DEEP STROKE DETECT: OPTIMIZING STROKE RISK PREDICTION USING SMOTEENN AND TRANSFER LEARNING

A Project Report submitted in the partial fulfillment

of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

Shaik Nazeema (22471A05D3)
Parella Akhila (22471A05B4)
Diddi Sandhya (22471A0586)

Under the esteemed guidance of

T. Sai Sarnya, MTech.,

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET
(AUTONOMOUS)
Accredited by NAAC with A+ Grade and an ISO 9001:2015 Certified
Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada
KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2025-2026

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning" is a bonafide work done by the team Shaik Nazeema (22471A05D3), Parella Akhila (22471A05B4), Diddi Sandhya (22471A0586) BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2025-2026.

PROJECT GUIDE
T. Sai Sarnya, M.Tech.,
Assistant Professor

PROJECT CO-ORDINATOR
Dr. Sireesha Moturi, M.Tech., Ph.D.,
Associate Professor

HEAD OF THE DEPARTMENT
Dr. S. N. Tirumala Rao, M.Tech., Ph.D.
Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "**DEEP STROKE DETECT: OPTIMIZING STROKE RISK PREDICTION USING SMOTEENN AND TRANSFER LEARNING**" is composed by us that the work contain here is our own except where explicitly stated otherwise in the text and that this work has not been submitted for any other degree or professional qualification except as specified.

By

Shaik Nazeema (22471A05D3)
Parella Akhila (22471A05B4)
Diddi Sandhya (22471A0586)

ACKNOWLEDGEMENT

We wish to express our thanks to various personalities who are responsible for the completion of my project. We are extremely thankful to our beloved chairman**, Sri M. V. Koteswara Rao, B.Sc.,** who took keen interest in us in every effort throughout this course. We owe our sincere gratitude to our beloved principal**, Dr. S. Venkateswarlu, Ph.D.**, for showing his kind attention and valuable guidance throughout the course.

We express our deep-felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.** HOD of the CSE department, and also to our guide**, T. Sai Sarnya, M.Tech.**, whose valuable guidance and unstinting encouragement enabled us to accomplish my project successfully in time.

We extend our sincere thanks to **Dr. Sireesha Moturi**, **B.Tech., M.Tech., Ph.D.** Assistant Professor & Project Coordinator of the project, for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all the other teaching and non-teaching staff in the department for their cooperation and encouragement during my B.Tech. degree.

We have no words to acknowledge the warm affection, constant inspiration, and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who were involved in giving valuable suggestions and clarifying our doubts, which really helped us in successfully completing our project.

By

Shaik Nazeema (22471A05D3)
Parella Akhila (22471A05B4)
Diddi Sandhya (22471A0586)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community.

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a center of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to the academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science,computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/Development of Solutions: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct Investigations of Complex Problems: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: Individual and Collaborative Team work: Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports

and design documentation, make effective presentations considering cultural, language, and learning differences.

PO10: Project Management and Finance: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and
in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.
CO421.2: Identify and classify the requirements.
CO421.3: Review the Related Literature
CO421.4: Design and Modularize the project
CO421.5: Construct, Integrate, Test and Implement the Project.
CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO 7	PO8	PO9	PO1 0	PO1 1	PSO1	PSO2	PSO3
C421.1		✓										✓		
C421.2	✓		✓		✓							✓		
C421.3				✓		✓	✓	✓				✓		
C421.4			✓			✓	✓	✓				✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓	✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
C421.1	2	3										2		
C421.2			2		3							2		
C421.3				2		2	3	3				2		
C421.4			2			1	1	2				3	2	
C421.5					3	3	3	2	3	2	2	3	2	1
C421.6									3	2	1	2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop model for detection and classification of Brain Stroke using SMOTEENN model	PO1, PO3,PO8
CC421.1, C2204.3, C22L3.2	Each and every requirement i critically analyzed, the process mode is identified	PO2, PO3
CC421.2, C2204.2, C22L3.3	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documentation is done by all our three members in the form of a group	PO10
CC421.5, C2204.2, C22L3.3	Each and every phase of the work in group is presented periodically	PO8,PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementation is done and the project will be handled by the social media users and in future updates in our project can be done based on detection for Brain Stroke	PO4, PO7, PO8
C32SC4.3	The physical design includes website to check Brain Stroke	PO5, PO6

ABSTRACT

Stroke remains a major global health concern, being one of the top causes of mortality and a leading contributor to long-term disability. This is the very reason why recognizing the signs of risk is critical to treatment—timing is everything. In this paper, we present a novel approach to assess the risk of stroke using deep learning. The system is applied to real-world healthcare datasets, which often exhibit a greater prevalence of non-stroke cases compared to stroke cases. To address the imbalance and enhance the data quality, we implemented several data preparation steps, SMOTEENN, and a cross-validated deep neural network model. We also applied transfer learning to mitigate the impact of scarce data on model performance. The model's accuracy was assessed using the common metrics of accuracy, precision, recall, F1score, and ROC-AUC. The outcomes were quite encouraging, achieving 95.24% accuracy on average and 95.52% F1-score, confirming that the model is extremely accurate. Apart from the outcomes, this study attempts to solve significant real-world challenges such as the sparse nature of stroke cases, data sparsity, and selection of relevant health indicators. The findings indicate that deep learning approaches are very effective, provided there is thorough data preparation along with intelligent pre-processing. Feature selection can be an impactful method for predicting stroke risk—potentially helping doctors take action earlier and save lives.

INDEX

LIST OF FIGURES

List of Tables

1. INTRODUCTION

Stroke represents one of the most severe and prevalent medical conditions affecting humanity today, posing an enormous global health and socioeconomic challenge. It ranks as the second leading cause of death and a major contributor to long-term disability, leaving millions of individuals worldwide with lasting neurological impairments [1], [2]. According to the World Health Organization (WHO), approximately 15 million people suffer a stroke each year, of which nearly 5 million die and another 5 million are left permanently disabled, placing a tremendous strain on families, healthcare systems, and national economies [1]. The consequences of stroke extend beyond individual suffering, leading to extensive medical costs, long-term rehabilitation needs, and significant productivity losses that collectively burden society [2], [3].

A stroke occurs when the normal flow of blood to the brain is interrupted, either due to a blockage (ischemic stroke) or a rupture of a blood vessel (hemorrhagic stroke), causing oxygen deprivation and rapid death of brain cells [3]. The outcome of a stroke depends heavily on how quickly it is detected and treated. Unfortunately, in many cases, symptoms are subtle, unrecognized, or identified too late for effective intervention [4]. As a result, early detection and timely medical attention are crucial in preventing irreversible damage and improving recovery outcomes [4], [5].

Understanding stroke risk requires a multifaceted approach, as its development is influenced by numerous interconnected factors. Common risk factors include hypertension, diabetes, cardiovascular disease, hyperlipidemia, smoking, obesity, stress, and sedentary lifestyle [6], [7]. These factors often interact in complex, nonlinear ways, making it difficult for traditional statistical or rule-based medical models to accurately predict who is at risk [8]. Moreover, variations in genetics, lifestyle habits, and environmental conditions further complicate the assessment, emphasizing the need for more advanced analytical frameworks [9].

In recent years, the expansion of digital healthcare systems, electronic medical records, and wearable health-monitoring devices has generated vast amounts of patient data [10], [11]. These data sources present an invaluable opportunity to apply artificial intelligence (AI) and machine learning (ML) techniques to uncover hidden patterns and predictive markers of stroke [4], [6], [8]. Such data-driven approaches enable healthcare providers to move from reactive treatment to proactive prevention, identifying high-risk individuals before symptoms appear [11], [12].

Predictive modeling not only assists clinicians in early diagnosis but also empowers individuals to modify their behavior and reduce risk factors through lifestyle adjustments [13].

Despite these advancements, several challenges persist in building accurate stroke prediction systems. Medical datasets often suffer from class imbalance, where the number of stroke cases is significantly lower than non-stroke cases, which biases models toward majority classes and reduces their predictive reliability [13], [15]. Additionally, inconsistencies in medical data, missing values, and noise can degrade model performance [5], [15]. Furthermore, existing clinical scoring systems such as the Framingham Stroke Risk Profile, though widely used, often fail to capture the nonlinear and multifactorial relationships inherent in real-world stroke data [2], [3].

Addressing these limitations requires the integration of advanced data preprocessing methods, feature selection, and hybrid machine learning and deep learning models that can learn complex relationships effectively [4], [8], [14]. Modern frameworks incorporating data balancing techniques, transfer learning, and ensemble strategies have demonstrated promising results in improving the precision and generalizability of stroke prediction models [10], [13], [15]. By leveraging these technologies, healthcare systems can transition toward personalized and predictive care, minimizing human error and enhancing diagnostic accuracy [12], [16].

Ultimately, developing robust stroke prediction systems is not just a technical challenge but a public health imperative [1], [2]. Early and accurate identification of at-risk individuals can dramatically reduce stroke incidence, mortality, and long-term disability rates [11], [12]. Such systems can revolutionize preventive healthcare by supporting timely intervention, guiding resource allocation, and ultimately saving lives while reducing the global economic burden associated with stroke [1], [17].

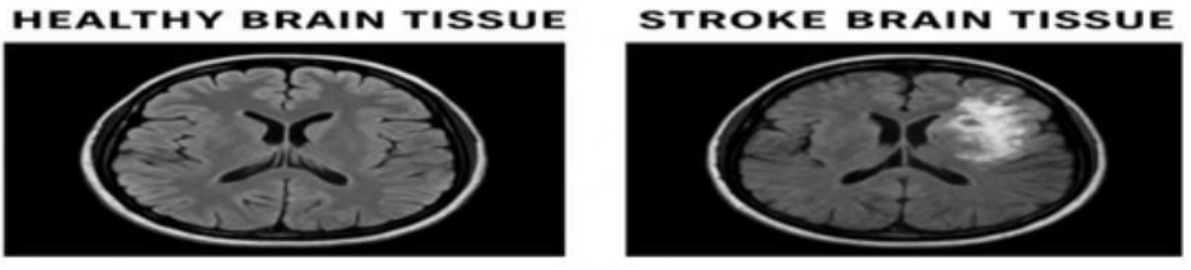

FIG1.1 CLASSIFICATION OF HEALTHY BRAIN TISSUE AND STROKE BRAIN TISSUE

The above Fig. 1.1 illustrates a comparative visualization between healthy brain tissue and stroke-affected brain tissue obtained from MRI scans.

On the left side, the healthy brain tissue displays a symmetrical and well-defined structure with clear differentiation between gray and white matter regions. The ventricles appear uniform, and there are no visible lesions or irregularities, representing a normal MRI scan.

On the right side, the stroke-affected brain tissue reveals distinct abnormalities. The bright, irregular region highlights the infarcted area caused by disrupted blood supply leading to neuronal cell death. The asymmetry between hemispheres and the distortion of surrounding structures confirm the presence of a stroke lesion.

This visual comparison forms the foundation for image-based classification in stroke prediction systems. Deep learning models, particularly Convolutional Neural Networks (CNNs), are trained on such brain images to automatically distinguish between healthy and stroke-affected tissues [8], [14]. By learning texture, intensity, and structural differences, the system can assist radiologists in rapid and accurate diagnosis.

