985315.15
16	A17	EEG	µV	-997259.65	-4149.81	1595.94	10132.93	984171.78
17	A18	EEG	µV	-774960.36	-3960.31	3565.69	10542.56	823428.97
18	A19	EEG	µV	-998308.40	-6725.68	982.94	6641.93	983681.78
19	A20	EEG	µV	-998414.27	-4862.69	2918.56	7994.93	984754.90
20	A21	EEG	µV	-999592.40	-4507.56	4033.44	10065.68	986245.40
21	A22	EEG	µV	-968734.53	-2066.19	3559.44	11321.93	921665.80
22	A23	EEG	µV	-999373.77	-4780.94	950.94	7009.43	985990.03
23	A24	EEG	µV	-919883.18	-7628.81	208.06	10019.68	927190.05
24	A25	EEG	µV	-889828.57	-2049.94	4455.69	10927.81	908526.68
25	A26	EEG	µV	-991067.02	-4518.69	388.19	6432.68	984552.40
26	A27	EEG	µV	-997698.40	-4520.69	2513.94	11172.81	975340.53
27	A28	EEG	µV	-997677.15	-3745.19	4190.31	10283.68	985232.28
28	A29	EEG	µV	-998009.90	-3593.19	3813.56	10907.43	986483.28
29	A30	EEG	µV	-757909.00	-4377.31	3023.06	11120.06	742954.00
30	A31	EEG	µV	-996917.77	-5642.94	2354.19	11695.06	985156.53
31	A32	EEG	µV	-823194.84	-5379.44	4124.31	14730.06	791331.98
32	Status	STIM	NA	65280.00	65280.00	65280.00	65280.00	65329.00

3-3- Data Preprocessing

Data preprocessing is a crucial step in developing AI models. The EEG dataset was preprocessed using the MNE package and scikit-learn (sklearn) library. The dataset initially included 45 records from 33 individuals (23 stroke patients and 10 healthy) with 33 features. The preprocessing steps are described below:

- ***Reference Setting*:** The raw EEG data were re-referenced to the average using the set_eeg_reference() function. This step improves the overall signal representation and was applied in all experimental setups except the ICA-based setup (fourth experiment), which separates independent components, allowing artifacts to be identified and removed after decomposition.
- ***Downsampling*:** The raw EEG data were originally sampled at 16,384 Hz, which was computationally intensive. To reduce complexity and computational resource limitations during deep learning training while preserving sufficient information, the data were downsampled to 250 Hz using the resample(250) function. This rate satisfies the Nyquist criterion, as the highest observed frequency was 124 Hz, providing a safe margin while optimizing computational efficiency.
- ***Filtering***: The EEG data were filtered to a frequency range of 0.5–124 Hz using a bandpass filter implemented with the firwin method. An initial range of 1–45 Hz was selected based on prior studies and resources from a relevant GitHub repository [21]. However, this range did not yield satisfactory performance. Therefore, the range was extended to 0.5–124 Hz to capture the critical EEG frequency bands while eliminating irrelevant noise. This adjustment improved model performance by preserving the relevant signal features.

- ***Segmenting the Data*:** Continuous EEG signals were divided into overlapping epochs of 5 seconds with a 10% overlap. Segmenting into fixed-length epochs provides consistent input sizes for DL algorithms and enables temporal analysis. The 10% overlap maximizes data utilization and prevents information loss between consecutive epochs. A Hamming window was applied during segmentation to reduce edge artifacts caused by epoching. The 5-second length was selected to balance empirical experience. Using longer segments would increase input size and computational cost, which would be challenging for deep learning models given the available computational resources. Therefore, shorter epochs were used to ensure efficient training and stable model performance while still preserving sufficient temporal information from the EEG signals.
- ***Spliting the Data*:** The resulting samples were divided into 65% for training, 15% for validation, and 20% for testing. A stratified split was applied to ensure class balance across the training, validation, and testing sets.
- ***Standardization*:** Standardization was performed using the StandardScaler() function from scikit-learn to normalize the features, ensuring that all input variables had a mean of 0 and a standard deviation of 1. This step was applied to all experimental setups.

3-4- Preprocessed Dataset Statistics

EEG datasets are typically represented in a 3D format with shape (samples/records, channels, time points). In this study, 45 EEG recordings were used, each containing 33 channels. The number of time points was calculated by multiplying the duration of the EEG signal (in seconds) by the sampling frequency (in Hz). Before preprocessing, each recording had 33 channels, but the number of time points varied due to differences in signal duration. On average, each recording contained approximately 11,810,133 time points. After applying segmentation for the first, second, and third experiments, the dataset was expanded to 7,150 samples instead of 45. Downsampling, filtering, and segmentation reduced the number of time points per sample to 1,250. In the fourth experiment, which included only downsampling and filtering without segmentation, each sample retained approximately 177,000 time points after preprocessing. Figure 3 shows a sample epoch of EEG signals after preprocessing.

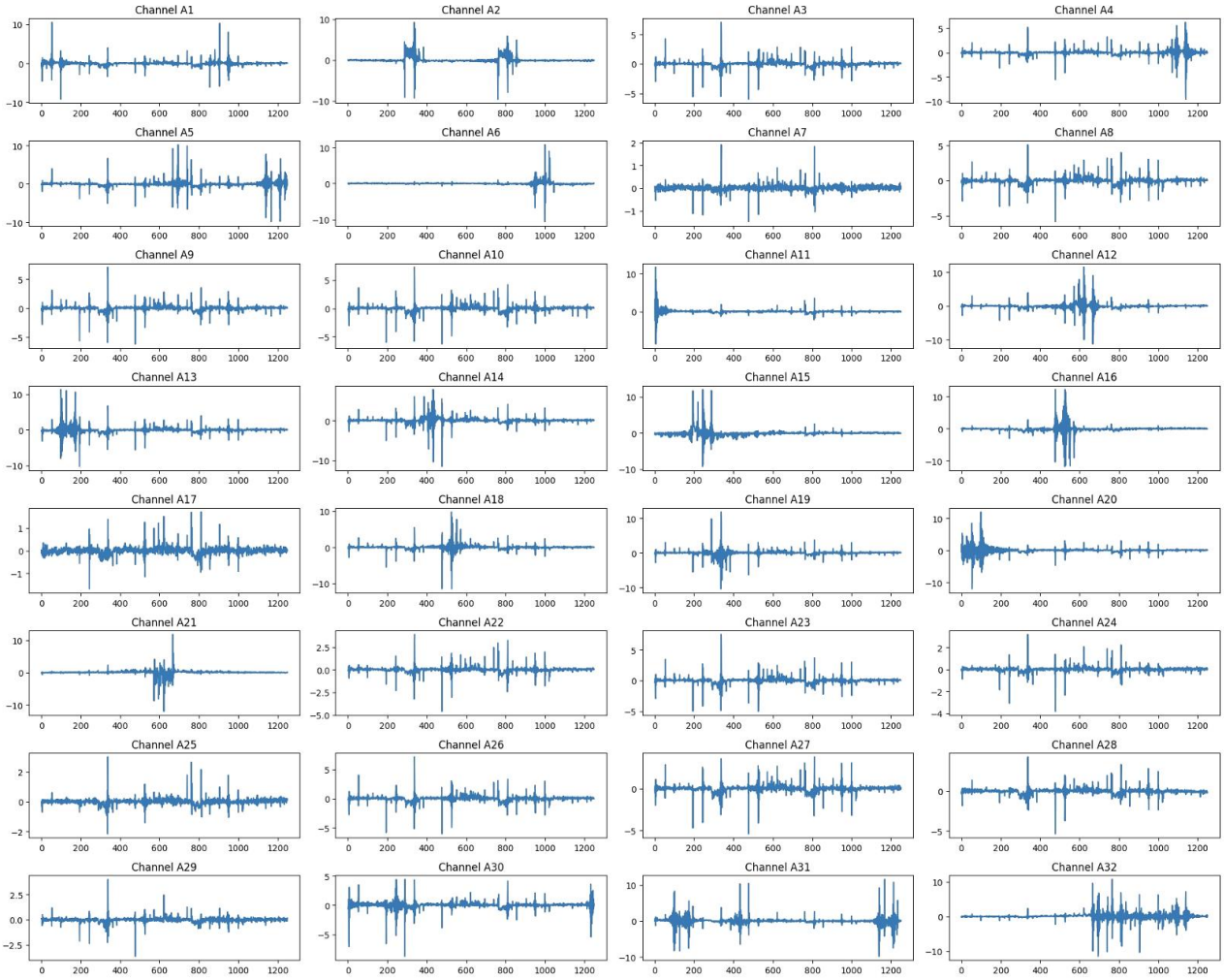


Figure 3. Sample epoch of EEG signals after preprocessing

3-5- Feature Selection Techniques

In the first experimental setup, all 33 channels with 1,250 time points per channel (33 × 1,250 = 41,250 features) were used. The remaining experiments applied feature selection techniques as follows:

3-5-1- Decision Tree (DT)-Based

In the second experimental setup, only features selected by the DT model were used. Since the DT model requires a fixed-length 1D feature vector for each sample, the 2D EEG feature matrix with shape (33, 1,250) was flattened into a 1D vector of length 41,250 features. The feature selection procedure using the DT model involves three main steps. Firstly, a DT classifier was trained on the DL training set. Secondly, the key features were selected from the DT-trained model based on the threshold parameter, which is used to define the feature importance weights. This study utilized three thresholds: 0.01, 0.001, and 0.00001. Only those features with an importance weight greater than the predefined threshold are selected as the key features. The number of selected features was determined based on their importance scores. As shown in Figure 4, the importance scores were plotted, revealing that only 10 features had an importance score greater than 0. Our methodology involved incrementally refining the threshold by increasing its decimal precision and observing its effect on feature selection. In particular, a threshold of 0.01 yielded 6 features, and 0.001 yielded 9 features. Notably, when testing at a threshold of 0.0001, we consistently identified 9 features. Therefore, we increased the threshold to 0.00001, yielding 10 features. Only features with an importance weight exceeding the threshold were retained as key features. Thirdly, the training, validation, and testing datasets are transformed based on the selected features [22].

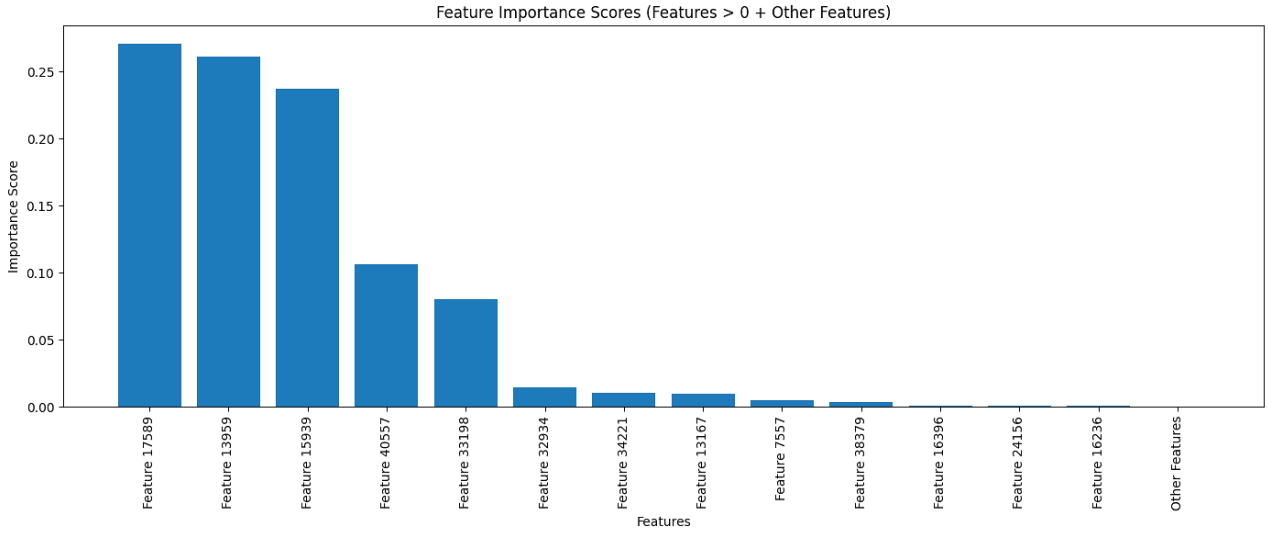


Figure 4. The relationship between the features and their importance scores

The hyperparameters that have been used in the feature selection technique to build the DT model are max depth, min samples leaf, and the criterion. The max depth value is 20, the value of the min samples leaf is 5, and the criterion value is gini. Table 5 shows the number of features selected by DT with each threshold.

Table 5. Features selected by DT

Threshold Value	Number of Selected Features Using DL's Training Set
0.01	5
0.001	7
0.00001	9

3-5-2- Principal Component Analysis (PCA)

In the third experimental setup, the extracted features by PCA are used. PCA is a mathematical technique for reducing the dimensionality of a dataset while preserving most of its important information. To reduce a dataset's dimensionality, PCA converts the original variables (features) into a new set of variables known as principal components, which are linear combinations of the original features. These principal components are orthogonal to one another and are ordered by the amount of variation they explain in the data [23]. The mathematical formulation of PCA is as follows [24]:

$$\mathbf{x}' = \mathbf{A}^{\mathbf{T}}(\mathbf{x} - \boldsymbol{\mu}) \quad (1)$$

where, x original input vector, $\boldsymbol{\mu}$: mean vector, $\mathbf{A}$: matrix of eigenvectors (PCA projection matrix), and $\mathbf{x}'$: projected data in the reduced subspace.

In this experimental setup, PCA was applied to the channel dimension of the EEG dataset, reducing the original 33 channels to either 16 or 30 principal components. However, the time dimension remained unchanged, resulting in 1,250 time-point features per component. Figures 5 and 6 show a sample of the data before and after PCA, respectively.

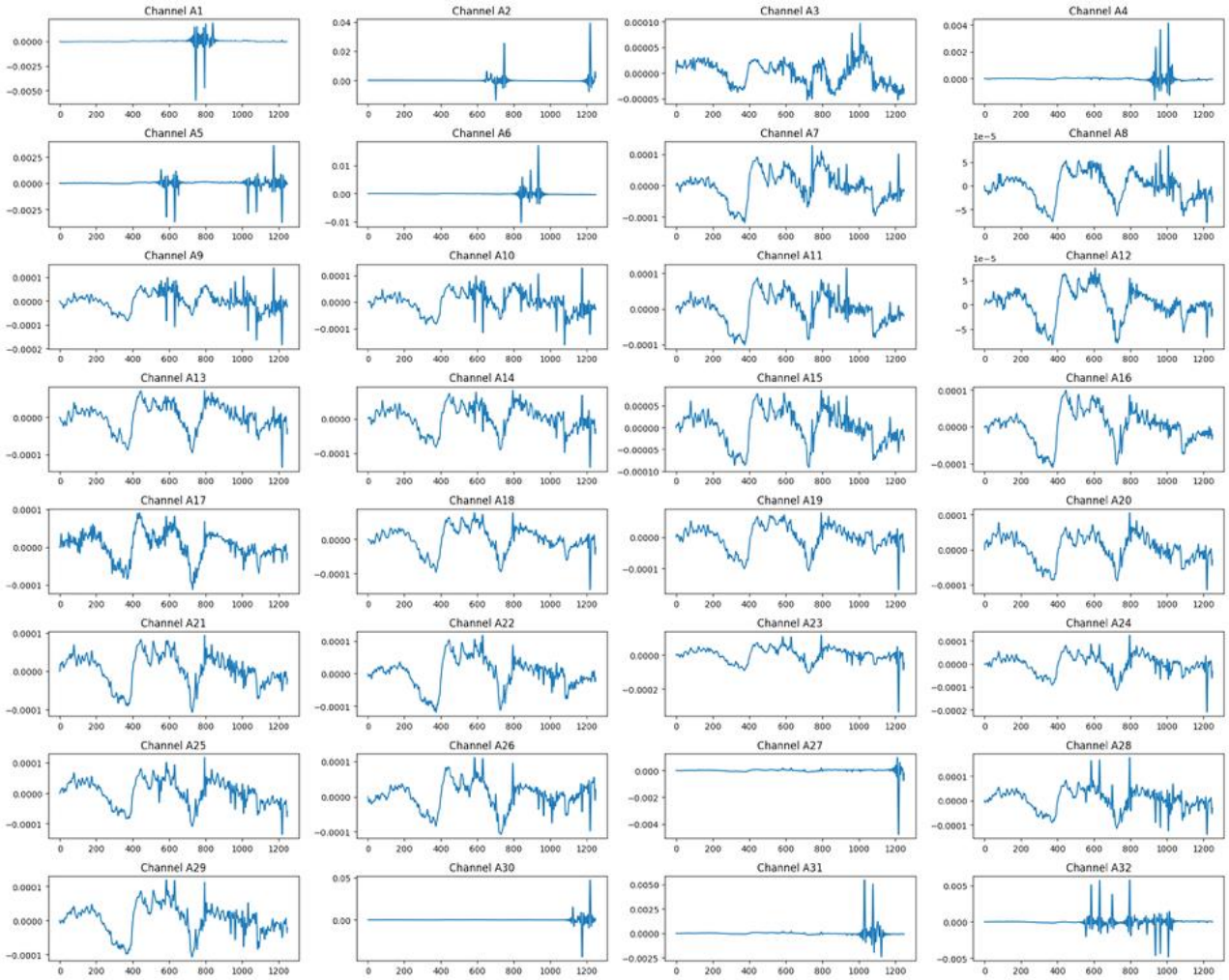


Figure 5. Data sample before PCA

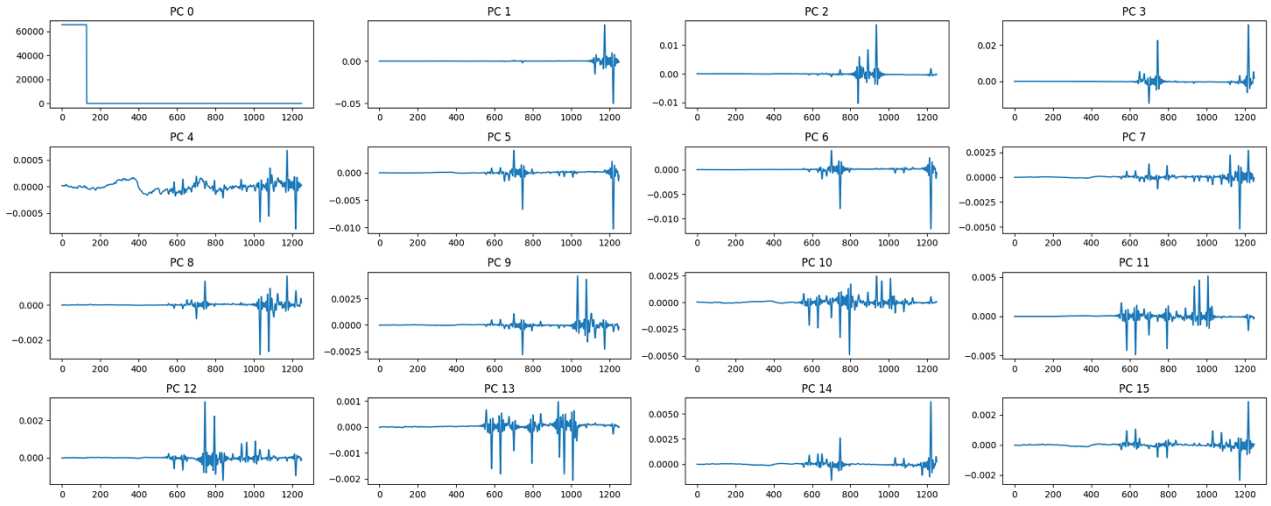


Figure 6. Data sample after PCA

3-5-3- Independent Component Analysis (ICA)

Lastly, in the fourth experimental setup, the features extracted using ICA were used. ICA aims to decompose the data by estimating independent components from raw signals, removing artifacts, extracting relevant features, and reducing dimensionality. For example, when EEG data are recorded, multiple sensors are placed on the subject's scalp, which can lead to overlapping signals across sensors. This is similar to recording in a crowded area with multiple microphones, where various sources of sound are present. In such cases, ICA separates the different signals in the original recording into independent components [25]. The mathematical formulation of ICA is given as follows [26]:

$$x(t) = A\, s(t) \tag{2}$$

where, A is an unknown matrix called the mixing matrix, and $x(t)$, $s(t)$ are the two vectors representing the observed signals and source signals, respectively.

The objective is to recover the original signals, $s_i(t)$, from only the observed vector $x_i(t)$. We obtain estimates for the sources by first obtaining the unmixing matrix W, where

$$W = A^{-1} \tag{3}$$

This enables an estimate, $\hat{s}(t)$ of the independent sources to be obtained:

$$\hat{s}(t) = W\, x(t) \tag{4}$$

In this experiment, the ICA component is set to 32. Figures 7 and 8 show a sample of the data before and after ICA, respectively.