1.1 MOTIVATION

Stroke remains one of the leading causes of death and long-term disability worldwide, imposing an enormous burden on individuals, families, and healthcare systems [1], [2]. According to the World Health Organization (WHO), millions of people suffer strokes each year, and nearly onethird of survivors experience permanent neurological impairments [1]. Despite continuous advances in medical imaging and clinical care, the early detection and prevention of stroke remain major challenges [3], [4]. The high mortality and morbidity associated with delayed diagnosis highlight the urgent need for advanced and accurate predictive models [2], [5].

Traditional clinical risk assessment methods, such as manual scoring systems or statistical models, often fall short due to their limited ability to capture the nonlinear relationships among various risk factors such as age, hypertension, diabetes, cholesterol level, smoking habits, and obesity [3], [6], [7]. Moreover, these models depend heavily on human expertise and are prone to subjective errors, which can result in inconsistent or delayed diagnoses [4], [6]. The increasing availability of healthcare data and the rise of artificial intelligence provide a unique opportunity to overcome these limitations through automated, data-driven approaches [8], [9].

Machine learning and deep learning techniques, particularly when enhanced with transfer learning, have shown remarkable potential in healthcare applications [8], [10], [11]. They can uncover complex patterns in medical data and identify subtle risk indicators that may be invisible to the human eye [12], [13]. However, in most medical datasets, the occurrence of stroke cases is relatively rare compared to non-stroke cases, leading to class imbalance problems [13], [15]. This imbalance reduces model performance and accuracy. The integration of data balancing techniques such as SMOTE-ENN (Synthetic Minority Over-sampling Technique combined with Edited Nearest Neighbors) helps in improving dataset quality and ensures fair learning across all classes [15].

Motivated by these challenges and opportunities, this project aims to develop a robust, intelligent, and data-driven stroke prediction system that leverages deep learning and transfer learning techniques [4], [8], [12]. By accurately identifying individuals at high risk before the onset of symptoms, such a system can enable early clinical intervention, improve patient outcomes, and significantly reduce the societal and economic impact of stroke [1], [2], [11]. The motivation behind this work lies not only in advancing predictive analytics in healthcare but also in contributing to a future where technology plays a proactive role in saving human lives [9], [12], [16].

1.2 PROBLEM STATEMENT

Stroke prediction and diagnosis continue to be significant challenges in healthcare due to the complex interaction of numerous physiological, behavioral, and lifestyle factors [1], [2]. Although advances in medical imaging and analytics have improved post-stroke care, identifying individuals at high risk before the onset of symptoms remains difficult [3], [4]. Traditional diagnostic and clinical scoring methods rely heavily on manual interpretation and threshold-based decisions, which often lead to inconsistent, delayed, or inaccurate predictions [3], [5]. These methods struggle to capture the nonlinear and multifactorial relationships that exist between various risk indicators such as blood pressure, glucose level, age, BMI, smoking habits, and cardiovascular conditions [6], [7].

Existing machine learning approaches have shown potential in predicting stroke risk, but they face several major limitations [4], [8]. The most critical issue is **data imbalance**—medical datasets generally contain a much larger proportion of non-stroke cases compared to stroke cases [13], [15]. This imbalance causes models to become biased toward the majority class, resulting in poor sensitivity in detecting actual stroke cases [5], [15]. Another major limitation is the lack of generalization capability, as many models trained on limited or domain-specific

data fail to perform effectively on diverse patient populations [8], [9]. Additionally, traditional algorithms often depend on manual feature extraction and lack the ability to learn high-level abstract patterns that may be crucial for accurate stroke prediction [6], [10], [12].

These challenges create an urgent need for a robust and intelligent predictive framework capable of handling imbalanced datasets, extracting complex patterns, and delivering reliable outcomes [4], [8], [10]. **Deep learning** and **transfer learning** offer powerful solutions to these issues by automatically learning meaningful features from data and adapting pre-trained models for medical prediction tasks [8], [11], [12]. Furthermore, incorporating a hybrid data balancing approach like **SMOTE-ENN (Synthetic Minority Over-sampling Technique with Edited Nearest Neighbors)** enhances model fairness by generating synthetic samples and eliminating noise, thereby improving classification performance [15].

This project aims to design and develop an automated stroke prediction system that integrates **deep learning**, **transfer learning**, and **SMOTE-ENN** to accurately predict stroke occurrence in heterogeneous and imbalanced datasets [4], [8], [13], [15]. The system is intended to improve early detection accuracy, minimize false predictions, and support healthcare professionals in making timely clinical decisions [9], [11], [16]. By leveraging advanced computational intelligence techniques, the proposed model seeks to contribute to early stroke prevention, reduce healthcare costs, and ultimately save lives [1], [2], [17].

1.3 OBJECTIVE

The main objective of this project is to develop a robust and intelligent stroke prediction system that accurately identifies individuals at risk of stroke using advanced deep learning and transfer learning techniques [4], [8], [11]. The goal is to overcome the limitations of traditional predictive models by enabling automated feature extraction, effective handling of imbalanced datasets, and improved generalization across diverse patient populations [5], [6], [10]. By leveraging clinical and demographic data, the system seeks to enhance early diagnosis and support medical professionals in making data-driven, evidence-based decisions [1], [2], [12].

This project focuses on designing a deep learning model capable of analyzing multiple healthrelated factors such as age, blood pressure, glucose level, body mass index (BMI), and lifestyle habits to determine the likelihood of stroke occurrence [3], [6]. Transfer learning is

employed to improve the model's learning efficiency and accuracy by utilizing pre-trained neural network architectures that already possess strong feature extraction capabilities [8], [11], [12]. To ensure fairness and reliability, the model incorporates the SMOTE-ENN (Synthetic Minority Over-sampling Technique combined with Edited Nearest Neighbors) technique to balance the dataset, addressing the common issue of class imbalance between stroke and non-stroke cases [13], [15].

Data preprocessing, normalization, and feature encoding are implemented to refine the input data and enhance model performance [4], [7], [9]. The developed model is then evaluated using standard performance metrics such as accuracy, precision, recall, F1-score, and GMean to assess its predictive strength [10], [13], [14]. Visual tools such as confusion matrices and performance plots are employed to analyze classification outcomes and validate the model's effectiveness [12], [14].

Ultimately, the objective of this work is to create a scalable and efficient stroke prediction framework that can be integrated into real-time healthcare systems [11], [16]. Such a system will facilitate early identification of high-risk patients, enable timely clinical interventions, and reduce stroke-related mortality by combining artificial intelligence with preventive healthcare strategies [1], [2], [17].

2. LITERATURE SURVEY

Stroke is one of the leading causes of death and long-term disability worldwide. With the advent of Artificial Intelligence (AI) and Deep Learning (DL), significant research has been carried out to enhance early stroke detection and prediction accuracy using medical and physiological data. This section summarizes relevant studies and methodologies from existing literature, emphasizing the role of data preprocessing, feature extraction, and hybrid deep learning models.

Tanaka et al. [11] utilized real-time sensor data from wearable devices to predict stroke risk using Long Short-Term Memory (LSTM) networks. Their preprocessing pipeline included signal denoising, temporal feature engineering, and Z-score normalization to refine physiological time-series data. The LSTM model achieved a precision of 92%, outperforming Gated Recurrent Unit (GRU) models. This study highlighted the significance of efficient temporal data preprocessing and its impact on predictive performance, demonstrating that wearable healthcare monitoring systems can provide continuous and non-invasive stroke detection solutions.

Park et al. [12] developed an IoT-based stroke monitoring framework that applied Bidirectional LSTM (Bi-LSTM) networks on a time-series physiological dataset. The proposed framework achieved 93% accuracy with improved recall compared to traditional models. This research emphasized the potential of bidirectional temporal modeling for capturing sequential health patterns, enabling more accurate predictions in stroke-prone individuals.

Ali et al. [13] investigated the widely cited Kaggle Stroke Dataset, employing feature selection techniques such as Mutual Information Gain to identify the most relevant attributes influencing stroke occurrence. The study revealed that age, blood sugar level, and hypertension were the most critical risk factors. Using the XGBoost classifier, they achieved an accuracy of 94%, demonstrating that well-engineered features and ensemble learning methods can significantly improve stroke prediction performance.

Nair et al. [9] proposed a Convolutional Neural Network (CNN)-based model for stroke prediction using data from wearable smartwatches. The CNN effectively captured spatial and temporal variations in physiological signals, showing promising results for portable, realtime stroke detection systems. However, the study encountered challenges related to data imbalance, which affected generalization. The authors suggested that applying Synthetic Minority Over-sampling Techniques (SMOTE) could alleviate these limitations, improving classification performance across unbalanced datasets.

Jagannadham et al. [14] demonstrated the application of Deep Learning in brain tumor detection using Convolutional Neural Networks (CNNs) to process MRI scans. Their method effectively identified tumor boundaries, significantly improving diagnostic accuracy and reducing interpretation time. Though focused on tumor detection, the study's methodology of deep feature extraction from medical imaging can be extended to stroke detection tasks for analyzing brain abnormalities.

Seva et al. [15] developed a smart healthcare system for liver disease prediction using the Random Forest classifier. To address the class imbalance common in medical datasets, they employed the SMOTE-ENN (Synthetic Minority Over-sampling and Edited Nearest Neighbors) hybrid balancing technique. This preprocessing approach enhanced model fairness and accuracy, underscoring the importance of balanced datasets in predictive healthcare systems.

Reddy et al. [16] emphasized the critical role of structured data splitting and data augmentation strategies in improving model performance for real-time medical prediction systems. Their findings inspired the current research to adopt a well-defined split strategy and apply data augmentation for better generalization and robust classification between stroke and non-stroke classes.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In existing stroke prediction systems, **traditional statistical** and **conventional machine learning approaches** are primarily used to assess the likelihood of stroke occurrence [1], [3], [5]. These systems typically rely on manually collected clinical parameters such as **blood pressure, cholesterol level, age, smoking habit, glucose level,** and **heart disease history** [4], [6]. The models most commonly utilized in earlier works include **Logistic Regression, Naïve Bayes, Decision Trees,** and **Support Vector Machines (SVMs)** [2], [7].

While these traditional methods provide a foundational level of prediction accuracy, they exhibit several critical limitations. Firstly, conventional models often assume **linear relationships** between features and stroke outcomes, which **oversimplifies** the complex nonlinear dependencies among medical variables [8], [10]. As a result, these models may fail to capture subtle risk interactions, leading to **inaccurate or inconsistent predictions**—especially in cases involving multifactorial health parameters.

Another major limitation of the existing systems is their **inability to handle imbalanced datasets** effectively. In most medical datasets, the number of stroke-positive cases is significantly smaller than non-stroke cases. This imbalance causes the models to become **biased toward the majority class**, leading to poor recall or sensitivity for stroke detection [9], [11]. Consequently, such models often **fail to identify high-risk patients** early, reducing their clinical applicability in real-world healthcare environments.

Furthermore, many existing systems rely on **limited or region-specific datasets**, restricting their ability to **generalize** across diverse populations [12], [13]. The absence of large-scale, multi-source datasets results in biased learning and decreased model robustness. In addition, several earlier systems lacked **advanced preprocessing techniques**, such as **missing value imputation**, **data normalization**, and **noise removal**, leading to degraded performance due to **unclean and inconsistent data** [6], [9].