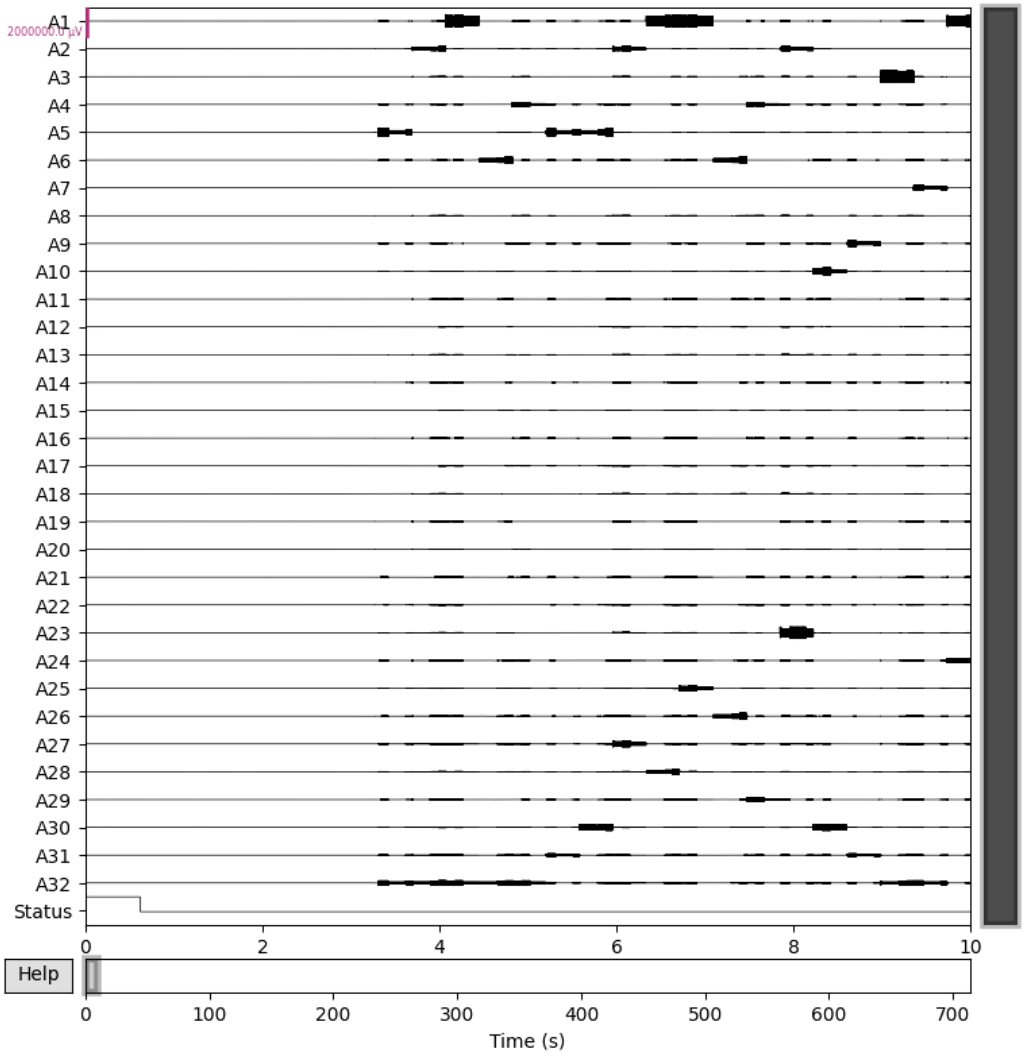


Figure 7. Data sample before ICA

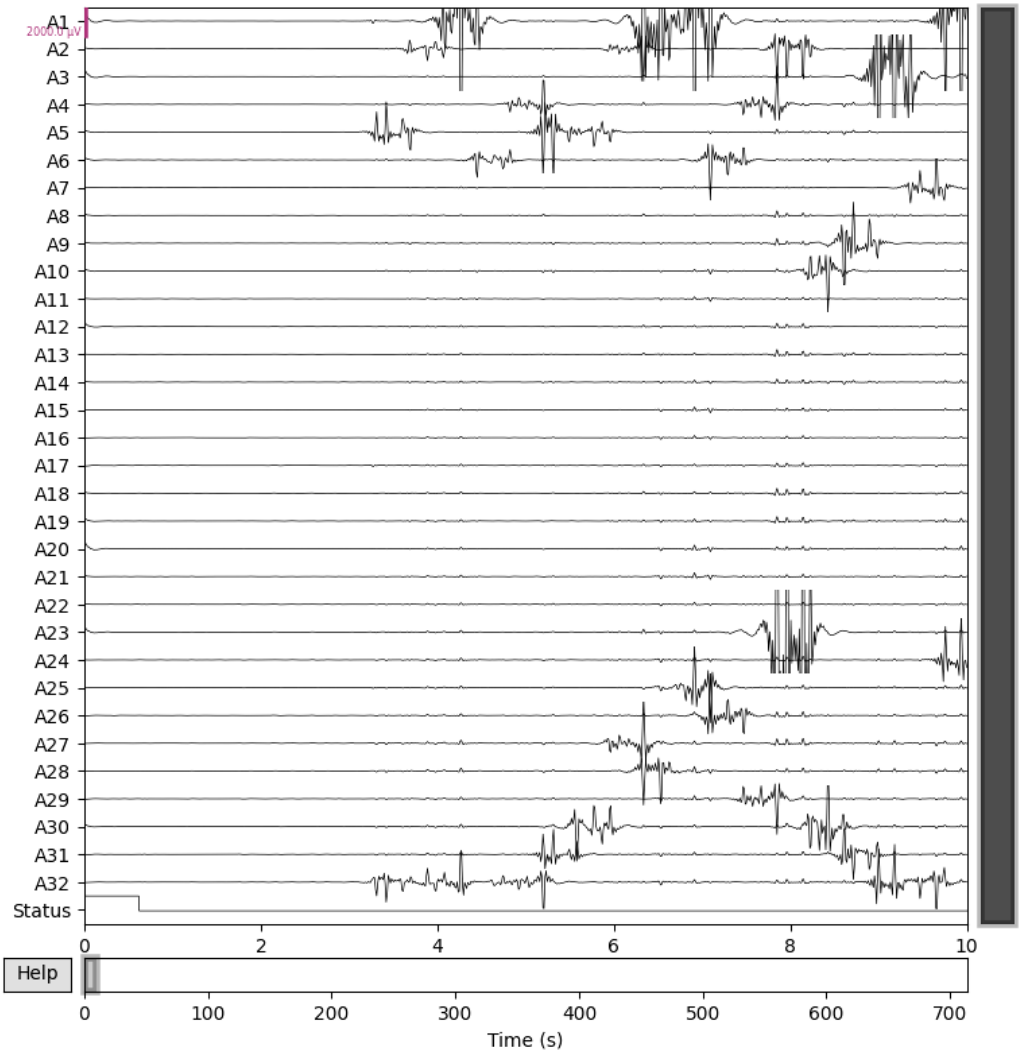


Figure 8. Data sample after ICA

Even though feature selection methods such as DT, PCA, and ICA can reduce dimensionality and potentially improve DL model performance, their computational efficiency and trade-offs should be considered. While DT is relatively efficient and interpretable, it can become computationally intensive for high-dimensional data, especially when deep trees are constructed [27]. PCA, on the other hand, provides a linear transformation that is computationally efficient for moderately sized datasets. However, it may reduce interpretability since the transformed features are linear combinations of the original variables [28]. ICA is robust in identifying statistically independent features, but it is generally more computationally demanding than PCA and DT due to its iterative nature and sensitivity to initial conditions. Moreover, ICA assumes statistical independence, which may not always hold in real-world datasets [25].

3-6- Description of DL Algorithms

This study utilized several fundamental DL algorithms, including RNN, GRU, LSTM, and CNN. These algorithms were selected based on their demonstrated effectiveness in stroke prediction in previous studies. Additionally, the models were evaluated using three distinct feature extraction methods, as well as using the complete set of features. Although other DL architectures, such as Transformer-based models, are commonly used for time-series data, they generally require large datasets, which was not applicable for the dataset used in this study.

3-6-1- Convolutional Neural Networks (CNN)

CNNs are a type of DL algorithm designed for data with a grid-like structure, particularly well-suited for image, audio, and signal inputs. They have a hierarchical structure consisting of multiple layers. The input layer receives the raw data, followed by convolutional layers that apply filters to detect local patterns and features in the input data. Activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity. Pooling layers reduce spatial dimensions and extract the most essential features, often using max pooling. Fully connected layers connect every neuron

to the next layer for classification and prediction, while the output layer provides the final classification result. CNNs excel at spatial invariance, allowing them to detect patterns regardless of their location in the signal, making them particularly valuable for shift-invariant data. They generate a hierarchical representation of features, enabling the identification of complex patterns. CNNs eliminate the need for manual feature engineering, as they can learn and adapt to the unique characteristics of the data. This automation simplifies signal analysis, improves accuracy in tasks such as classification and regression, and makes CNNs highly effective for signal processing applications [29]. The mathematical formulation of CNN is given as follows [30]:

$$Z_{t,f} = \sum_{i=0}^{K-1} \sum_{c=0}^{C_{in}-1} X_{t+i,c} \cdot K_{i,c,f} + b_f \quad (5)$$

$$A_{\{t,f\}} = \sigma\left(Z_{\{t,f\}}\right) \quad (6)$$

where, $X \in \mathbb{R}^{T \times C_{in}}$ is the input signal with T time steps and C_{in} input channels, $X \in \mathbb{R}^{T \times C_{in} \times C_{out}}$ is the convolution kernel with filter size k, b_f is the bias for output filter f, $Z_{t,f}$ is the pre-activation output at time step t for filter f, $A_{t,f}$ is the activation output after applying non-linearity, σ is an activation function such as ReLU: $\sigma(z) = \max(0, z)$.

3-6-2- Recurrent Neural Networks (RNN)

RNNs are a type of DL algorithm designed for sequence data processing. They differ from Feed-Forward Neural Networks (FFNNs) by having connections that form a directed cycle, allowing the network to maintain a hidden state. RNNs are particularly useful for tasks that require understanding temporal context, such as natural language processing (NLP) and time-series prediction. An RNN consists of an input layer, one or more hidden layers, and an output layer. The input layer processes the input sequence, with each element represented as a feature vector. The hidden layer(s) maintain a hidden state that evolves as the network processes the sequence. The structure of the hidden layer(s) can vary, including specialized architectures such as LSTM or GRU. The output layer produces the final prediction, which can be a forecast for each time step or a single prediction for the entire sequence. Traditional RNNs, however, face challenges with long-term dependencies due to the vanishing gradient problem during training. Advanced architectures like LSTM and GRU include mechanisms to selectively retain and forget information, addressing this issue and enhancing performance on tasks involving long-term dependencies [31]. The mathematical formulation of the RNN at time step t is given by [32]:

$$h_t = \tan h\left(W_{\{xh\}x_t} + W_{\{hh\}h_{\{t-1\}}} + b_h\right) \quad (7)$$

$$y_t = W_{\{hy\}h_t} + b_y \quad (8)$$

where, $X_t \in \mathbb{R}^C$ is the input signal vector at time step t (with C chan- nels/features), $h_t \in \mathbb{R}^H$ is the hidden state at time t, $y_t \in \mathbb{R}^k$ is the output at time t, $W_{xh} \in \mathbb{R}^{H \times C}$: weights from input to hidden state, $W_{hh} \in \mathbb{R}^{H \times H}$: recurrent weights, $W_{hy} \in \mathbb{R}^{K \times H}$: hidden to output weights, $b_h \in \mathbb{R}^H$, $b_y \in \mathbb{R}^K$: bias vectors, and tanh is the activation function applied element-wise.

3-6-3-Long Short-Term Memory (LSTM)

LSTM is an extension of RNN, primarily presented to address problems where RNN fails. The primary difference between the LSTM architecture and the RNN architecture is the hidden layer of the LSTM, which is a gated cell. LSTM's structure consists of a cell to retain information and three gates: the input gate, the forget gate, and the output gate. Gates are presented to restrict the information that is passed through the cell. Similar to GRUs, LSTMs also address the vanishing gradient problem [33]. The mathematical formulation of an LSTM cell at time step t is as follows [15]:

$$f_t = \sigma\left(W_f \cdot \left[h_{\{t-1\}}, x_t\right] + b_f\right) \quad (9)$$

$$i_t = \sigma\left(W_i \cdot \left[h_{\{t-1\}}, x_t\right] + b_i\right) \quad (10)$$

$$i_t = \sigma\left(W_i \cdot \left[h_{\{t-1\}}, x_t\right] + b_i\right) \quad (11)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C_t} \quad (12)$$

$$o_t = \sigma\left(W_o \cdot \left[h_{\{t-1\}}, x_t\right] + b_o\right) \quad (13)$$

$$h_t = o_t \odot tanh(C_t) \quad (14)$$

where, x_t is the input vector at time t, $h_{\{t-1\}}$ is the hidden state from the previous time step, f_t, i, and o_t denote the forget, input, and output gates, respectively, $\widetilde{C_t}$ is the candidate cell state, C_t is the updated cell state, σ is the sigmoid activation function, tanh is the hyperbolic tangent activation function, $\odot$ denotes element-wise multiplication, W_f , W_i, W_C, and W_o are the weight matrices, and b_f , b_i, b_C, and b_o are the bias vectors.

3-6-4- Gated Recurrent Unit (GRU)

GRU is an advancement above standard RNNs. It implements gating methods to control information flow within the network. GRUs have two gates: the reset gate and the update gate. The reset gate determines what information to discard from the previous state, while the update gate determines what new information to keep. This enables GRUs to capture dependencies across longer sequences more effectively than typical RNNs, handling the issue of vanishing gradients and making them a popular choice for sequential data problems [34]. The mathematical formulation at time step *t* is given by [35]:

$$z_t = \sigma\left(W_z \cdot \left[h_{\{t-1\}}, x_t\right] + b_z\right) \quad (15)$$

$$r_t = \sigma\left(W_r \cdot \left[h_{\{t-1\}}, x_t\right] + b_r\right) \quad (16)$$

$$\widetilde{h_t} = tanh\left(W_h \cdot \left[r_t \odot h_{\{t-1\}}, x_t\right] + b_h\right) \quad (17)$$

$$h_t = (1 - z_t) \odot h_{\{t-1\}} + z_t \odot \widetilde{h_t} \quad (18)$$

where, x_t is the input vector at time *t*, $h_{\{t-1\}}$ is the hidden state from the previous time step, z_t is the update gate vector, rt is the reset gate vector, $\widetilde{h_t}$ is the candidate hidden state, ht is the final hidden state at time *t*, σ denotes the sigmoid activation function, *tanh* denotes the hyperbolic tangent activation function, $\odot$ represents element-wise multiplication, W_z, W_r, W_h are weight matrices, and b_z, b_r, b_h are the corresponding bias vectors.

3-6-5- Convolutional Neural Network–Long Short-Term Memory (CNN-LSTM)

In this hybrid model, the advantages of CNN and LSTM are combined. The CNN effectively captures local features and patterns within the time series, while the LSTM utilizes both historical and contextual information to enhance sequence prediction. This combined approach enables more accurate classification and forecasting of complex time-series data [36, 15].

3-6-6- Convolutional Neural Network–Gated Recurrent Unit (CNN-GRU)

In this hybrid model, the CNN provides powerful feature extraction capabilities, while the GRU effectively models sequential dependencies in time-series data, such as EEG signals. Both LSTM and GRU can retain information from previous time steps, but GRUs have only two gates (update and reset), making them computationally more efficient and faster to converge than LSTMs. Additionally, GRUs are less complex than LSTMs, which can help reduce overfitting, particularly in smaller datasets [37-39].

3-7- Deep Learning Models Architectures

This section will be divided into five subsections for each experiment: CNN model architectures, LSTM model architectures, CNN-LSTM model architectures, CNN-GRU model architectures, and RNN Model's architecture.

3-7-1- CNN Model Architectures

The CNN architectures used in the first and second experiments are identical, except for differences in input shapes and hyperparameters, as shown in Figure 9-a. The architecture consists of an input layer with an input shape of (1250, 33), where 1250 is the length of the signal, and 33 is the number of channels. However, each experiment's input shape might differ based on the input data. The input layer is followed by three blocks, where each block comprises three 1D convolution layers and a Filter Concat layer that concatenates the output of 1D convolution layers. Additionally, the three blocks are followed by two CNN layers with a ReLU activation function for feature extraction, followed by a max pooling layer for dimensionality reduction. Subsequently, the output of the max pooling layer is flattened into a one-dimensional tensor. Lastly, the flattening layer's output becomes the dense layer's input with a sigmoid function for binary classification. Two activation functions were used: ReLU and Sigmoid activation functions. The ReLU was used in the 1D convolution layers, and the Sigmoid was used in the prediction layer. In addition, the Adam optimizer was used with a learning rate of 0.0001. Furthermore, the binary cross-entropy is used as the loss function.

(a) (b) (c)

Figure 9. The CNN model architecture for the: (a) first and second experiments; (b) third experiment; (c) fourth experiment

The applied CNN model architecture in the third experiment (using PCA) is illustrated in Figure 9-b. The model begins with an input layer that takes an input shape of (1250, 16), indicating the signal length and the number of channels. This input configuration can vary based on the specifics of the dataset used. After the input, the architecture includes three blocks, each consisting of three 1D convolution layers. The outputs of these convolution layers within each block are merged using a concatenate layer, which helps combine feature maps to capture information from different perspectives at each stage. The concatenated output from the last block then passes through two additional 1D convolution layers, enhancing the feature extraction further. Following these layers, a max pooling layer is employed to reduce the spatial dimensions of the feature maps, effectively summarizing the most important features while reducing the computational complexity. The output from the max pooling layer is then flattened into a one-dimensional tensor, suitable for the final classification stage. This flattened tensor is fed into a dense layer that utilizes a sigmoid activation function to output a binary classification result. The model utilizes ReLU activation functions in all convolutional layers to introduce non-linearity, enabling the model to learn complex patterns in the data. The sigmoid activation function in the final layer is specifically chosen for binary classification tasks. For optimization, the model uses the Adam optimizer with a learning rate of 0.0001, and binary cross-entropy serves as the loss function to evaluate the model's performance. This architecture is particularly suitable for applications involving sequential data analysis for classification purposes.

The applied CNN model architecture in the fourth experiment (using ICA) is illustrated in Figure 9-c. It is designed for efficient processing of small-scale input data. The architecture starts with an input layer accepting data of shape (32, 32), which is suitable for handling compact input arrays. The model features three 1D convolutional layers, each progressively reducing the spatial dimension of the input while maintaining 32 feature maps. Specifically, the first convolutional layer halves the input length to 16, the second layer reduces it further to 8, and the third layer maintains the length at 8. This configuration allows for detailed feature extraction at each stage. Following the convolutional layers, a max pooling layer is applied, which reduces the dimensionality to (4, 32) by selecting the maximum value over a window of 2 units. This layer helps in reducing the computational load while retaining the most significant features. The output from the max pooling layer is then flattened into a vector of 128 features, preparing it for the final classification stage. A dense layer with a single unit and a sigmoid activation function is used to produce a binary output, determining the class of the input data based on the extracted features. This compact and sequential CNN architecture is optimized for applications that require rapid processing of small data arrays, leveraging convolution and pooling layers to distill essential features before classification effectively

3-7-2- LSTM Model Architectures

The LSTM model architectures of the first and second experiments are identical except for input shapes and hyperparameters, as shown in Figure 10-a. The architecture consists of an input layer with an input shape of (1250, 33), where 1250 is the length of the signals, and 33 is the number of channels. However, the input shape may differ

in each experiment based on the input data. The input layer is followed by two LSTM layers, each with 100 units and a dropout of 0.2. The last layer is the dense layer with a sigmoid function for binary classification. Additionally, the Adam optimizer was used with a learning rate of 0.001. Furthermore, the binary cross-entropy is used as the loss function.

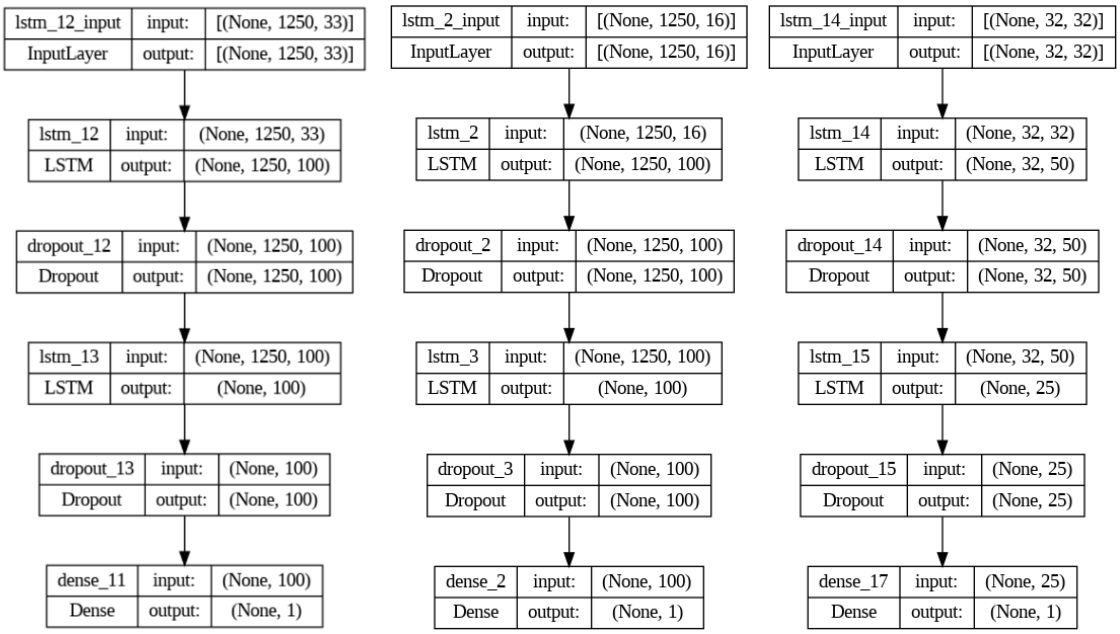


Figure 10. The LSTM model architecture for the: (a) first and second experiments; (b) third experiment; (c) fourth experiment

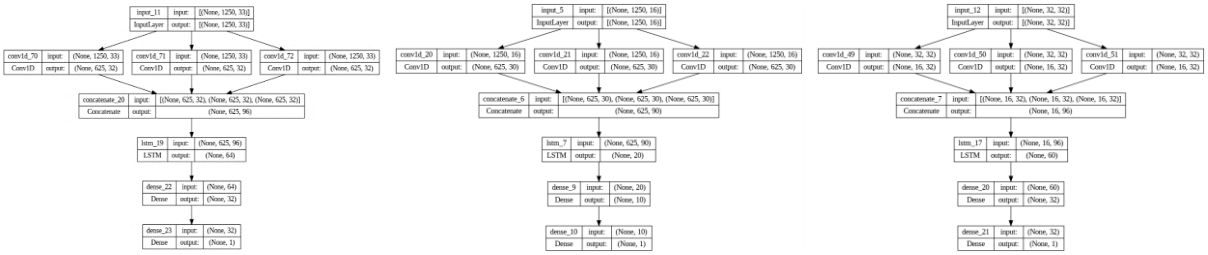
The LSTM model architecture in the second experiment (using PCA) is depicted in Figure 10-b. The architecture consists of an input layer with an input shape of (1250, 16), where 1250 represents the length of the signals, and 16 is the number of features per timestep. Following the input layer, the model includes an LSTM layer with 100 units designed to capture long-term dependencies within the signal data. This LSTM layer is followed by a dropout layer to prevent overfitting by randomly omitting a fraction of the units during training, enhancing the model's ability to generalize to new data. Subsequent to this, another LSTM layer with 100 units further processes the signal, extracting deeper temporal features. This is followed again by a dropout layer to maintain robustness against overfitting. The final component of the model is a dense layer with a single output unit employing a sigmoid activation function suitable for binary classification tasks. The model uses hyperbolic tangent (tanh) as the activation function in the LSTM layers to process data nonlinearly and a sigmoid activation in the dense layer for the binary output. Additionally, the Adam optimizer with a learning rate of 0.0001 is used to optimize the training process. Binary cross-entropy serves as the loss function, evaluating the model's performance in classifying signals into two categories. This LSTM model structure is particularly effective for analyzing and predicting patterns in time series data or other sequential datasets.

The LSTM model architecture used in the third experiment is illustrated in Figure 10-c. The model begins with an input layer of shape (32, 32), where 32 represents the sequence length, and 32 denotes the number of features per timestep. This is followed by an LSTM layer with 50 units that captures temporal dependencies in the sequential data and returns sequences. A dropout layer is then applied to reduce overfitting. Subsequently, a second LSTM layer with 25 units further processes the extracted temporal features and outputs the final hidden representation. Another dropout layer is used to improve generalization. Finally, a dense layer with a single neuron produces the output for binary classification, typically using a sigmoid activation function. The LSTM layers employ the tanh activation function, and the model is commonly trained using the Adam optimizer with binary cross-entropy loss.

3-7-3- CNN-LSTM Model Architectures

The CNN-LSTM model has been chosen based on its promising results in Choi et al. [15]. The applied CNN-LSTM model architectures of the first and second experiments are identical except for input shapes and hyperparameters, as shown in Figure 11-a. The architecture consists of an input layer with an input shape of (1250, 33), where 1250 is the

length of the signal, and 33 is the number of channels. However, each experiment's input shape might differ based on the input data. The input layer is followed by a block consisting of three 1D convolution layers and a concat layer that concatenates the output of 1D convolution layers. Furthermore, the Concatenate layer is followed by one LSTM layer and two dense layers with ReLU and sigmoid activation functions, respectively. Lastly, the binary cross-entropy is used as the loss function.



(a) (b) (c)

Figure 11. The CNN-LSTM model architecture for the: (a) first and second experiments; (b) third experiment; (c) fourth experiment

The CNN-LSTM model architecture of the third experiment (using PCA) is depicted in Figure 11-b. It is designed for sequential data processing, combining convolutional and recurrent layers to enhance feature extraction and temporal analysis. The architecture begins with an input layer that takes data with dimensions of (1250, 16), indicative of the signal length and feature count. Three 1D convolutional layers process the input in parallel, each producing an output of (625, 30). These layers are adept at detecting localized feature patterns within the signal. The outputs from these convolutional layers are merged through a concatenate layer, forming a comprehensive feature map with dimensions (625, 90). This merging facilitates a holistic view of the input features, capturing a broad spectrum of data characteristics. Subsequently, this concatenated output is fed into an LSTM layer with 20 units designed to capture the data's long-term dependencies and temporal relationships. The LSTM's output is then directed through two dense layers: the first has 10 units to condense the features further, and the second, a single unit with a sigmoid activation function, computes the final classification output. This hybrid model structure, combining convolutional layers for initial feature extraction and an LSTM layer for temporal dynamics, along with dense layers for output generation, is effective for complex signal classification tasks where both spatial and temporal insights are crucial.

The CNN-LSTM model architecture of the fourth experiment (using ICA) is depicted in Figure 11-c. It integrates convolutional and recurrent layers and is designed for efficient feature extraction and signal processing. Starting with an input layer configured for inputs of shape (None, 32, 32), the model utilizes three parallel Conv1D layers to analyze the data independently, each halving the signal length to 16 while maintaining 32 feature channels. These Conv1D layers' outputs are then merged via a concatenate layer, combining the feature maps to create a more comprehensive representation with dimensions (None, 16, 96). This concatenated output feeds into an LSTM layer with 60 units, enabling the model to effectively capture temporal dependencies and signal dynamics. Following the LSTM, the network includes two dense layers. The first dense layer, with 32 units, condenses the LSTM's output, preparing it for final classification. The subsequent dense layer reduces this to a single output with a sigmoid activation function aimed at binary classification. This model structure is optimal for applications requiring detailed analysis of both spatial and temporal data characteristics, such as in time series classification or complex pattern recognition tasks, leveraging the combined strengths of convolutional and recurrent neural network technologies.

3-7-4- CNN-GRU Model Architectures:

The architectures of the first and second experiments are identical except for input shapes and hyperparameters, as shown in Figure 12-a. This model architecture is based on a model proposed by Roy et al. [40] called the Inception Convolutional Densely Connected Gated Recurrent Neural Network (ChronoNet) architecture. Two changes were applied to the proposed ChronoNet architecture: the shape of the input layer and the activation function in the prediction layer. The architecture consists of an input layer with an input shape of (1250, 33), where 1250 is the length of the signal, and 33 is the number of channels. However, the input shape may differ in each experiment based on the input data. Following the input layer, the model includes three sequential blocks, each consisting of three parallel 1D convolution layers, followed by a concatenation layer that merges the outputs from these convolution layers. After the third block, the model features a series of GRU layers. The output from the last block is fed into the first GRU layer, which is followed by additional GRU layers with their outputs concatenated at different stages to increase the model's capacity to capture temporal dependencies at multiple scales. The final GRU layer's output is then fed into a dense layer with a sigmoid activation function for binary classification.

(a) (b) (c)

Figure 12. The CNN-GRU model architecture for the: (a) first and second experiments; (b) third experiment; (c) fourth experiment

The CNN-GRU model architecture of the third experiment (using PCA) is depicted in Figure 12-b. It is designed for advanced signal processing, utilizing both convolutional and recurrent layers. The network starts with an input layer that receives data with dimensions (1250, 16), ideal for handling signals with multiple features. Initially, the input passes through three parallel 1D convolutional layers, each producing outputs with dimensions (625, 32). These layers are adept at extracting localized features from the input signals. Following the initial convolutional stage, a concatenate layer merges the outputs, forming a feature map with dimensions (625, 96). This feature map undergoes further convolution through three pairs of convolutional layers, with each pair's output being concatenated, progressively refining and enhancing the feature extraction.