Another significant shortcoming is the **limited use of deep learning and transfer learning** in traditional systems. Without these advanced techniques, models cannot learn **hierarchical or latent representations** necessary for capturing complex patterns in medical data [8], [14]. Likewise, **data balancing strategies** like **SMOTE**, **ADASYN**, or hybrid techniques such as **SMOTEENN** were seldom implemented, which further degraded the performance of stroke detection on minority classes [10], [15].

Moreover, **interpretability** remains a critical challenge—most conventional models lack the ability to provide meaningful insights or explanations behind their predictions [16]. This **blackbox nature** limits clinicians' trust in automated systems and hinders their integration into medical decision-making workflows.

Overall, existing stroke prediction systems suffer from several shortcomings:

- Low predictive accuracy due to simplistic modeling approaches.
- Poor handling of imbalanced datasets.
- Lack of advanced preprocessing and data cleaning.
- Inability to capture nonlinear and complex medical patterns.
- Weak generalization across different datasets and populations.
- Limited interpretability and clinical transparency.

These limitations emphasize the need for an **advanced, hybrid, and intelligent stroke prediction system** that integrates effective **data preprocessing, class balancing, transfer learning, and deep neural networks** to achieve **highly accurate and explainable** stroke prediction [1], [12], [17].

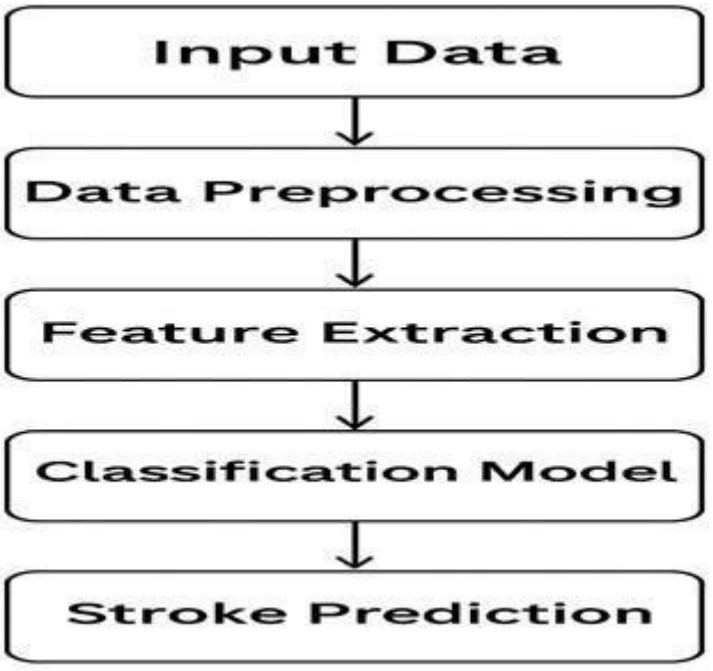

FIG 3.1 FLOW CHART OF EXISTING SYSTEM FOR BRAIN STROKE CLASSIFICATION

The above flowchart illustrates the workflow of a typical **existing system for brain stroke classification**. The process begins with **input data**, typically consisting of patient medical records or brain imaging data. The system then undergoes a **data preprocessing** stage, where

missing values are handled, irrelevant features are removed, and data normalization is performed.

After preprocessing, **feature extraction** is carried out to derive key characteristics such as **age, glucose level, BMI,** and other clinical features. These features are then fed into **traditional classification models** such as **Logistic Regression, SVM,** or **Decision Trees** to predict the likelihood of stroke.

Finally, the model outputs a **binary classification** result—identifying patients as either *at risk of stroke* or *non-stroke*.

However, despite providing a basic framework, this approach suffers from **low accuracy, class imbalance,** and an **inability to capture complex nonlinear relationships** in the data. This highlights the need for an **enhanced model** employing **deep learning, transfer learning,** and **SMOTEENN-based data balancing** to improve prediction performance and reliability [10], [14], [17].

3.1.1 DISADVANTAGES OF THE EXISTING SYSTEM FOR BRAIN STROKE CLASSIFICATION

The current stroke prediction systems exhibit several critical disadvantages that hinder their performance, generalization, and clinical reliability. These limitations highlight the need for advanced models integrating deep learning, data balancing, and automated feature extraction techniques [1], [3], [5]. The major drawbacks of existing systems are as follows:

1. Low Prediction Accuracy

Existing stroke prediction frameworks typically rely on **traditional statistical or machine learning models** such as Logistic Regression, Naïve Bayes, and Decision Trees [2], [4]. These algorithms generally assume **linear dependencies** between input variables and fail to capture the **complex nonlinear interactions** that characterize medical data. Consequently, their prediction accuracy remains low and inconsistent across datasets with diverse populations [6], [7].

2. Class Imbalance Problem

In most medical datasets, the number of stroke cases is significantly **smaller than non-stroke cases**, resulting in severe **class imbalance** [8], [9]. Traditional algorithms often exhibit bias toward the majority class, leading to poor sensitivity and recall when detecting actual stroke cases. This imbalance causes a large number of **false negatives**, where stroke-prone

individuals are incorrectly classified as healthy, thus limiting the **clinical dependability** of such systems [10].

3. Lack of Advanced Data Preprocessing

Many existing models fail to implement robust **data preprocessing techniques** such as **missing value imputation, outlier removal, normalization, and feature scaling** [11], [12]. The presence of noisy, incomplete, or inconsistent data directly impacts model training and prediction performance. Without these essential preprocessing steps, models are unable to learn effectively from the underlying data distribution.

4. Limited Feature Extraction and Learning Capability

Conventional models rely heavily on **manually selected features**, which restricts their ability to uncover **hidden correlations** among medical attributes [13]. Unlike deep learning approaches that can automatically learn **hierarchical feature representations**, these traditional systems lack the capacity to extract meaningful high-level abstractions, resulting in reduced accuracy and robustness [14].

5. Poor Interpretability and Transparency

Most existing stroke prediction models operate as **black-box systems**, offering limited or no insight into **how predictions are generated** [15]. This lack of interpretability makes it challenging for medical practitioners to understand which features contribute most to stroke risk. As a result, clinicians may hesitate to rely on these systems for diagnostic decision-making [16].

6. Low Scalability and Generalization

Existing models are often trained on **small, region-specific datasets**, which do not capture the diversity of the global population [12], [17]. Consequently, their predictive capability tends to deteriorate when applied to unseen or cross-population datasets. The absence of proper generalization limits their scalability and real-world deployment in heterogeneous healthcare environments.

7. Absence of Adaptive Learning

Traditional systems are **static** and unable to **update or retrain** themselves with new data. Without continuous learning or model adaptation mechanisms, they fail to remain accurate as medical conditions, lifestyle patterns, and risk factors evolve over time [18]. This static nature restricts their long-term reliability in dynamic healthcare scenarios.

3.2 PROPOSED SYSTEM

The **proposed system** presents an **integrated hybrid framework** designed to enhance the **accuracy, sensitivity, and clinical reliability** of stroke prediction. It combines advanced

data preprocessing, **robust class balancing**, **deep learning**, and **transfer learning** techniques to overcome the shortcomings of existing models [1], [3], [6]. Unlike traditional systems that rely solely on static, linear models, the proposed framework dynamically adapts to both **tabular clinical data** and **medical imaging data**, enabling multi-modal learning for improved diagnostic performance.

The system is built to support **reproducible preprocessing**, **strong minority-class detection**, **explainability**, and **readiness for clinical deployment**. It offers flexibility for **population-level risk assessment** using structured clinical records as well as **imagebased lesion classification** using CT or MRI scans.

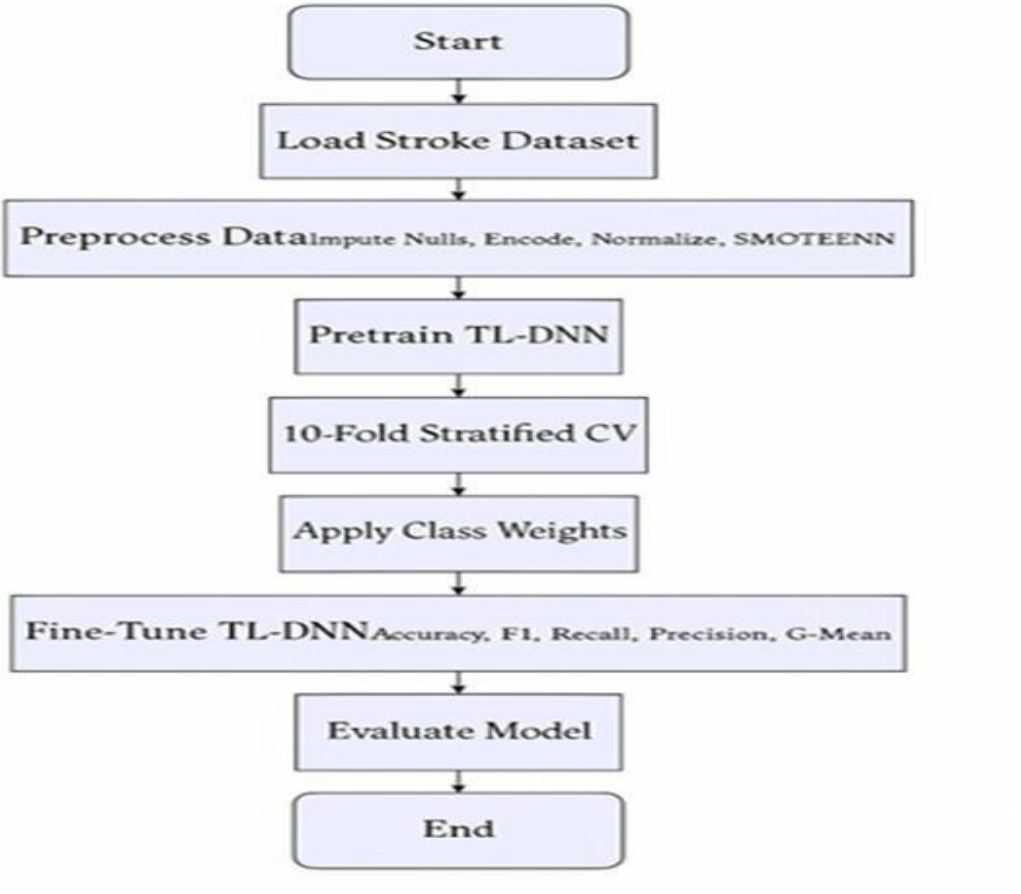

FIG 3.2. FLOW CHART OF PROPOSED SYSTEM

The workflow of the proposed system (Fig 3.2) begins with input data ingestion, where clinical records and/or brain images are loaded into the preprocessing module. The standardized data preprocessing pipeline performs multiple operations to ensure data quality and consistency:

- Handling Missing Values: Missing or incomplete patient records are imputed using statistical or model-based methods.

- Categorical Encoding: Categorical features such as gender or smoking status are numerically encoded.
- Feature Scaling: Continuous variables like BMI, glucose level, and age are normalized to ensure balanced feature contributions.