The refined features are then processed through a series of GRU (Gated Recurrent Unit) layers. The first GRU layer takes the concatenated feature map with dimensions (157, 96) and reduces it to (157, 32). Two subsequent GRU layers further process this output, with intermediate concatenation steps that help preserve and emphasize important temporal features. The final output from the GRU layers passes through another dense layer with a single unit, employing a sigmoid activation function for binary classification. This configuration allows the model to effectively integrate spatial features extracted by convolutional layers with temporal dynamics captured by GRU layers, making it particularly suited for tasks that require a slight understanding of time-dependent data.

The CNN-GRU model architecture of the fourth experiment (using ICA) is depicted in Figure 12-c. It combines convolutional and recurrent layers tailored for efficient signal processing. It starts with an input layer that accepts input arrays shaped (None, 32, 32), ideal for data that represents signals of length 32 with 32 features each. The model employs two parallel Conv1D layers to process the input, both reducing the signal length to 16 while maintaining 32 feature channels. These layers are effective at extracting spatial features from each timestep of the input signal.

Outputs from these convolutional layers are then merged using a concatenate layer, which combines the feature maps into a single tensor of dimensions (None, 16, 64), enhancing the feature representation by integrating information from both processing streams. This concatenated output is then processed by a GRU layer with 32 units, which helps to capture temporal dependencies and dynamics from the enhanced feature set. Following the GRU

layer, a dense layer with a single output unit and a sigmoid activation function finalizes the model, providing a binary classification of the input signal. This architecture is particularly well-suited for applications that require an understanding of both the spatial features and temporal signals of the data, such as in time series analysis or sequential event prediction.

3-7-5- RNN Model Architectures

The architectures of the first and second experiments are identical except for input shapes and hyperparameters, as shown in Figure 13-a. The architecture consists of an input layer with an input shape of (1250, 33), where 1250 is the length of the signal, and 33 is the number of channels. However, the input shape may differ in each experiment based on the input data. The input layer is followed by two SimpleRNN layers with 128 units. The last layers are two dense layers, one with a ReLU activation function and the other with a sigmoid activation function for binary classification. Additionally, the Adam optimizer was used with a learning rate of 0.0001. Furthermore, binary cross-entropy is used as the loss function.

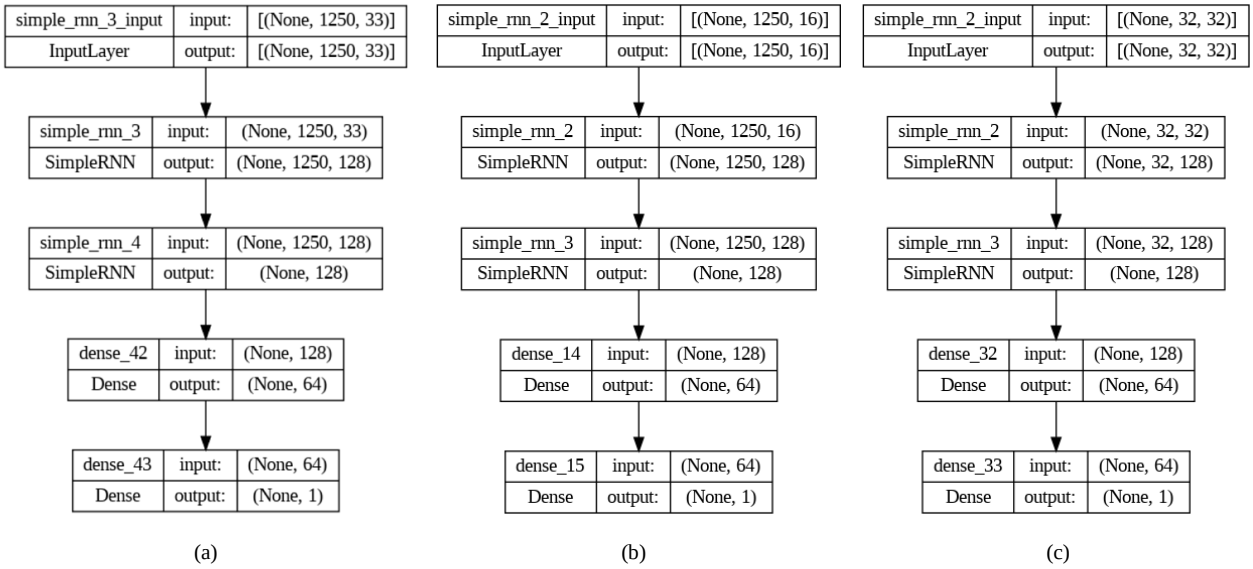


Figure 13. The CNN-RNN model architecture for the: (a) first and second experiments; (b) third experiment; (c) fourth experiment

The RNN model architecture of the third experiment (using PCA) is illustrated in Figure 13-b. The network starts with an input layer that accepts signals with shape of (1250, 16), where 1250 represents the signal length and 16 is the feature dimension. The first layer in the model, a SimpleRNN, processes the entire signal with 128 units, capturing temporal dependencies within the input. The output from this initial RNN layer is then fed into another SimpleRNN layer. Subsequent to the recurrent layers, the model incorporates two dense layers. The first dense layer has 64 units, further refining the features prepared by the RNN layers. The final dense layer, consisting of a single unit with a sigmoid activation function, outputs a probability, classifying the signal into one of two categories.

The RNN model architecture of the fourth experiment (using ICA) is illustrated in Figure 13-c. It is simplified for signal processing, starting with an input layer configured to handle data with shape (None, 32, 32). This setup indicates each signal is 32 timesteps long, with 32 features per timestep. The model comprises two sequential SimpleRNN layers. The first SimpleRNN layer, with 128 units, processes the entire signal, capturing the initial temporal dependencies and feature transformations. It outputs a signal of the same length (32) but with 128 features per timestep. The second SimpleRNN layer, also with 128 units, further analyzes the signal, this time returning only the output from the final timestep. This reduction simplifies the data to a single 128-dimensional vector, focusing the model on the end-of-signal characteristics crucial for prediction tasks. Following the recurrent layers, the model features two dense layers. The first dense layer has 64 units and further condenses the features, preparing them for the final output. The last layer, a dense layer with a single unit and a sigmoid activation function, provides the probability output for binary classification.

3-8- Performance Measure

The performance of the developed models will be evaluated based on several performance evaluation measures such as accuracy, precision, recall, confusion matrix, and F1-score.

3-8-1- Accuracy

The accuracy measures the percentage of correctly classified instances (stroke or non-stroke) out of the total number of instances. The formula is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

3-8-2- Precision

The precision measures the proportion of correctly predicted stroke instances out of the total instances predicted as stroke. The formula is as follows:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{20}$$

3-8-3- Recall

The recall measures the proportion of correctly predicted stroke instances out of the total actual stroke instances. The formula is as follows:

$$\text{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}} \tag{21}$$

3-8-4- Confusion Matrix

The confusion matrix is a visualization of the model predictions in a matrix. It shows true positives, true negatives, false positives, and false negatives. The matrix is as follows (Table 6):

Table 6. Confusion Matrix

	Actual Positive	**Actual Negative**
Predicted Positive	TP	FP
Predicted Negative	FN	TN

3-8-5- F1-Score

The F1 score combines both precision and recall, providing a single metric that balances the trade-off between the two. It is a popular metric for imbalanced datasets because it evaluates the model's performance while accounting for class imbalance and provides a comprehensive assessment of the model's ability to correctly identify stroke cases while minimizing errors. The formula is as follows:

$$\text{F1} = \frac{\text{Precision} \,.\, \text{Recall}}{\text{Precision+ Recall}} \tag{22}$$

3-9- Optimization Strategy

GridSearchCV is used to find the optimal hyperparameters that achieve the best results. It works by first passing predefined values to GridSearchCV. Then, GridSearchCV tries all possible combinations based on the values provided and selects the optimal combination of hyperparameters [41].

In this study, GridSearchCV is employed to identify the optimal hyperparameters. The searched hyperparameters for each model are shown in Table 7. However, due to memory limitations in the online environment, GridSearchCV was unable to complete the full search. Nevertheless, the partial results obtained were sufficient to narrow down the range of promising hyperparameter values. Based on these preliminary outcomes, the final optimal hyperparameters were determined through manual tuning.

Table 7. The GridSearchCV searched hyperparameters

Model	Hyperparameter	Values
CNN	Optimizer	(Adam, SGD)
	Learning rate	(0.00001, 0.0001, 0.001)
	Activation function	(ReLU, Sigmoid, Tanh)
	Batch_size	(132, 128, 64)
LSTM	Optimizer	(Adam, SGD)
	Learning rate	(0.0001, 0.001, 0.01, 0.1)
	Activation function	(Sigmoid, ReLU)
	Batch_size	(132, 128, 64)
CNN-LSTM	Optimizer	(Adam, SGD)
	Learning rate	(0.0001, 0.001, 0.01)
	Activation function	(ReLU, Sigmoid, Tanh)
	Batch_size	(132, 128, 64)
CNN-GRU	Optimizer	(Adam, SGD)
	Learning rate	(0.00001, 0.0001, 0.001, 0.01)
	Activation function	(ReLU, Sigmoid, Tanh)
	Batch_size	(132, 128, 64)
RNN	Optimizer	(Adam, SGD)
	Learning rate	(0.000001, 0.00001, 0.0001, 0.001)
	Activation function	(ReLU, Sigmoid, Tanh)
	Batch_size	(132, 128, 64)

The optimal parameters found for the first and second experiments (using all features) are shown in Table 8. The optimal parameters found for the third and fourth experiments are shown in Tables 9 and 10, respectively.

Table 8. The optimal hyperparameters of the first experimental setup (using all features)

Model	Hyperparameter	Values
CNN	Optimizer	Adam
	Learning rate	0.0001
	Activation function	ReLU and Sigmoid
	Batch_size	64
LSTM	Optimizer	Adam
	Learning rate	0.01
	Activation function	Sigmoid
	Batch_size	128
CNN-LSTM	Optimizer	Adam
	Learning rate	0.001
	Activation function	ReLU and Tanh and Sigmoid
	Batch_size	64
CNN-GRU	Optimizer	Adam
	Learning rate	0.001
	Activation function	ReLU and Tanh and Sigmoid
	Batch_size	64
RNN	Optimizer	Adam
	Learning rate	0.0001
	Activation function	ReLU and Sigmoid
	Batch_size	64

Table 9. The optimal hyperparameters of the third experimental setup (using PCA with 16 and 30 features)

Model	16 Features		30 Features	
	Hyperparameter	Values	Hyperparameter	Values
CNN	Optimizer	Adam	Optimizer	Adam
	Learning rate	0.0001	Learning rate	0.0001
	Activation function	ReLU and Sigmoid	Activation function	ReLU and Sigmoid
	Batch_size	64	Batch_size	64
LSTM	Optimizer	Adam	Optimizer	Adam
	Learning rate	0.001	Learning rate	0.01
	Activation function	Sigmoid	Activation function	Sigmoid
	Batch_size	128	Batch_size	64
CNN-LSTM	Optimizer	Adam	Optimizer	Adam
	Learning rate	0.0001	Learning rate	0.001
	Activation function	ReLU and Sigmoid	Activation function	ReLU, Tanh and Sigmoid
	Batch_size	64	Batch_size	64
CNN-GRU	Optimizer	Adam	Optimizer	Adam
	Learning rate	0.001	Learning rate	0.001
	Activation function	ReLU and Tanh and Sigmoid	Activation function	ReLU, Tanh and Sigmoid
	Batch_size	64	Batch_size	64
RNN	Optimizer	Adam	Optimizer	Adam
	Learning rate	0.001	Learning rate	0.0001
	Activation function	ReLU and Sigmoid	Activation function	ReLU and Sigmoid
	Batch_size	64	Batch_size	64

Table 10. The optimal hyperparameters of the fourth experimental setup (using ICA)

Model	Hyperparameter	Values
CNN	Optimizer	Adam
	Learning rate	0.001
	Activation function	ReLU and Sigmoid
	Batch_size	10
LSTM	Optimizer	Adam
	Learning rate	0.001
	Activation function	Sigmoid
	Batch_size	128
CNN-LSTM	Optimizer	Adam
	Learning rate	0.001
	Activation function	ReLU and Tanh and Sigmoid
	Batch_size	64
CNN-GRU	Optimizer	Adam
	Learning rate	0.0001
	Activation function	ReLU and Tanh and Sigmoid
	Batch_size	32
RNN	Optimizer	Adam
	Learning rate	0.001
	Activation function	ReLU and Sigmoid
	Batch_size	64

4- Results

This section presents the results of the proposed models based on the experimental setups. According to the findings of the first experiment, as shown in Table 11, CNN and CNN-GRU showed the highest accuracy rate of 70%. They also achieved 70% precision, 70% recall, and a 70% F1-score.

Table 11. The results of the first experiment (using all features

Model	Accuracy	Precision	Recall	F1-score
CNN	**70%**	**70%**	**70%**	**70%**
LSTM	66%	67%	66%	67%
CNN-LSTM	68%	69%	68%	68%
CNN-GRU	**70%**	**70%**	**70%**	**70%**
RNN	63%	60%	63%	61%

Moving on to the second experiment, as shown in Table 12, the best result was achieved by the CNN-LSTM model with seven features, yielding 86% accuracy, 99% precision, 81% recall, and an 89% F1-score.

Table 12. The results of the second experiment across different feature sets (Using DT)

Model	Accuracy	Precision	Recall	F1
5 Features				
CNN	56%	64%	81%	72%
LSTM	58%	69%	69%	69%
CNN-LSTM	67%	73%	83%	77%
CNN-GRU	58%	69%	69%	69%
RNN	77%	84%	84%	84%
7 Features				
CNN	75%	84%	79%	82%
LSTM	59%	69%	72%	71%
CNN-LSTM	**86%**	**99%**	**81%**	**89%**
CNN-GRU	60%	100%	42%	59%
RNN	67%	73%	83%	78%
9 Features				
CNN	68%	61%	68%	56%
LSTM	38%	37%	58%	45%
CNN-LSTM	64%	67%	93%	78%
CNN-GRU	69%	69%	100%	81%
RNN	68%	69%	97%	81%

The results of the third experiment, as shown in Table 13, indicate that the CNN model performed relatively well with 16 features, achieving 62% accuracy, precision, recall, and F1-score. However, as the number of features increased to 30, the performance of most models declined compared to their results with 16 features.

Table 13. The results of the third experiment (using PCA with 16 and 30 features)

Model	Accuracy	Precision	Recall	F1
16 Features				
CNN	**62%**	**62%**	**62%**	**62%**
LSTM	60%	63%	60%	61%
CNN-LSTM	62%	53%	62%	56%
CNN-GRU	48%	55%	48%	50%
RNN	52%	56%	52%	53%
30 Features				
CNN	57%	53%	57%	48%
LSTM	54%	54%	54%	54%
CNN-LSTM	58%	53%	58%	44%
CNN-GRU	55%	53%	55%	53%
RNN	51%	52%	51%	51%

Finally, in the fourth experiment, as shown in Table 14, the CNN model achieved the highest performance, with 78% accuracy, precision, recall, and F1-score. Figure 14 illustrates the confusion matrix for the best result of each experiment.

Table 14. The results of the fourth experiment (using ICA)

Model	Accuracy	Precision	Recall	F1-score
CNN	**78%**	**78%**	**78%**	**78%**
LSTM	56%	56%	56%	56%
CNN-LSTM	67%	44%	67%	53%
CNN-GRU	67%	44%	67%	53%
RNN	33%	43%	33%	30%

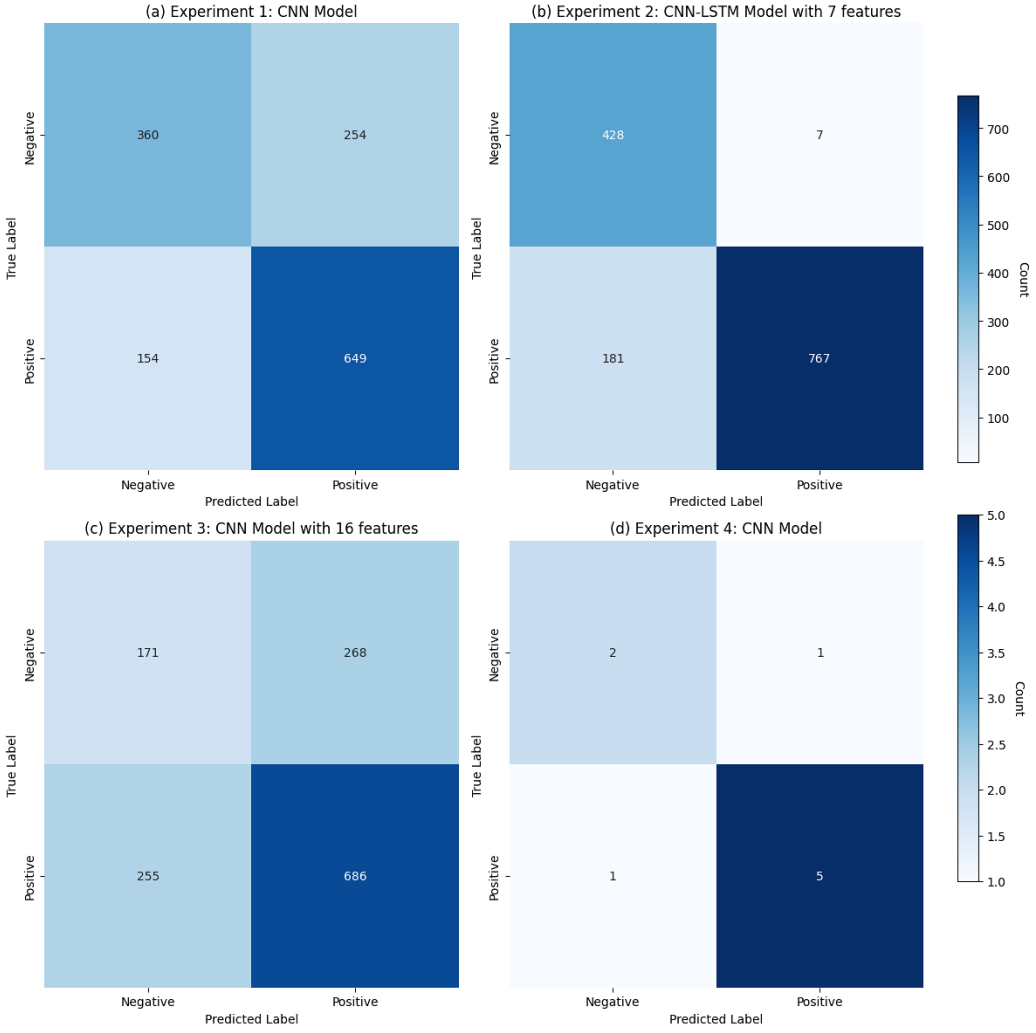


Figure 14. The confusion matrix for the best result of each experiment

In this work, four experimental setups were examined. The first setup employed the full feature set, and the obtained results are summarized in Table 11. The findings show that CNN and CNN-GRU achieved the best performance, reaching an accuracy of 70%, with both models also achieving 70% precision, recall, and F1-score. CNN-LSTM ranked second with 68% accuracy, recall, and F1-score, and 69% precision. LSTM achieved similar results, while RNN showed the lowest performance. Comparing the obtained results with the related works reported previously in Table 2, the accuracy achieved in this study is lower than that reported in previous works due to differences in dataset size, signal characteristics, applied techniques, experimental setups, and the use of highly curated or limited-size datasets. However, across all evaluation metrics, the results of the proposed work demonstrate balanced classification performance and better generalization capability.

Table 12 presents the results of the second experiment, in which DL models using different feature sets (5, 7, and 9) were investigated. The results show that model performance varies significantly depending on the number of features used. The best result was achieved by the CNN-LSTM model with seven features, yielding 86% accuracy, 99% precision, 81% recall, and an 89% F1-score.

Using nine features, CNN-GRU achieved the highest recall at 100%, followed by the RNN with 97%. The results obtained in this study demonstrate the importance of feature selection.

In the third experiment, DL models using PCA with 16 and 30 features were evaluated, as shown in Table 13. The best accuracy was achieved with 16 input features, where CNN maintained balanced performance across all evaluation metrics: accuracy, precision, recall, and F1-score. CNN-LSTM also achieved 62% accuracy, with lower performance across the remaining metrics. In contrast, CNN-GRU and RNN achieved lower accuracy.

With the set of 30 features, the overall performance decreased across all metrics compared to the 16-feature set, indicating that having a large set of features may negatively impact performance by introducing irrelevant information. Comparing the outcomes of the second and third experiments highlights the positive impact of careful feature engineering in similar models. Using a moderate feature set size produces the best classification results.

The results shown in Table 14 present the outcome of the fourth experiment, in which ICA was applied. The CNN model reported balanced classification performance across all metrics at 78%. CNN-LSTM and CNN-GRU both ranked second, with approximately an 11% lower accuracy. RNN obtained the lowest performance, achieving 33% accuracy, 43% precision, 33% recall, and a 30% F1-score.

Figure 14 illustrates the confusion matrix of the best performing model for each of the conducted experiments. The matrix shows the numbers of true positives, true negatives, false positives, and false negatives. The obtained results confirm that the best performing models correctly classified the majority of stroke cases (true positives), while the misclassification of normal and stroke cases (false positives and false negatives) was minimal.

Additionally, the McNemar test was employed as a statistical significance test to compare the performance differences among the five DL models: CNN, LSTM, CNN-LSTM, CNN-GRU, and RNN. P-values below the 0.05 threshold were used as indicators of statistical significance. This test was used to determine whether the observed differences in model performance were statistically significant or simply the result of random variation within the dataset. For the second and third experiments, the number of features that yielded the highest accuracy was selected, which were seven features for the second experiment and 16 features for the third experiment

The results presented in Table 15 indicate that the first three experiments, each including five DL models (CNN, LSTM, CNN-LSTM, CNN-GRU, and RNN), yielded statistically significant results, except for the comparison between CNN and CNN-LSTM in the first experiment, for which the p-value was 0.093903701. However, the fourth experiment did not show statistically significant results, likely due to the small test set, which included only nine records. As mentioned earlier, segmentation preprocessing was not applied in this experiment.

Table 15. Pairwise McNemar test results for model comparisons (✓indicates a statistically significant difference (p-value < 0.05), while × indicates no significant difference)

Model	Exp.	CNN	LSTM	CNN-LSTM	CNN-GRU	RNN
CNN	1st	-	✓	×	✓	✓
	2nd	-	✓	✓	✓	✓
	3rd	-	✓	✓	✓	✓
	4th	-				
LSTM	1st	✓	-	✓	✓	✓
	2nd	✓	-	✓	✓	✓
	3rd	✓	-	✓	✓	✓
	4th	×	-	×	×	×
CNN-LSTM	1st	×	✓	-	✓	✓
	2nd	✓	✓	-	✓	✓
	3rd	✓	✓	-	✓	✓
	4th	×	×	-	×	×
CNN-GRU	1st	✓	✓	✓	-	✓
	2nd	✓	✓	✓	-	✓
	3rd	✓	✓	✓	-	✓
	4th	×	×	×	-	×
RNN	1st	✓	✓	✓	✓	-
	2nd	✓	✓	✓	✓	-
	3rd	✓	✓	✓	✓	-
	4th	×	×	×	×	-

5- Discussion

This research initially involves developing several DL models, including CNN, RNN, LSTM, and GRU, to predict brain strokes using EEG signals. In addition, various feature engineering techniques, including DT, ICA, and PCA, were applied to investigate their impact on model performance.

In the first experiment, the CNN and CNN–LSTM models outperformed the other models, achieving 70% across all evaluation metrics. In the second experiment, which applied the DT model for feature selection, the CNN–LSTM model achieved the best performance among all models using seven selected features, reaching 86% accuracy, 99% precision, 81% recall, and an F1-score of 89%. Moreover, when the PCA feature reduction technique was applied, the CNN model outperformed the other models using 16 features, achieving 62% across all performance metrics. In the final experiment, which applied the ICA feature reduction technique, the CNN model again achieved the best performance, reaching 78% across all evaluation metrics.

Among all 35 conducted experiments involving different DL models, feature reduction and engineering techniques, and varying numbers of included features, the CNN–LSTM model with seven selected features using the DT model produced the best results. As shown in Figure 15, the training accuracy initially remains relatively stable before increasing significantly after approximately epoch 11 and gradually approaching 80%. The validation accuracy shows a noticeable increase around epoch 15 and then stabilizes at approximately 71–72%. This trend indicates that the model progressively learns meaningful patterns from the training data while maintaining reasonable generalization performance on the validation set. In addition, Figure 16 shows that the training loss decreases steadily over the epochs, demonstrating that the model is effectively optimizing its parameters. The validation loss also decreases during the early training stages and then stabilizes with minor fluctuations after approximately epoch 15. This behavior suggests that the model converges well and maintains stable generalization performance without severe overfitting.