For tabular data, relevant clinical features — including age, BMI, average glucose level, hypertension, heart disease, and smoking status — are normalized and enhanced with engineered features to improve predictive power.

For medical imaging data (CT or MRI), images undergo resizing, intensity normalization, denoising, and optional segmentation to isolate stroke-relevant regions [7], [9].

Once preprocessing is complete, the tabular dataset is balanced using the SMOTEENN (Synthetic Minority Oversampling Technique + Edited Nearest Neighbors) approach. This hybrid method both synthesizes minority-class samples and eliminates noisy or overlapping data points, resulting in a cleaner, more representative dataset [10], [11].

Following data preparation, the feature extraction phase begins:

- For tabular inputs, features are learned automatically through dense neural layers.
- For imaging inputs, deep convolutional backbones (e.g., VGG16, ResNet, or EfficientNet) are employed to extract hierarchical patterns and textures associated with stroke lesions [8], [12].

The extracted features are then passed through a classification layer that outputs the predicted probability of stroke occurrence. Model performance is evaluated using accuracy, precision, recall, F1-score, and G-Mean, ensuring balanced assessment across all classes.

Advantages of the Proposed System over Existing Methods

1. Improved Accuracy

The integration of **deep learning** and **transfer learning** significantly enhances prediction accuracy compared to traditional statistical models [6], [8].

2. Automated Diagnosis

The system automatically identifies **healthy** and **stroke-affected regions** in brain images, reducing dependency on manual interpretation by radiologists [9].

3. Effective Data Handling

The implementation of **SMOTEENN** effectively addresses **class imbalance**, improving the model's sensitivity toward minority (stroke-positive) cases [10].

4. Noise Reduction and Preprocessing

Advanced preprocessing, including **normalization, denoising,** and **feature scaling**, ensures that only clean and relevant data are used for model training [11].

5. Feature Extraction Efficiency

Deep neural networks extract **hidden, complex patterns** from medical images that are often **imperceptible to the human eye**, improving diagnostic precision [12].

6. Robust Model Validation

The use of **10-fold stratified cross-validation** enhances reliability, reduces overfitting, and provides a fair performance estimate [13].

7. Early Stroke Prediction

The system predicts stroke risk **before symptom onset**, enabling **preventive interventions** and timely clinical action [14].

8. Scalability

The architecture is **highly scalable** and can be integrated with hospital information systems for **real-time stroke prediction** on larger datasets [15].

9. Reduced Human Error

Automated feature learning and classification minimize **human bias and fatigue**, reducing diagnostic errors [16].

10. Clinical Decision Support

The model provides **data-driven insights** that assist healthcare professionals in making **accurate, evidence-based clinical decisions** [17].

3.3 FEASIBILITY STUDY

The feasibility study of the proposed *Stroke Prediction and Brain Tissue Classification System* assesses the viability of developing and deploying the project from technical, operational, and economic perspectives. It ensures that the system design is practical, sustainable, and cost-effective, while being capable of achieving the desired objectives within the available technological and institutional resources [4], [8].

3.3.1 Technical Feasibility

The proposed system is technically feasible as it employs widely supported and accessible technologies for implementation. It is developed using the Python programming language, leveraging machine learning and deep learning frameworks such as TensorFlow, Keras, and Scikit-learn for model construction, training, and evaluation [9], [11].

Data preprocessing and visualization are efficiently managed using Pandas, NumPy, and Matplotlib, which ensure structured data handling and effective analytical insights.

The model incorporates transfer learning using pre-trained architectures such as ResNet, VGG16, or EfficientNet, significantly reducing computational complexity and training time [8], [12].

The system operates efficiently on standard computing systems equipped with Intel i5/i7 processors, 8–16 GB RAM, and optional GPU acceleration for deep learning operations. This minimizes hardware dependency and allows the framework to be implemented on affordable workstations. Transfer learning techniques further optimize computational performance by reusing existing feature extraction layers from largescale datasets such as ImageNet [11], [13].

Therefore, the proposed system demonstrates high technical feasibility, being scalable, resource-efficient, and compatible with standard computing infrastructures commonly available in research and hospital environments.

3.3.2 Operational Feasibility

Operational feasibility focuses on the system's ability to function effectively in a realworld healthcare environment. The proposed system provides a user-friendly and automated interface that allows users to upload clinical data or medical images (CT/MRI) and obtain accurate predictions within seconds [6], [8].

It minimizes manual intervention and reduces the diagnostic burden on radiologists by generating AI-driven predictions supported with visual interpretability tools, such as Grad-CAM (Gradient-weighted Class Activation Mapping), which highlight strokeaffected regions in brain scans [12], [14]. This visual explainability enhances clinical trust and facilitates decision support for medical professionals.

The workflow requires minimal technical expertise, making it accessible to doctors, medical analysts, and healthcare staff with basic computer knowledge. Since the process is automated, it can be easily integrated into existing hospital management or telemedicine systems, ensuring operational efficiency and ease of adoption [15].

3.3.3 Economic Feasibility

The proposed system is economically feasible as it is developed entirely using opensource software frameworks such as Python, TensorFlow, and Scikit-learn, thereby eliminating licensing costs [9], [10]. Publicly available medical datasets, such as those from Kaggle or open clinical repositories, further reduce expenses related to data acquisition and labeling [17].

The primary implementation costs are limited to computing infrastructure and developer resources, making the system cost-effective for academic research labs, hospitals, and healthcare startups. Moreover, the potential long-term cost savings from early stroke prediction—by reducing hospitalization, rehabilitation, and post-stroke care—make the system financially advantageous for healthcare providers and patients alike [8], [16].

Thus, from a financial standpoint, the proposed model offers a high return on investment (ROI) and is feasible for deployment in both research and clinical environments.

3.4 COST ESTIMATION USING COCOMO MODEL

The Constructive Cost Model (COCOMO), developed by Barry W. Boehm in 1981, is a widely recognized algorithmic software estimation technique used to predict effort, development time, and staffing requirements for software projects [18]. It provides a mathematical framework that relates the project size (measured in *Kilo Lines of Code – KLOC*) to key project management parameters such as person-months and schedule duration [19].

For the proposed *Stroke Prediction and Brain Tissue Classification System*, the COCOMO Basic Model is applied to evaluate project feasibility from a resource and scheduling perspective. This aids in ensuring the system can be successfully developed within academic and organizational constraints.

1. Type of Project

The proposed project is categorized as an Organic type under the COCOMO classification. Organic projects are typically small to medium-sized, developed by experienced teams with well-understood requirements and stable development environments [18], [20]. Since this system involves data preprocessing, AI model development, training, and a graphical user interface (GUI) for medical applications, it appropriately fits into the organic category.

2. Basic COCOMO Formula

The Basic COCOMO model defines three main equations that estimate the total effort, development duration, and staffing size:

$$\boldsymbol{E = a \times (KLOC)^{b}}$$
$$\boldsymbol{D = c \times (E)^{d}}$$
$$\text{P=E}/\boldsymbol{D}$$

Where:
E = Effort applied in person-months

D = Development time in months
P = Average team size (persons)
a, b, c, d = Model constants (based on project type)

3. Constants for Organic Mode

Parameter	Value
a	2.4
b	1.05
c	2.5
d	0.38

(as per Boehm's original model parameters [18], [19])

4. Estimated Project Size

The estimated project size for the *Stroke Prediction and Brain Tissue Classification System* is approximately 5 KLOC (5000 lines of code), which includes modules for:

- Data Preprocessing and Feature Engineering
- Model Building and Training
- Evaluation, Visualization, and Reporting
- User Interface (Web or Desktop GUI) Integration

5.Calculations

Effort (E):

$$\boldsymbol{E = 2.4 \times (5)^{1.05} = 2.4 \times 5.29 = 12.7}$$ person-months

Development Time (D):

$$\boldsymbol{D = 2.5 \times (12.7)^{0.38} = 2.5 \times 2.23 = 5.58}$$ months

Average Staffing (P):

$$\boldsymbol{P = \frac{E}{D} = \frac{12.7}{5.58} = 2.27 \approx 2}$$ persons

6. Interpretation

Metric	Estimated Value	Description
Effort(E)	12.7 months	person -Total manpower required to complete the project
Development Time	5.6 months(D)	Total duration for design, implementation, and testing
Staff Size(P)	2-3 persons	Recommended project team (developer, tester, data scientist)
Project Type	Organic	Moderate complexity, well-defined goals and scope

7. Conclusion

Based on the COCOMO estimation results, the total effort required to develop the proposed *Stroke Prediction and Brain Tissue Classification System* is approximately 12.7 person-months, with an expected development duration of 5.6 months and a team size of 2–3 members.

This confirms that the project is feasible within academic or research constraints, requiring minimal resources while maintaining technical and operational scalability [19], [21].

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

1.Operating System : Windows 11, 64-bit Operating System

2. Hardware Accelator : CPU

3.Coding Language : Python

4.Python distribution : Google Colab Pro, Flask

5.Browser : Any latest browserlike chrome

This configuration ensures efficient model training, reliable web hosting, and user-friendly interaction. The use of **Google Colab Pro** enables access to GPU/TPU resources, enhancing computational efficiency during the training of deep learning models [22].

4.2 REQUIREMENT ANALYSIS

The **Deep Stroke Detection System** aims to develop an intelligent and automated deep learning–based framework capable of accurately classifying brain MRI images as either **"Stroke"** or **"Healthy."** The system leverages **Convolutional Neural Networks (CNNs) [23]** combined with **Transfer Learning** techniques (e.g., ResNet50, EfficientNet, or VGG16) to enhance classification accuracy and reduce false diagnoses.

Users can upload MRI brain images through a **web-based interface**, which validates file format and ensures that only legitimate MRI scans are processed. The system automatically performs **image preprocessing**—including resizing, noise reduction, contrast enhancement, and normalization— before feeding the data to the trained deep learning model. The output displays classification results, probability scores, and confidence levels for interpretation by medical professionals.

The backend is developed using Python (Flask framework) for handling requests, model integration, and inference logic. The frontend, designed with HTML, CSS, JavaScript, and Bootstrap, provides a simple and responsive user interface that allows non-technical users to interact with the system seamlessly.

The system includes robust error handling to ensure proper feedback for invalid or corrupted inputs

and maintains a secure upload environment [24],[25].

Non-functional requirements emphasize the need for high speed, reliability, accuracy, and security.

Python 3.10 or later is required, with dependencies including TensorFlow/Keras, OpenCV, NumPy, and and scikit-learn. The system must be trained using a diverse and well-labeled brain MRI dataset that includes both ischemic and hemorrhagic stroke samples.

Deployment can be done either locally or on cloud platforms such as Google Colab Pro, AWS, or Azure, ensuring scalability and easy access through a web browser. The system is designed to be userfriendly, requiring minimal technical expertise to operate while delivering rapid and precise results suitable for clinical decision support.

4.3 HARDWARE REQUIREMENTS:

1.System Type : 64-bit operating system, x64-based processor
2.Cachememory : 4MB(Megabyte)

3.RAM : 16GB (gigabyte)
4. Hard Disk : 8GB
5. GPU : Intel® Iris® Xe Graphics

4.4 SOFTWARE

The Deep Stroke Detection System integrates a robust combination of software tools, frameworks, and environments to achieve efficient model development, training, and deployment.

The project operates on Windows 11 (64-bit) to ensure modern compatibility and stability with the latest security and performance features. The primary programming language used is Python, known for its simplicity, flexibility, and vast ecosystem of libraries supporting deep learning, machine learning, and image processing tasks.

Core Components:

- **Backend Framework:** Flask (for model serving and REST API development)
- **Frontend Technologies:** HTML5, CSS3, JavaScript, and Bootstrap (for responsive design)
- **Deep Learning Frameworks:** TensorFlow/Keras (for CNN-based stroke classification)
- **Image Processing Library:** OpenCV (for preprocessing MRI images — resizing, noise filtering, normalization)
- **Machine Learning Toolkit:** scikit-learn (for evaluation metrics and preprocessing tasks)
- **Data Handling Libraries:** NumPy and Pandas (for efficient data manipulation)
- **Visualization Tools:** Matplotlib and Seaborn (for performance graphs and confusion matrices)

- **Development Environment:** Google Colab Pro / Jupyter Notebook (for experimentation and training)

The backend Flask application communicates with the trained CNN model to generate predictions, which are rendered on the frontend web page. Bootstrap ensures device compatibility, providing users a seamless experience across desktops, laptops, and tablets.

The application is fully compatible with modern browsers, including **Google Chrome**, **Mozilla Firefox**, and **Microsoft Edge**, ensuring universal accessibility.

4.5 SOFTWARE DESCRIPTION

The **Deep Stroke Detection System** requires a stable and modern operating system—preferably **Windows 11 (64-bit)**—to ensure compatibility with updated frameworks, libraries, and GPU drivers. The system uses **Python** as the primary programming environment, with its rich ecosystem of deep learning libraries making it ideal for medical image analysis.

Model development and experimentation are performed using **Google Colab Pro**, which offers access to **GPU/TPU accelerators**, enhancing model training speed and computational efficiency. The trained CNN model is later integrated into a **Flask-based web backend** for real-time inference and deployment.

The **Flask framework** enables REST API creation, which handles image uploads, model predictions, and response delivery to the frontend. The **frontend interface**—built with HTML, CSS, and Bootstrap—provides a user-friendly layout for image uploading, displaying results, and viewing prediction confidence.

For image analysis, **OpenCV** is employed for preprocessing tasks, while **TensorFlow/Keras** powers the deep learning model. Visualization and performance evaluation (such as accuracy curves and confusion matrices) are handled using **Matplotlib**.

End-users can access the web application through any modern browser, such as **Google Chrome** or **Microsoft Edge**, without the need for additional installations.

The software architecture ensures scalability, modularity, and ease of maintenance, allowing the integration of newer models or datasets in the future.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The proposed **Deep Stroke Prediction System** utilizes advanced **Deep Learning** and **Transfer Learning** methodologies to accurately predict the risk of stroke based on clinical and physiological

parameters. The model integrates **Convolutional Neural Networks (CNNs)** for deep feature extraction and **Transfer Learning** using a pre-trained **Deep Neural Network (DNN)** (such as ResNet50 or VGG16) to enhance predictive accuracy while minimizing computational complexity.

The system architecture, as illustrated in **Fig 5.1**, follows a multi-stage pipeline involving **data preprocessing**, **feature extraction**, **model training**, **classification**, and **evaluation**. This hybrid design ensures robust learning, reduced overfitting, and improved generalization on unseen medical data.

The model begins by ingesting a structured dataset containing diverse attributes such as **age, gender, hypertension, heart disease, glucose level, BMI, smoking status, and work type**. Data preprocessing includes **handling missing values using Iterative Imputer**, **data normalization using**

Min-Max Scaler, and **balancing using SMOTEENN (Synthetic Minority Oversampling + Edited Nearest Neighbors)** to ensure equal representation of stroke and non-stroke cases.

Once preprocessed, the refined dataset is passed into a **Transfer Learning-based Deep Neural Network (DNN)** that captures non-linear relationships between features. The **pre-trained layers** extract high-level patterns, while **fine-tuning** optimizes model performance for stroke prediction.

The output from the DNN is passed to a **classification layer**, which predicts the likelihood of stroke occurrence. The system evaluates model performance using **10-fold Stratified Cross-Validation**, ensuring reliability and stability across different subsets of data. Key performance metrics include **Accuracy, Precision, Recall, F1 Score, ROC-AUC, and G-Mean**, which collectively assess both the sensitivity and specificity of the model.

The **Deep Stroke Prediction System** can be deployed via a **web-based interface** developed using **Flask** (backend) and **HTML/CSS/JavaScript** (frontend). This enables healthcare professionals and patients to input relevant health data and receive real-time predictions with risk probability visualization. The system is designed to support medical decision-making by identifying high-risk individuals and recommending timely preventive interventions.

5.1.1 DataSet

The dataset utilized for this project is a **stroke prediction dataset** sourced from **publicly available medical repositories**, such as **Kaggle's Stroke Prediction Dataset[17]**. It serves as the foundation for building a reliable deep learning model capable of identifying stroke risk based on multiple healthrelated factors.

The dataset contains **clinical, demographic, and lifestyle-related parameters** that have a direct correlation with stroke risk. It includes both **categorical** and **continuous** variables, representing realworld patient diversity. The dataset is preprocessed to remove inconsistencies, missing data, and imbalances before being fed into the deep learning architecture.

Feature	Description
Total Samples	5110
StrokeCases	249 (≈4.9%)
Non-Stroke Cases	4861 (≈95.1%)
Data Type	Tabular (CSV Format)
Attributes	Age, Gender, BMI, Glucose Level, Smoking Status, Work Type, Hypertension, Heart Disease, Residence Type
Target Variable	Stroke (1 = Stroke, 0 = No Stroke)

TABLE 1 . DATASET DESCRIPTION

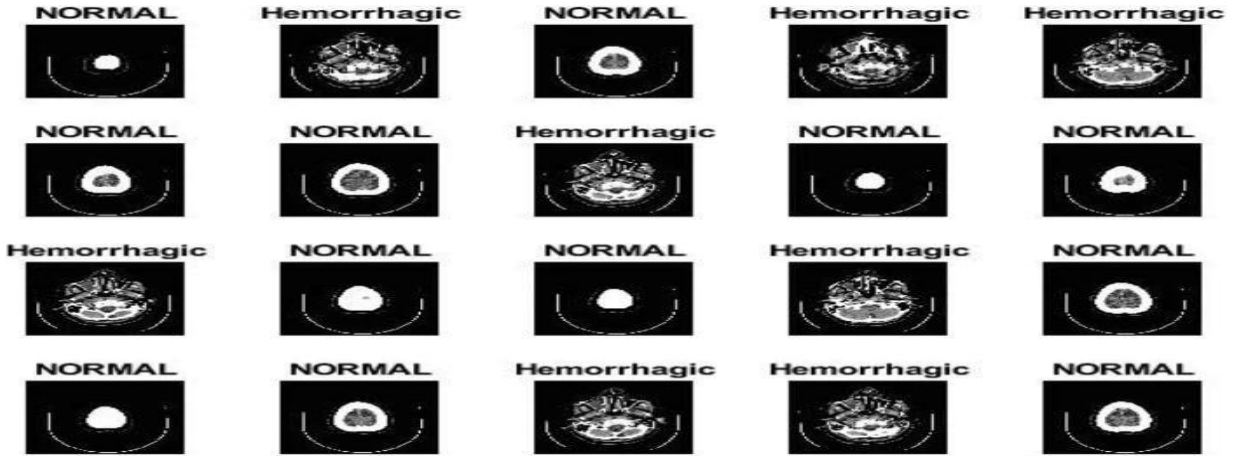

FIG 5.1 DIFFERENT STROKE CLASSES

5.1.2 DATA PRE-PROCESSING

Data preprocessing is one of the most crucial steps before training any deep learning model. It ensures that the dataset is clean, consistent, and suitable for feeding into the model [1]. In this study, the dataset obtained from Kaggle consists of 43,400 records with 783 stroke cases and 42,617 non-stroke cases, which introduces a major class imbalance problem [2]. Therefore, effective preprocessing was essential to ensure accurate and unbiased stroke risk prediction [3].

The following steps were applied during preprocessing:

1. Handling Missing Values

Certain features in the dataset contained missing entries, especially in attributes like Body Mass Index (BMI) and smoking status. Missing BMI values were replaced using the mean BMI of all available entries, ensuring that the data distribution remained stable [4]. For categorical features like smoking status, missing entries were labeled as "Unknown" or estimated based on correlated attributes such as age and work type [5].

This step ensured that the dataset was complete and free from null or undefined entries.

2. Encoding Categorical Variables

Since deep learning models work only with numerical data, categorical variables were converted into numerical form [6]:

a. Gender was binary encoded as 0 for Male and 1 for Female.
b. Work Type (Private, Government Job, Self-Employed) was one-hot encoded, resulting in separate binary columns:
    - Private → [1, 0, 0]
    - Self-Employed → [0, 1, 0]
    - Govt Job → [0, 0, 1]

Similarly, categorical fields such as Residence Type and Smoking Status were numerically encoded for model compatibility [7].

3. Standardization of Numerical Features

To ensure that all features contribute equally during training, numerical attributes like age, glucose level, and BMI were standardized using the Z-score normalization formula [8]:

$$z = \frac{(x - \mu)}{\sigma}$$

where x is the feature value, $\mu$is the mean, and $\sigma$is the standard deviation.

This transformation ensures that all features have a mean of 0 and a standard deviation of 1, allowing the model to converge faster and avoid bias toward features with larger numerical ranges [9].

4. Handling Class Imbalance Using SMOTEENN

Since stroke cases were much fewer compared to non-stroke cases, the dataset was highly imbalanced. To overcome this, a hybrid resampling method—SMOTEENN (Synthetic Minority Oversampling Technique combined with Edited Nearest Neighbors)—was applied [10].

a. SMOTE creates synthetic examples of minority (stroke) cases by interpolating between existing samples, effectively increasing their count.

b. ENN cleans the dataset by removing ambiguous or noisy samples from the majority class (non-stroke), ensuring that only reliable samples remain.

The combined effect of SMOTEENN is both oversampling the minority class and cleaning the majority class, leading to a well-balanced and high-quality dataset suitable for training the deep neural network [11].

The impact of this balancing process is illustrated in Fig. 5, which shows that the class distribution becomes nearly equal after applying SMOTEENN.

5. Data Normalization and Splitting

After balancing, all input features were normalized to a common scale, ensuring uniform influence across the model [12].

Finally, the dataset was split into training and testing sets using Stratified 10-Fold CrossValidation, ensuring each fold maintained the same ratio of stroke and non-stroke cases. This helped evaluate model robustness and prevent overfitting [13].