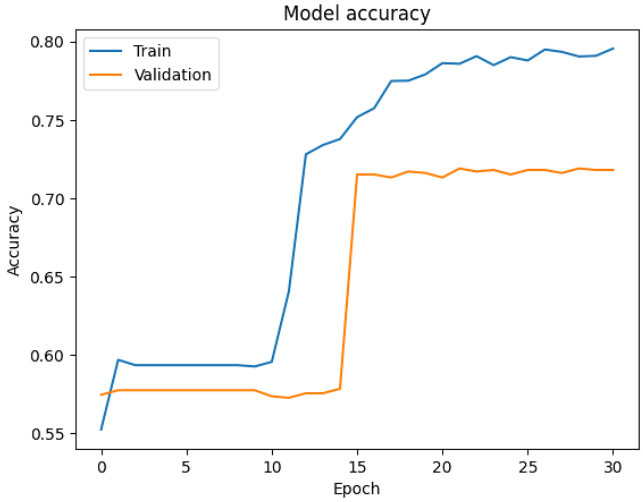


Figure 15. The training and validation accuracy over epochs for the CNN-LSTM model with seven selected features using the DT model

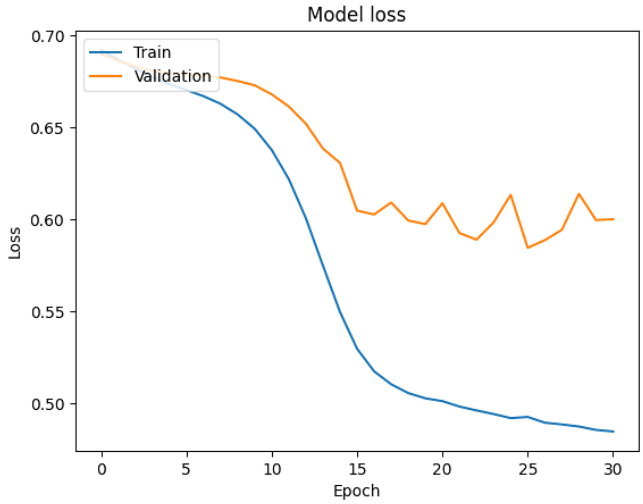


Figure 16. The training and validation loss over epochs for the CNN-LSTM model with seven selected features using the DT model

The DT model performs better than PCA and ICA for feature selection, as DT is considered one of the most effective supervised feature selection techniques. This is because DT selects features based on their relevance to the target variable. DT is also suitable for small datasets, as in this research; however, its performance may decrease when applied to very large datasets. Unlike PCA and ICA, which reduce dimensionality by transforming the original features into fewer components, the DT model selects a subset of the original features. This finding suggests that early stroke detection could potentially be achieved using a wearable EEG device with fewer electrodes (channels), making it more affordable and easier to install without requiring specialized expertise. However, due to the dataset preprocessing described in Section 3.3, it is difficult to explicitly identify the most important selected features, as they depend on variations across channels and time points. This limitation can be considered a drawback of the DT-based feature selection technique.

It is important to note that the final experiment, which employed ICA for feature engineering, was the only one that did not incorporate segmentation as a preprocessing step. Our results indicate that the performance in this experiment was lower than in experiments where segmentation was applied. ICA operates on the entire dataset globally to extract independent components [42] and is commonly used in EEG analysis for tasks such as removing artifacts like eye blinks. Applying segmentation before ICA could restrict its ability to effectively separate signals; for this reason, segmentation was omitted in the fourth experiment

As mentioned earlier, segmenting the dataset increased the number of records from 45 to 7,150 samples. Therefore, the decrease in the results of the fourth experiment, which used ICA for feature engineering, could be due to the smaller dataset size. In addition, ICA is an unsupervised learning method, meaning it does not use class labels to determine which components are most useful for classification. As a result, it may retain components that are statistically independent but not necessarily relevant to the classification task. In contrast, DT is a supervised learning algorithm that selects features based on the actual class labels, giving it a strong advantage in supervised classification tasks.

While using GridSearchCV followed by manual tuning to refine hyperparameter configurations allowed us to identify reasonably effective model settings, it may not guarantee globally optimal hyperparameter configurations. Therefore, some models' performance could potentially be improved by exploring a wider hyperparameter search space. However, to mitigate potential bias, the same hyperparameter search strategy was consistently applied across all five models to ensure a fair comparison, even if the resulting hyperparameters were not globally optimal.

The primary goal of this research is to develop precise and efficient DL models that achieve high accuracy in early stroke detection using EEG signal data. Based on the experimental results, the CNN model outperformed the other models across different feature engineering methods. Strokes often cause sudden localized changes in brain signals due to the lack of blood flow to specific brain regions. CNN models are well suited for detecting such localized patterns because they can learn filters that focus on small regions of the input signal, enabling them to identify patterns or anomalies within specific parts of the signal.

On the other hand, LSTM and RNN models are designed to capture temporal dependencies in sequential data. However, combining CNN and LSTM allows the proposed model to capture both local patterns and temporal relationships using well-selected features, which significantly enhances detection performance. In addition, the proposed approach improves generalization. As shown in Figures 17 and 18, the CNN model alone (using seven features selected via the DT method) exhibits overfitting. Training accuracy continues to increase across epochs, while validation accuracy plateaus around 0.71, indicating that the model learns the training data, including noise, but struggles to generalize to unseen data. Incorporating an LSTM layer after the CNN helps reduce this overfitting, improving the model's generalization

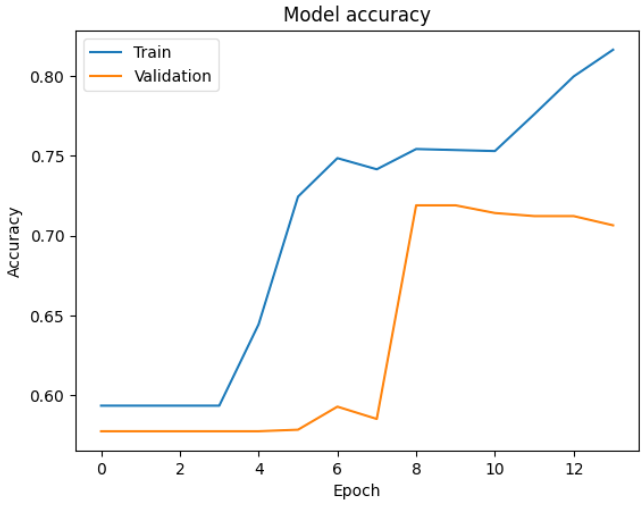


Figure 17. The training and validation accuracy over epochs for the CNN model with seven selected features using the DT model

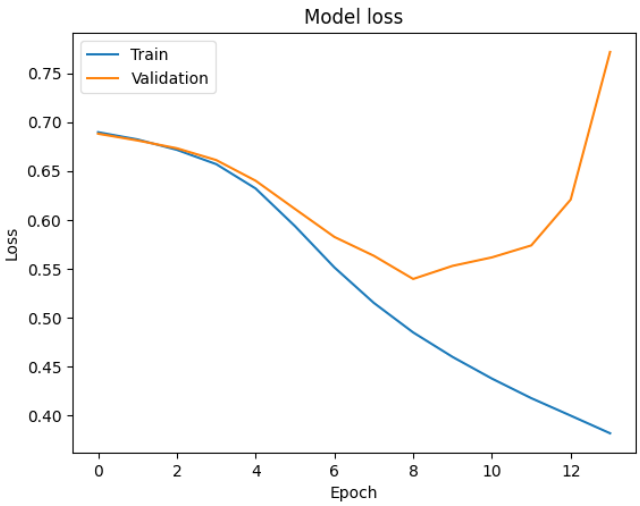


Figure 18. The training and validation loss over epochs for the CNN model with seven selected features using the DT model

5-1- Ablation Study

An ablation study is conducted in this section to evaluate the contribution of each preprocessing and feature engineering step. By selectively removing or modifying components, such as feature selection techniques (e.g., ICA, PCA) or parts of the model, we evaluated their impact on model performance for the best model, CNN–LSTM with DT-based feature selection. Table 16 shows the ablation study results.

Table 16. The ablation study results

Feature Selection Technique	Model	No. of Feature	Accuracy	Precision	Recall	F1-score
DT-based	CNN-LSTM	7	**86%**	**99%**	**81%**	**89%**
DT-based	CNN	7	75%	84%	79%	82%
DT-based	LSTM	7	59%	69%	72%	71%
DT-based	CNN-LSTM	5	67%	73%	83%	77%
DT-based	CNN-LSTM	9	64%	67%	93%	78%
NA	CNN-LSTM	(33,1250)	68%	69%	68%	68%
ICA	CNN-LSTM	(32,32)	67%	44%	67%	53%
PCA	CNN-LSTM	(16,1250)	62%	53%	62%	56%

In this research, there were some limitations due to the small dataset and limited computational resources. Therefore, the results of the DL models are not highly accurate. This is because DL models typically require a large dataset to achieve high performance. Additionally, potential biases in data collection, such as the lack of demographic diversity among stroke patients, should be considered. Addressing these limitations would provide valuable insights for improving model generalizability and fairness.

5-2- Real-World Application

The results indicate promising performance in classifying stroke and non-stroke signals, highlighting the potential of integrating the proposed model into portable diagnostic systems for early stroke prediction. Compared to traditional approaches such as CT scans and MRIs, portable diagnostic systems offer advantages in cost-effectiveness and rapid diagnosis. For instance, as noted by Wilkinson et al. [43], the device requires only five minutes to set up, making it feasible for use in ambulances. However, a key limitation of this approach is the need for a large, diverse training dataset to address signal variability across individuals and ensure accurate classification. Furthermore, the system should be designed to be user-friendly to minimize reliance on specialized operators, thereby addressing challenges related to setup and interpretation by non-expert users.

The integration of wearable EEG devices into ambulance settings presents a compelling real-world application with significant potential to enhance emergency care by enabling paramedics to make timely, data-driven decisions. However, several practical challenges should be carefully considered for successful real-world implementation. First, real-time data processing remains a critical hindrance, as EEG signal analysis often requires high computational resources that may be difficult to support in compact, wearable devices. Additionally, EEG signals are highly susceptible to noise and artifacts, especially in dynamic environments like ambulances, where patient movement, vibrations, and environmental interference can degrade signal quality. Moreover, wearable devices encounter limitations in battery life, processing

power, and data storage capacity, all of which can hinder continuous monitoring and real-time decision-making. Addressing these technical and operational constraints is essential to ensure the proposed solution's feasibility, reliability, and clinical utility.

Additionally, ethical and privacy concerns are associated with the collection and utilization of EEG signals for stroke detection, emphasizing the importance of responsible data handling and implementation. EEG signals are capable of revealing not only neurological conditions but also cognitive and emotional states, raising significant privacy considerations [44]. The possibility of misuse or unauthorized access to EEG signals necessitates stringent data protection measures, informed consent protocols, and transparent data governance frameworks. Furthermore, ethical considerations must guide the deployment of EEG-based diagnostic tools to ensure they do not inadvertently contribute to bias, inequality in access, or misdiagnosis. Addressing these concerns is critical to fostering trust among patients and healthcare providers and ensuring the safe integration of such technologies in both clinical and real-world environments.

6- Conclusion

With the increase in stroke diagnoses worldwide, it is essential to develop tools to detect strokes accurately and safely. Although several studies have been conducted to detect strokes using bio-signals, specifically EEG data, there were several limitations to be addressed. This study aimed to fill the gaps in prior research by utilizing EEG signals to detect brain stroke with multiple DL model architectures and feature selection techniques, providing a foundation for future clinical experiments involving wearable mobile diagnostic systems. Five DL models were used: RNN, CNN-GRU, LSTM, CNN-LSTM, and CNN. The Stroke EIT dataset, consisting of 45 EEG records from 33 healthy and stroke-diagnosed individuals, was used to develop these models. Two optimization techniques, manual hyperparameter tuning and GridSearchCV, were applied to find the optimal parameters to improve the performance and generalizability of the models. Furthermore, three feature engineering techniques, DT, ICA, and PCA, were applied to extract meaningful information from the EEG records and reduce data dimensionality for analysis. Several experiments were performed, including setups using all features, DT-selected features, PCA, and ICA. The results indicated that the CNN-LSTM model with seven selected features using the DT method performed best, achieving 86% accuracy, 99% precision, 81% recall, and an F1-score of 89%. This remarkable result demonstrates the effectiveness of combining a CNN model to localize signal patterns with LSTM support for capturing temporal relationships. These promising results encourage further exploration of integrating the model into real-world applications and evaluating its performance in a clinical setup.

6-1- Future Work

In the future, it would be beneficial to expand the size of the datasets used in this study to include more records from larger cohorts of individuals, both healthy and stroke-diagnosed, to further increase the robustness and generalizability of the models. Another area of future work is to explore the use of ECG (electrocardiogram) instead of EEG to examine the potential impact it can have on stroke detection. Even though the findings indicate promising results, showcasing the potential of the fundamental DL models for sequential data in accurately identifying strokes, future work could explore more advanced learning strategies, such as Transformer-based models, transfer learning, multimodal data integration, or domain adaptation techniques, to improve model generalizability and robustness.

7- Declarations

7-1- Author Contributions

Conceptualization, M.I.A. and F.H.A.; methodology, M.I.A, R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; software, R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; validation, D.A.A., S.S.A., and S.O.O.; formal analysis, F.H.A., R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; investigation, M.I.A, R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; resources, M.I.A, R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; data curation, R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; writing—original draft preparation, M.I.A, F.H.A., R.A.H.A., H.A.A., R.A.A., M.S.A., and S.F.A.; writing—review and editing, M.I.A. and F.H.A.; visualization, A.A.A.; supervision, M.I.A. and F.H.A.; project administration, M.I.A. All authors have read and agreed to the published version of the manuscript.

7-2- Data Availability Statement

The data presented in this study are available on request from the corresponding author.

7-3- Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

7-4- Institutional Review Board Statement

Not Applicable.

7-5- Informed Consent Statement

Not Applicable.

7-6- Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancies have been completely observed by the authors.

8- References

[1] Gupta, S., & Raheja, S. (2022). Stroke Prediction using Machine Learning Methods. 2022 12th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 553–558. doi:10.1109/Confluence52989.2022.9734197.

[2] Shoily, T. I., Islam, T., Jannat, S., Tanna, S. A., Alif, T. M., & Ema, R. R. (2019). Detection of Stroke Disease using Machine Learning Algorithms. 2019 10th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2019, 1–6. doi:10.1109/ICCCNT45670.2019.8944689.

[3] Yu, J., Park, S., Kwon, S. H., Ho, C. M. B., Pyo, C. S., & Lee, H. (2020). AI-based stroke disease prediction system using real-time electromyography signals. Applied Sciences (Switzerland), 10(19), 6791. doi:10.3390/app10196791.

[4] Chiaramonte, R., Pavone, P., & Vecchio, M. (2020). Speech rehabilitation in dysarthria after stroke: A systematic review of the studies. European Journal of Physical and Rehabilitation Medicine, 56(5), 547–562. doi:10.23736/S1973-9087.20.06185-7.

[5] Aldossary, M. I., Alghamedy, F. H., Alabbad, D. A., Alnuaim, R. A., Altaweel, M. S., Alshami, R. A. H., … Olatunji, S. O. (2025). ML and DL Models for Stroke Prediction from Bio-Signals: A Systematic Review and Bibliometric Analysis. HighTech and Innovation Journal, 6(3), 1062–1078. doi:10.28991/HIJ-2025-06-03-019.

[6] Thayumanavan, M., & Ramasamy, A. (2023). Deep Learning Based Stroke Disease Prediction using Multi Modal Biosignals. 2023 International Conference on Next Generation Electronics, NEleX 2023, 1–6. doi:10.1109/NEleX59773.2023.10421131.

[7] Yoon, D., Jang, J. H., Choi, B. J., Kim, T. Y., & Han, C. H. (2020). Discovering hidden information in biosignals from patients using artificial intelligence. Korean Journal of Anesthesiology, 73(4), 275–284. doi:10.4097/kja.19475.

[8] Shreve, L., Kaur, A., Vo, C., Wu, J., Cassidy, J. M., Nguyen, A., Zhou, R. J., Tran, T. B., Yang, D. Z., Medizade, A. I., Chakravarthy, B., Hoonpongsimanont, W., Barton, E., Yu, W., Srinivasan, R., & Cramer, S. C. (2019). Electroencephalography Measures are Useful for Identifying Large Acute Ischemic Stroke in the Emergency Department. Journal of Stroke and Cerebrovascular Diseases, 28(8), 2280–2286. doi:10.1016/j.jstrokecerebrovasdis.2019.05.019.

[9] Mak, J., Kocanaogullari, D., Huang, X., Kersey, J., Shih, M., Grattan, E. S., Skidmore, E. R., Wittenberg, G. F., Ostadabbas, S., & Akcakaya, M. (2022). Detection of Stroke-Induced Visual Neglect and Target Response Prediction Using Augmented Reality and Electroencephalography. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 30, 1840–1850. doi:10.1109/TNSRE.2022.3188184.

[10] Hussain, I., & Park, S. J. (2020). HealthSOS: Real-Time Health Monitoring System for Stroke Prognostics. IEEE Access, 8, 213574–213586. doi:10.1109/ACCESS.2020.3040437.

[11] Zhang, H., Zhou, Q. Q., Chen, H., Hu, X. Q., Li, W. G., Bai, Y., Han, J. X., Wang, Y., Liang, Z. H., Chen, D., Cong, F. Y., Yan, J. Q., & Li, X. L. (2023). The applied principles of EEG analysis methods in neuroscience and clinical neurology. Military Medical Research, 10(1). doi:10.1186/s40779-023-00502-7.

[12] Chicco, D., Karaiskou, A. I., & De Vos, M. (2024). Ten quick tips for electrocardiogram (ECG) signal processing. PeerJ Computer Science, 10, 1–22. doi:10.7717/PEERJ-CS.2295.

[13] Mehendale, N., & Gohel, V. (2020). Review on Electromyography Signal Acquisition, Processing and Its Applications. SSRN Electronic Journal. doi:10.2139/ssrn.3598928.

[14] Park, J., Seok, H. S., Kim, S. S., & Shin, H. (2022). Photoplethysmogram Analysis and Applications: An Integrative Review. Frontiers in Physiology, 12. doi:10.3389/fphys.2021.808451.

[15] Choi, Y. A., Park, S. J., Jun, J. A., Pyo, C. S., Cho, K. H., Lee, H. S., & Yu, J. H. (2021). Deep learning-based stroke disease prediction system using real-time bio signals. Sensors, 21(13), 4269. doi:10.3390/s21134269.

[16] Kalahasty, R., & Motati, L. S. (2022). An EEG-Based Diagnostic Framework for Strokes Using Spectral Analysis and Deep Learning. 2022 IEEE MIT Undergraduate Research Technology Conference (URTC), 1–5. doi:10.1109/URTC56832.2022.10002236.

[17] Kumar, S., & Sengupta, A. (2022). EEG Classification For Stroke Detection Using Deep Learning Networks. 2022 2nd International Conference on Emerging Frontiers in Electrical and Electronic Technologies, ICEFEET 2022. doi:10.1109/ICEFEET51821.2022.9847883.

[18] Anand Kumar, M., Abirami, N., Guru Prasad, M. S., & Mohankumar, M. (2022). Stroke Disease Prediction based on ECG Signals using Deep Learning Techniques. 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES), 453–458. doi:10.1109/CISES54857.2022.9844403.

[19] Kunwar, P., & Choudhary, P. (2023). A stacked ensemble model for automatic stroke prediction using only raw electrocardiogram. Intelligent Systems with Applications, 17, 200165. doi:10.1016/j.iswa.2022.200165.

[20] Elbagoury, B. M., Vladareanu, L., Vlădăreanu, V., Salem, A. B., Travediu, A. M., & Roushdy, M. I. (2023). A Hybrid Stacked CNN and Residual Feedback GMDH-LSTM Deep Learning Model for Stroke Prediction Applied on Mobile AI Smart Hospital Platform. Sensors, 23(7), 3500. doi:10.3390/s23073500.

[21] Goren, N., Avery, J., Dowrick, T., Mackle, E., Witkowska-Wrobel, A., Werring, D., & Holder, D. (2018). Multi-frequency electrical impedance tomography and neuroimaging data in stroke patients. Scientific Data, 5(1), 180112. doi:10.1038/sdata.2018.112.

[22] Theng, D., & Bhoyar, K. K. (2024). Feature selection techniques for machine learning: a survey of more than two decades of research. Knowledge and Information Systems, 66(3), 1575–1637. doi:10.1007/s10115-023-02010-5.

[23] Jollife, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374(2065), 20150202. doi:10.1098/rsta.2015.0202.

[24] Subasi, A., & Gursoy, M. I. (2010). EEG signal classification using PCA, ICA, LDA and support vector machines. Expert Systems with Applications, 37(12), 8659–8666. doi:10.1016/j.eswa.2010.06.065.

[25] Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., ... & Hämäläinen, M. (2013). MEG and EEG data analysis with MNE-Python. Frontiers in Neuroinformatics, 7, 267. doi:10.3389/fnins.2013.00267.

[26] Hao-Cun, W. U. (2007). Independent Component Analysis and Its Applications in Finance Independent Component Analysis. Thesis, University of Hong Kong, Pokfulam, Hong Kong. doi:10.5353/th_b3955909.

[27] Mienye, I. D., & Jere, N. (2024). A Survey of Decision Trees: Concepts, Algorithms, and Applications. IEEE Access, 12, 86716–86727. doi:10.1109/ACCESS.2024.3416838.

[28] Zhao, Y., Huang, Y., Wang, Z., & Liu, X. (2024). A new feature selection method based on importance measures for crude oil return forecasting. Neurocomputing, 581, 127470. doi:10.1016/j.neucom.2024.127470.

[29] Gautam, A., & Raman, B. (2021). Towards effective classification of brain hemorrhagic and ischemic stroke using CNN. Biomedical Signal Processing and Control, 63, 102178. doi:10.1016/j.bspc.2020.102178.

[30] Ruan, D., Wang, J., Yan, J., & Gühmann, C. (2023). CNN parameter design based on fault signal analysis and its application in bearing fault diagnosis. Advanced Engineering Informatics, 55, 101877. doi:10.1016/j.aei.2023.101877.

[31] Waqas, M., & Humphries, U. W. (2024). A critical review of RNN and LSTM variants in hydrological time series predictions. MethodsX, 13, 102946. doi:10.1016/j.mex.2024.102946.

[32] Kuila, S., Dhanda, N., & Joardar, S. (2022). ECG signal classification and arrhythmia detection using ELM-RNN. Multimedia Tools and Applications, 81(18), 25233–25249. doi:10.1007/s11042-022-11957-6.

[33] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[34] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1724–1734. doi:10.3115/v1/d14-1179.

[35] Xu, S., Li, J., Liu, K., & Wu, L. (2019). A Parallel GRU Recurrent Network Model and its Application to Multi-Channel Time-Varying Signal Classification. IEEE Access, 7, 118739–118748. doi:10.1109/ACCESS.2019.2936516.

[36] Wang, J., Yu, L.-C., Lai, K. R., & Zhang, X. (2016). Dimensional Sentiment Analysis Using a Regional CNN-LSTM Model. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 225–230. doi:10.18653/v1/p16-2037.

[37] Dua, N., Singh, S. N., & Semwal, V. B. (2021). Multi-input CNN-GRU based human activity recognition using wearable sensors. Computing, 103(7), 1461–1478. doi:10.1007/s00607-021-00928-8.

[38] Cao, B., Shang, H., Fan, M., & Sun, F. (2024). CNN-GRU based method for peak location of reflected Terahertz signals from thermal barrier coatings. Nondestructive Testing and Evaluation, 39(8), 2132–2149. doi:10.1080/10589759.2023.2288880.

[39] Khan, H. A., Ul Ain, R., Kamboh, A. M., Butt, H. T., Shafait, S., Alamgir, W., Stricker, D., & Shafait, F. (2022). The NMT Scalp EEG Dataset: An Open-Source Annotated Dataset of Healthy and Pathological EEG Recordings for Predictive Modeling. Frontiers in Neuroscience, 15. doi:10.3389/fnins.2021.755817.

[40] Roy, S., Kiral-Kornek, I., & Harrer, S. (2019). ChronoNet: A Deep Recurrent Neural Network for Abnormal EEG Identification. In: Riaño, D., Wilk, S., ten Teije, A. (eds) Artificial Intelligence in Medicine. AIME 2019. Lecture Notes in Computer Science. Springer, Cham, Switzerland. doi:10.1007/978-3-030-21642-9_8.

[41] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(2), 281-305.

[42] Shlens, J. (2014). A tutorial on independent component analysis. arXiv Preprint, arXiv:1404.2986. doi:10.48550/arXiv.1404.2986.

[43] Wilkinson, C. M., Burrell, J. I., Kuziek, J. W. P., Thirunavukkarasu, S., Buck, B. H., & Mathewson, K. E. (2020). Predicting stroke severity with a 3-min recording from the Muse portable EEG system for rapid diagnosis of stroke. Scientific Reports, 10(1),18465. doi:10.1038/s41598-020-75379-w.

[44] Jafari, M., Shoeibi, A., Khodatars, M., Bagherzadeh, S., Shalbaf, A., García, D. L., Gorriz, J. M., & Acharya, U. R. (2023). Emotion recognition in EEG signals using deep learning methods: A review. Computers in Biology and Medicine, 165, 107450. doi:10.1016/j.compbiomed.2023.107450.