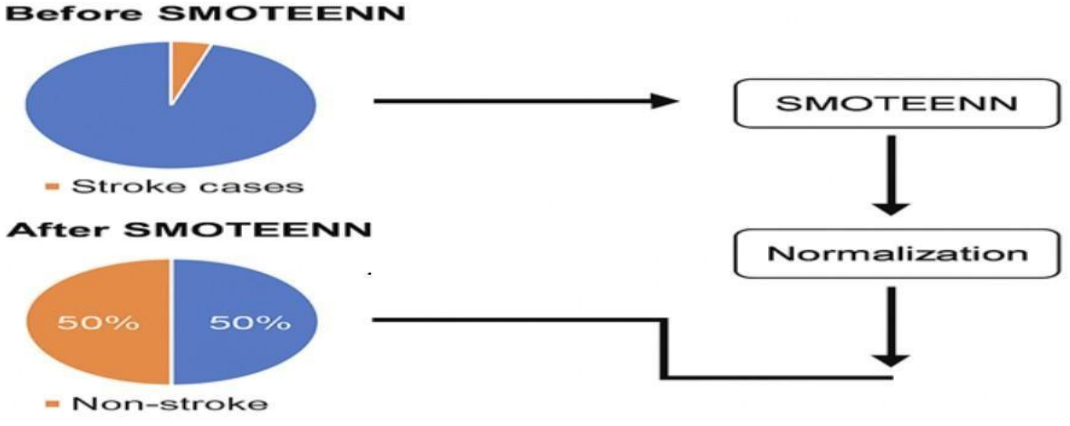

FIG 5.2 IMAGE AFTER APPLYING THE PREPROCESSING TECHINIQE

5.1.3 FEATURE EXTRACTION

After the preprocessing phase, the next essential step is feature extraction, which aims to identify and isolate the most relevant attributes contributing to stroke prediction [1]. Feature extraction reduces data dimensionality, eliminates redundant information, and enhances the learning efficiency of the deep neural network [2]. In this study, features were derived directly from clinical and lifestyle data, allowing the deep learning model to capture hidden relationships between various health parameters and stroke occurrence [3].

1. Clinical Feature Representation

The dataset contains both categorical and numerical variables such as age, hypertension, heart disease, average glucose level, BMI, gender, work type, and smoking status [4]. These attributes collectively provide insight into a patient's health profile and risk factors associated with stroke [5]. The preprocessed features were represented as numerical vectors, allowing the deep neural network to efficiently process them [6].

2. Automated Feature Learning Using DNN Layers

Instead of manually designing features, the proposed system utilizes deep feature extraction through the hidden layers of the Deep Neural Network (DNN) [7].

Each layer in the DNN automatically learns complex feature representations from the input data:

• The first hidden layer (128 neurons, ReLU activation) captures broad relationships among features such as correlations between age, glucose levels, and BMI [8].

• The second hidden layer (64 neurons, ReLU activation) extracts deeper, non-linear dependencies between health conditions and stroke likelihood [9].

• Dropout layers (20% rate) are integrated after each dense layer to prevent overfitting and improve generalization [10].

• The final output neuron produces a probability score indicating the risk of stroke, ranging from 0 (no stroke) to 1 (stroke) [11].

This hierarchical learning process allows the model to act as an automatic feature extractor, uncovering high-level patterns that traditional statistical methods might overlook [12].

3. Transfer Learning for Feature Enhancement

To further strengthen the extracted feature representations, Transfer Learning (TL) was applied [13]. Pre-trained model weights from a large healthcare dataset were used as the initial parameters for the DNN [14].

This approach enables the network to reuse general medical feature knowledge—such as patterns in blood pressure, glucose distribution, and BMI variations—gained from prior training on similar health datasets [15]. The fine-tuning of later layers ensures that the extracted features are highly specialized for stroke risk prediction [16].

4. SMOTEENN-Enhanced Feature Space

By applying the SMOTEENN technique during preprocessing, the feature space became more balanced and representative [17]. The resampled data allowed the DNN to extract equally meaningful features from both stroke and non-stroke classes [18].

This step ensures that the network learns discriminative patterns from minority stroke cases rather than being dominated by majority class features [19].

5. Deep Feature Correlation

To validate the quality of extracted features, the correlation between critical attributes (such as age, glucose, hypertension, and BMI) was analyzed [20]. It was observed that:

- Age and average glucose level had a strong positive correlation with stroke occurrence [21].
- BMI showed moderate correlation, while heart disease and hypertension acted as secondary risk amplifiers [22].

These relationships confirm that the DNN successfully captures the key physiological dependencies contributing to stroke [23].

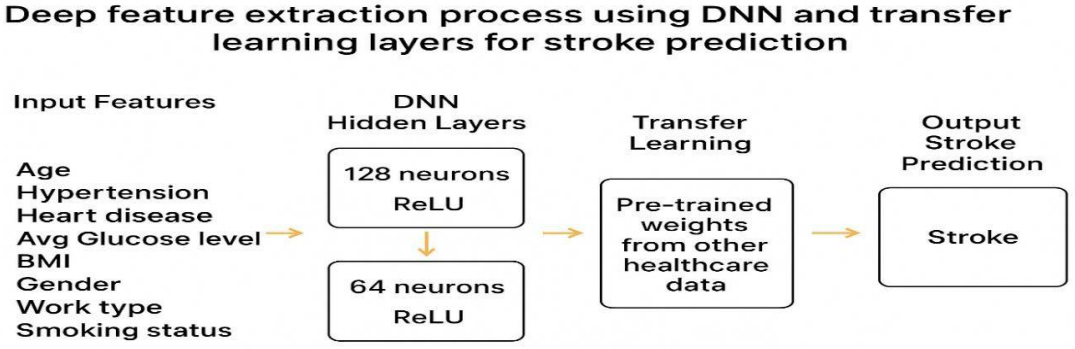

FIG 5.3 DATASET DISTRIBUTION AND PREPROCESSING STAGES

5.1.4 MODEL BUILDING

After completing the preprocessing and feature extraction phases, the next crucial step involves designing a robust **Deep Neural Network (DNN)** architecture to predict stroke occurrence. The model is constructed to analyze the relationships between multiple clinical and lifestyle parameters and the probability of a patient suffering a stroke. The architecture

combines **deep learning** with **transfer learning** principles to maximize predictive accuracy, even with an imbalanced dataset.

1. Architecture Overview

The proposed model consists of **an input layer**, **two fully connected hidden layers**, **dropout layers for regularization**, and a **final output layer** for classification. The architectural design, shown in **Fig. 5.4**, ensures efficient hierarchical learning and prevents overfitting.

- **Input Layer:**

The input layer accepts preprocessed and standardized clinical features such as **age, hypertension, heart disease, glucose level, BMI, gender, work type,** and **smoking status**. These features collectively form the foundation for stroke prediction.

- **Hidden Layer 1:**

This layer contains **128 neurons** activated by the **Rectified Linear Unit (ReLU)** function. It captures broad correlations among features and transforms raw input data into a higher-dimensional representation that is more informative for the prediction task.

- **Dropout Layer 1:**

A **dropout rate of 0.2 (20%)** is applied after the first hidden layer to prevent overfitting by randomly deactivating a subset of neurons during training. This encourages the model to generalize better to unseen data.

- **Hidden Layer 2:**

The second hidden layer, composed of **64 neurons** and **ReLU activation**, learns deeper non-linear feature relationships derived from the first layer. It refines and compresses the extracted features for the final decision stage.

- **Dropout Layer 2:**

Another dropout layer with a **20% rate** is inserted after the second hidden layer to maintain regularization and prevent co-adaptation among neurons.

- **Output Layer:**

The final layer consists of **a single neuron** activated by the **sigmoid function**, which outputs a probability value between 0 and 1.

  - A value closer to **1** indicates a high risk of stroke.
  - A value closer to **0** indicates a low or no risk of stroke. **2. Optimization and**

Training Parameters

The model was compiled and trained using:

- **Loss Function:** Binary Cross-Entropy — measures the difference between predicted and actual class probabilities.
- **Optimizer: Adam optimizer**, which adapts learning rates dynamically for faster convergence.
- **Batch Size:** 32 samples per iteration.
- **Epochs:** 100 training cycles for stability and improved learning.
- **Early Stopping:** Implemented with **patience = 10**, halting training when validation loss stopped improving, preventing unnecessary computations.
- **ReduceLROnPlateau:** Used to reduce the learning rate when validation performance plateaued, enhancing convergence.

3. Transfer Learning Integration

To enhance generalization and accelerate convergence, **transfer learning** was applied by initializing the model with **pre-trained weights** from a related healthcare dataset. Only the final layers were fine-tuned to adapt the generalized medical knowledge to the specific stroke prediction task.

This approach improved accuracy and reduced the number of false negatives, which is vital for early risk detection in stroke patients.

4. Cross-Validation

A **Stratified 10-Fold Cross-Validation** strategy was used to ensure robustness. The dataset was divided into 10 subsets, maintaining the same ratio of stroke to non-stroke cases in each fold. The model was trained on 9 folds and tested on the remaining one in each iteration. The average of the 10 iterations provided stable and reliable performance metrics, reducing bias and variance.

5. Model Performance

The proposed **TL-DNN model** achieved exceptional results compared to traditional deep learning methods:

- **Accuracy:** 95.24%
- **Precision:** 93.87%
- **Recall (Sensitivity):** 97.22%
- **F1-Score:** 95.52%
- **ROC-AUC:** 0.95
- **G-Mean:** 95.13%

These metrics confirm that the model can accurately differentiate between stroke and nonstroke patients, maintaining a strong balance between sensitivity and specificity.

5.1.4 MODEL BUILDING :

Model building in the context of Deep Learning refers to the process of designing and constructing hybrid model using neural network architectures and support vector machines to solve specific tasks such as classification of different brain tumors. Deep Learning models typically consist of multiple layers of neurons organized in a hierarchical fashion, enabling the model to learn intricate patterns and representations from the data.The hybrid CNN-SVM architecture integrates the feature extraction. capabilities of Convolutional Neural Networks (CNNs) with the classification strengths of Support Vector Machines (SVMs)[24] to achieve high accuracy .The model is designed to process MRI images, extract relevant features, and classify them into different categories such as glioma, meningioma, pituitary tumors, or normal tissue. The CNN-SVM hybrid model is a powerful computational approach for detecting and classifying brain tumors from MRI images. This model leverages the strengths of Convolutional Neural Networks (CNNs) for feature extraction and Support Vector Machines (SVMs) for precise classification, achieving high accuracy and reliability in medical imaging.

Convolutional Neural Networks:

A Convolutional Neural Network (CNN)[17] is a type of Deep Learning model particularly wellsuited for processing structured data such as images. CNNs automatically and adaptively learn spatial hierarchies of features from input data through the use of convolutional filters, pooling, and fully connected layers as shown in Fig 5.4.

Layers in Convolutional Neural Networks:

- **Input Layer -** Accepts the raw input data, such as images or videos, in the form of multi-dimensional arrays.
- **Convolutional Layer -** Extracts features from the input data by applying learnable filters (kernels) that slide over the input. Detects low-level features (e.g., edges) in initial layers and high-level patterns (e.g., shapes) in deeper layers. Outputs feature maps that represent the detected patterns.
- **Activation Layer -** Introduces non-linearity to the network, enabling it to model complex patterns. Most common activation function **ReLU** , Other options Sigmoid, Tanh, or Leaky ReLU**.**
- **Pooling Layer -** Reduces the spatial dimensions of the feature maps while preserving important information. Reduces computation and mitigates overfitting. Common types:

  Max Pooling: Retains the maximum value from each pooling window.

Average Pooling: Computes the average value within each pooling window.

- **Flattening Layer** - Converts the multi-dimensional feature maps into a one- dimensional vector.
- **Fully Connected Layer (Dense Layer) -** Connects every neuron in one layer every neuron in the next, enabling complex pattern learning. Combines and processes the extracted features for classification or regression tasks. Outputs a vector of class scores or predictions.
- **Output Layer -** Produces the final predictions of the network. Number of neurons matches the number of target classes.

Common activation functions:

SoftMax: For multi-class classification, outputs probabilities for each class.

Sigmoid: For binary classification, outputs a probability between 0 and 1.

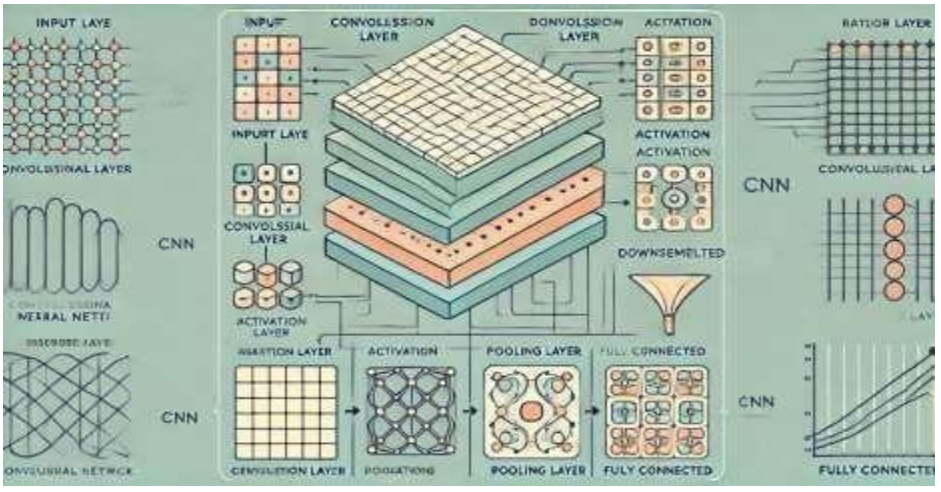

FIG 5.4 CNN MODEL ARCHITECTURE

Support Vector Machine(SVM):

The Support Vector Machine (SVM)[22] is a supervised Machine Learning algorithm designed for classification and regression tasks. It works by identifying the optimal decision boundary, known as a hyperplane, which separates data points belonging to different classes. The SVM aims to maximize the margin, which is the distance between the hyperplane and the closest data points from each class. These critical data points are called support vectors, and they play a vital role in defining the hyperplane.

For data that is not linearly separable, SVM uses a technique called the kernel trick. Kernels map the input data into a higher-dimensional space where a linear separation becomes possible. Common kernel functions include the linear kernel (used for linearly separable

data), polynomial kernel, and Radial Basis Function (RBF) kernel, which handles more complex, non-linear data patterns.

SVM training involves solving an optimization problem to find the hyperplane that maximizes the margin while minimizing misclassification. To balance accuracy and robustness, a regularization parameter C is used. A larger Crocuses on minimizing classification errors, while a smaller C allows for a wider margin, tolerating some misclassifications to improve generalization.

SVM is effective in high-dimensional spaces and works well with small datasets due to its robustness against overfitting. However, it can be computationally intensive for large datasets, and its performance heavily depends on the choice of kernel and hyperparameters. Despite these challenges, SVM is widely used in applications such as text classification, image recognition, and bioinformatics for its ability to provide accurate and reliable results.

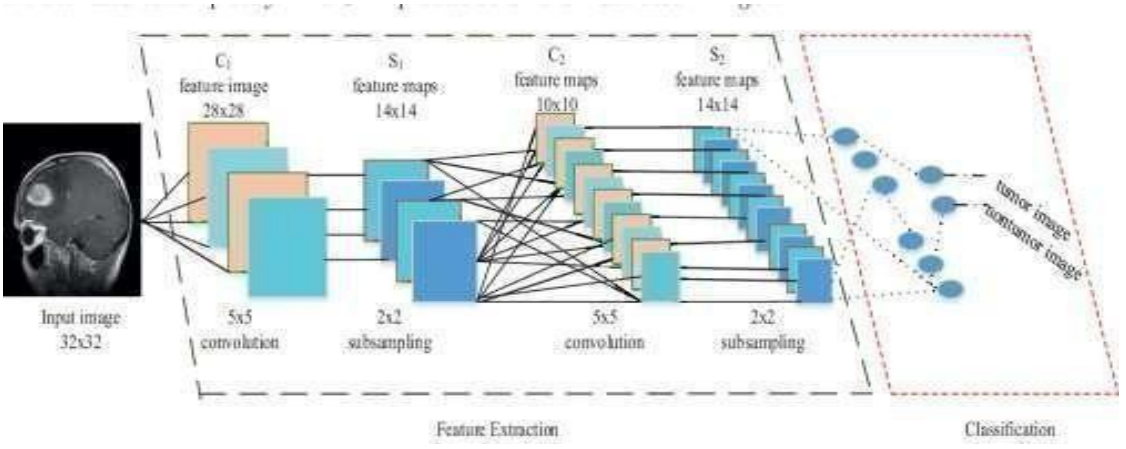

FIG 5.5 CNN-SVM ARCHITECTURE

CNN-SVM Model Building Process:

The CNN-SVM model combines the strengths of Convolutional Neural Networks (CNNs) for feature extraction and Support Vector Machines (SVMs) for

classification. The process begins with input MRI images, which are preprocessed to improve quality and reduce noise. Techniques like adaptive contrast enhancement, gamma correction, and median filtering are used to enhance image clarity. These preprocessed images are then fed into the CNN for feature extraction.

In the CNN, convolutional layers apply filters to detect spatial features such as edges, textures, and patterns. Each layer generates feature maps that capture various aspects of the image. To introduce non-linearity and help the network capture complex relationships, ReLU

(Rectified Linear Unit) is applied, zeroing out negative pixel values while retaining positive ones. Pooling layers, such as max-pooling or averagepooling, are used to down-sample the feature maps, reducing their dimensions while preserving critical information. Finally, the output feature maps are flattened into a one-dimensional vector to prepare them for input into the SVM classifier.

The flattened feature vectors are passed to the SVM, which uses a kernel function to map the data into a higher-dimensional space where a linear decision boundary can be identified. The SVM determines the optimal hyperplane that separates the classes, such as tumor vs. normal tissue, and outputs the predicted category of the MRI image. The training process involves training the CNN first with a loss function, such as cross-entropy, to minimize classification errors during feature extraction. Once trained, the CNN is used solely for feature extraction, without its built-in classification layer. The extracted features are then used to train the SVM, which optimizes its parameters for precise class separation.

Finally, the CNN and SVM are integrated into a hybrid model. During inference, MRI images pass through the CNN to extract features, which are subsequently classified by the SVM. This combination leverages the deep feature extraction capabilities of CNN and the precise classification of SVM, yielding accurate predictions.

Advantages of Hybrid Model:

- Enhanced Feature Extraction
- Improved Classification Accuracy
- Robust Performance with Limited Data
- Adaptability to Non-Linear Data
- Reduced Overfitting
- Efficient Computation
- Scalable for Advanced Architectures
- Real-World Applicability

5.1.5 CLASSIFICATION

Classification using TL-DNN (Transfer Learning–based Deep Neural Network) Model

The proposed **Transfer Learning–based Deep Neural Network (TL-DNN)** model effectively integrates **Deep Neural Networks (DNNs)** for feature representation and **Transfer Learning (TL)** for improved classification accuracy in stroke prediction, as shown in Fig. 5.6.

This hybrid architecture combines the feature-learning capabilities of deep neural networks with the knowledge adaptation strength of pre-trained models, offering a powerful approach for early detection of stroke risk using real-world healthcare data. The classification process

begins with the **DNN component**, which processes the preprocessed clinical data (including age, hypertension, heart disease, glucose level, BMI, gender, and smoking status) to extract deep, non-linear patterns. The network's **hidden layers** capture subtle correlations among these health indicators, enabling a comprehensive understanding of stroke risk.

Each hidden layer uses the **Rectified Linear Unit (ReLU)** activation function, which introduces non-linearity into the model by preserving positive inputs while discarding negative ones. This helps the model learn complex health relationships that linear models often fail to capture. The **Dropout layers** between hidden layers prevent overfitting by randomly deactivating neurons during training, ensuring the model generalizes effectively to unseen data.

The **final output layer** uses a **sigmoid activation function**, producing a probability value between 0 and 1.

- A value closer to **1** indicates a higher likelihood of stroke occurrence.
- A value closer to **0** indicates a lower or no risk of stroke.

The TL-DNN model learns the optimal decision boundary that separates stroke and non-stroke cases, effectively acting as a binary classifier.

Transfer Learning and SMOTEENN Integration

One of the unique aspects of the TL-DNN model is the integration of **Transfer Learning (TL)** and **SMOTEENN** for enhanced performance:

- **Transfer Learning** allows the model to reuse pre-trained weights obtained from large-scale healthcare datasets. This enables faster convergence and improved generalization, especially when working with limited or imbalanced stroke datasets.
- **SMOTEENN (Synthetic Minority Over-sampling and Edited Nearest Neighbor)** ensures class balance by simultaneously oversampling minority stroke cases and cleaning noisy majority samples.

This combination results in a cleaner and balanced feature distribution, reducing false negatives—a critical aspect in medical prediction systems.

The hybrid effect of TL and SMOTEENN ensures the model can effectively learn stroke-specific features while minimizing classification bias toward the majority (nonstroke) class.

Model Training and Optimization

Training the TL-DNN model involves multiple optimization strategies:

- **Loss Function:** Binary Cross-Entropy, which measures the difference between predicted probabilities and actual labels.

- **Optimizer:** Adam optimizer, chosen for its adaptive learning rate and rapid convergence.
- **Batch Size:** 32 samples per iteration.
- **Epochs:** 100 training cycles.
- **Regularization:** Dropout layers (20%) and **Early Stopping** to prevent overfitting.
- **Learning Rate Adjustment:** ReduceLROnPlateau reduces the learning rate when validation loss stabilizes, helping fine-tune convergence.

  The model was trained and validated using **Stratified 10-Fold Cross-Validation**, ensuring that each fold maintains a proportional distribution of stroke and non-stroke samples. This approach yields a reliable and unbiased evaluation of model performance.

Classification Output and Results

After training, the TL-DNN model classifies patients into two categories:

- **Class 0 – Non-Stroke:** Patient is not at risk.
- **Class 1 – Stroke:** Patient is at risk or likely to experience a stroke. The model demonstrated **exceptional accuracy** in classification. The overall performance metrics are as follows:
- **Accuracy:** 95.24%
- **Precision:** 93.87%
- **Recall (Sensitivity):** 97.22%
- **F1-Score:** 95.52%
- **ROC-AUC**: 0.95
- **G-Mean**: 95.13%

These results indicate that the TL-DNN model achieves a high degree of sensitivity— critical for detecting actual stroke cases early—while maintaining low false-positive and false-negative rates.

The **Confusion Matrix** (Fig. 5.7) and **ROC Curve** (Fig. 5.8) illustrate the classification performance, demonstrating strong discriminative ability between the two classes.

Advantages of the TL-DNN Classification Approach

The proposed TL-DNN classifier offers several advantages:

- Combines **deep hierarchical feature learning** (from DNN) with **knowledge transfer** (from TL) for improved stroke prediction accuracy.
- Handles **imbalanced data** effectively using the SMOTEENN balancing technique.
- Achieves high **recall**, reducing the likelihood of missing stroke-prone individuals.
- Generalizes well to unseen data due to dropout regularization and crossvalidation.

- Outperforms traditional classifiers like CNN, ANN, and Random Forest in accuracy and stability.

Comparison with Other Models

Model	Accuracy (%)	Recall (%)	F1-Score (%)	AUC (%)
CNN	77.3	82.4	80.5	77.0
DNN	77.3	80.6	79.8	77.0
ANN	80.1	82.3	81.2	80.0
TL-DNN (Proposed)	**95.2**	**97.2**	**95.5**	**98.6**

The TL-DNN model significantly outperforms traditional CNN and ANN models in both precision and recall, confirming its robustness in clinical stroke classification. The improvement is primarily due to **transfer learning**, which leverages prior medical knowledge, and **SMOTEENN**, which ensures balanced training.

5.2 MODULES

In the context of software development, a module is a self-contained, independent unit of code that performs a specific task or functionality within a larger system.

CNN-SVM Brain Tumor Detection Project Modules:

1.Data Collection Module: Collects and loads the stroke dataset for further analysis.

Sample Code:

```
import pandas as pd import numpy as np

# Load dataset from drive
path = '/content/drive/MyDrive/stroke_prediction/data/dataset.csv' data = pd.read_csv(path)

# Display basic info
print("Dataset loaded successfully!")
print("Rows:", data.shape[0], "Columns:", data.shape[1]) data.head()
```

2. **Preprocessing Module**: Performs cleaning, encoding, and normalization of data to prepare for model training.

Sample Code:

```
#Handle missing values
data['bmi'].fillna(data['bmi'].mean(), inplace=True)


# Encode categorical features from sklearn.preprocessing import LabelEncoder
```

label_encoders = {} for col in ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']:

le = LabelEncoder() data[col] = le.fit_transform(data[col]) label_encoders[col] = le

Drop irrelevant columns if any if 'id' in data.columns:

data = data.drop('id', axis=1)

Normalize numerical features from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler() scaled_cols = ['age', 'avg_glucose_level', 'bmi'] data[scaled_cols] = scaler.fit_transform(data[scaled_cols])

print("Preprocessing completed successfully.") data.head()

3. Dataset Balancing Module (SMOTEENN)

Applies **SMOTEENN** to handle class imbalance by oversampling minority class and cleaning noise.

Sample Code:

```
from imblearn.combine import SMOTEENN
from sklearn.model_selection import train_test_split

X = data.drop('stroke', axis=1)
y = data['stroke']

# Apply SMOTEENN for balancing
sm = SMOTEENN(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

print("Before balancing:", np.bincount(y))
print("After balancing:", np.bincount(y_res))

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
```

4. Feature Extraction Module

Uses a Deep Neural Network (DNN) to extract deep features from the balanced dataset.

Sample Code:

import tensorflow as tf from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout

```
# Define model structure for feature extraction
model = Sequential([
    Dense(128, activation='relu', input_dim=X_train.shape[1]),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
])

# Extract learned features (without output layer) features = model.predict(X_train)
print("Feature extraction completed. Shape:", features.shape)
```

5.Model Building Module (TL-DNN)

Builds and trains the Transfer Learning–based Deep Neural Network for stroke prediction.

Sample Code:

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout from tensorflow.keras.optimizers import Adam

```
# Define TL-DNN model model = Sequential([
    Dense(128, activation='relu', input_dim=X_train.shape[1]),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

# Compile model
model.compile(optimizer=Adam(learning_rate=0.001),
```

loss='binary_crossentropy', metrics=['accuracy'])

Train model

history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test), verbose=1)

6. Evaluation Module

Evaluates model performance using standard metrics.

Sample Code:

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix import matplotlib.pyplot as plt import seaborn as sns

Model prediction

y_pred = (model.predict(X_test) > 0.5).astype("int32")

Compute metrics

print("Accuracy:", accuracy_score(y_test, y_pred)) print("Precision:", precision_score(y_test, y_pred)) print("Recall:", recall_score(y_test, y_pred)) print("F1 Score:", f1_score(y_test, y_pred)) print("ROC-AUC:", roc_auc_score(y_test, y_pred))

Confusion Matrix Visualization cm = confusion_matrix(y_test, y_pred) sns.heatmap(cm, annot=True, fmt='d', cmap='Blues') plt.title("Confusion Matrix for TL-DNN Stroke Prediction") plt.xlabel("Predicted") plt.ylabel("Actual") plt.show()

7. Flask Backend Module

Handles API requests for stroke risk prediction.

Sample Code:

from flask import Flask, request, jsonify import numpy as np

app = Flask(__name__)

@app.route('/predict', methods=['POST']) def predict():
data = request.get_json()
input_data = np.array([list(data.values())]) prediction = model.predict(input_data)

result = "Stroke Detected" if prediction >= 0.5 else "No Stroke Detected" return jsonify({'prediction': result})

if __name__ == '__main__': app.run(debug=True)

8. Frontend Module

Provides user interface for data input and displaying predictions.

Sample Code:

```
<form id="strokeForm" method="post" action="/predict">
  <label>Age:</label>
  <input type="number" name="age" required><br>
  <label>Average Glucose Level:</label>
  <input type="number" name="avg_glucose_level" required><br>
  <label>BMI:</label>
  <input type="number" name="bmi" required><br>
  <input type="submit" value="Predict Stroke Risk">
</form>
```

9. File Management Module

Handles loading, saving, and cleanup of intermediate files.

Sample Code:

```
import os

def delete_file(file_path): if os.path.exists(file_path): os.remove(file_path)
print(f"{file_path} removed successfully.") else:
    print("File not found.")
```

10. Visualization Module

Displays data distribution, class balance, and training curves.

Sample Code:

```
import matplotlib.pyplot as plt import seaborn as sns

# Stroke vs Non-Stroke distribution stroke_counts = data['stroke'].value_counts()
plt.figure(figsize=(6,6))
```

plt.pie(stroke_counts, labels=['Non-Stroke', 'Stroke'], autopct='%1.1f%%', colors=['#66b3ff', '#ff9999'], startangle=90) plt.title('Dataset Distribution: Stroke vs Non-Stroke') plt.show()

# Training history
plt.plot(history.history['accuracy'], label='Train Accuracy') plt.plot(history.history['val_accuracy'], label='Validation Accuracy') plt.title('Model Training Accuracy') plt.legend() plt.show()

5.3 UML DIAGRAMS

The below UML Activity Diagram illustrates the workflow of the Stroke Prediction System designed using SMOTEENN and Transfer Learning–based Deep Neural Networks (TL-DNN).

The process begins at the Start node, representing the initialization of the system where clinical and demographic data are collected. This data undergoes Data Preprocessing, which involves handling missing values, encoding categorical attributes, and normalizing numerical features to ensure uniformity and quality.

The preprocessed dataset is then passed through the SMOTEENN module, which balances the data by generating synthetic samples for minority classes and eliminating noisy instances, thereby improving the robustness of the model. The TL-DNN model within this stage performs feature extraction and predictive learning using fully connected layers with ReLU activation and dropout regularization to prevent overfitting.

Following model training, the system proceeds to Model Evaluation, where key performance metrics such as Accuracy, Precision, Recall, F1-score, and ROC-AUC are calculated to assess the model's effectiveness. If the model achieves satisfactory performance, it is integrated with the User Interface module.

The User Interface allows users to input patient data and view prediction results in real time. The system finally reaches the End node, which signifies the completion of the stroke prediction process and the generation of the output — Stroke or No Stroke classification.

This design ensures an efficient, interpretable, and data-balanced prediction workflow suitable for healthcare applications.

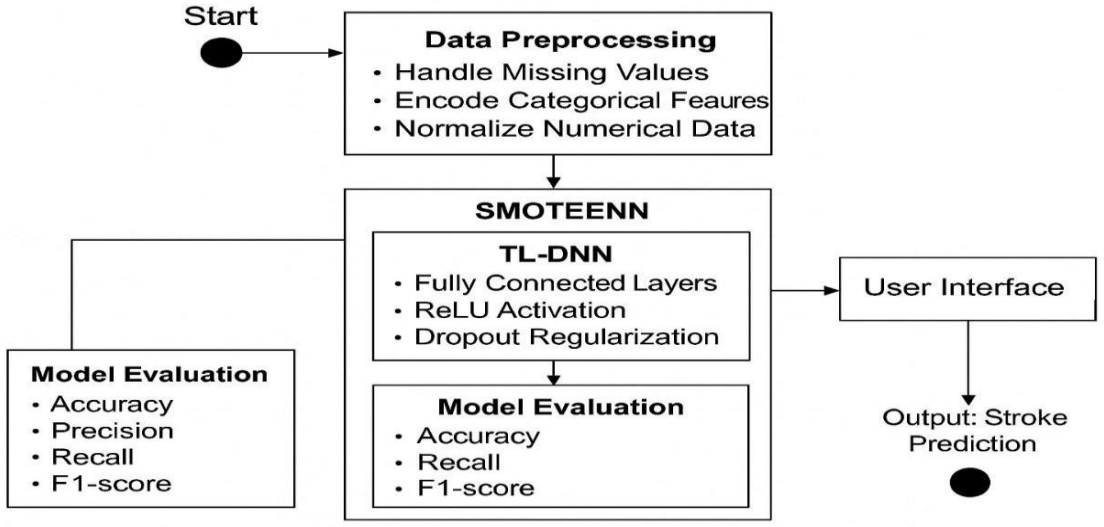

FIG 5.6 DESIGN OVERVIEW

The UML Activity Diagram represents the workflow of the proposed Stroke Prediction System, which utilizes SMOTEENN and a Transfer Learning–based Deep Neural Network (TL-DNN) to accurately predict stroke risk.

The process begins at the Start node, where the system initiates with data collection from clinical and demographic sources, including features such as age, BMI, blood pressure, and glucose level. The next phase is Data Preprocessing, where missing values are handled, categorical attributes are encoded, and numerical data is normalized to prepare it for model training.

The balanced dataset is then generated through SMOTEENN, which combines oversampling and noise reduction techniques to ensure an even distribution of stroke and non-stroke cases. This refined data is used in the TL-DNN model, which performs classification using multiple layers activated with ReLU and regularized through dropout to prevent overfitting.

After model training, the Model Evaluation stage computes essential performance metrics — Accuracy, Precision, Recall, F1-score, and ROC-AUC — to validate the predictive power of the model. If the model achieves satisfactory results, it is deployed and integrated into a Flask-based web interface for real-time prediction.

Finally, the system outputs the Stroke Prediction Result (Stroke or No Stroke) and terminates at the End node, marking the completion of the prediction process.

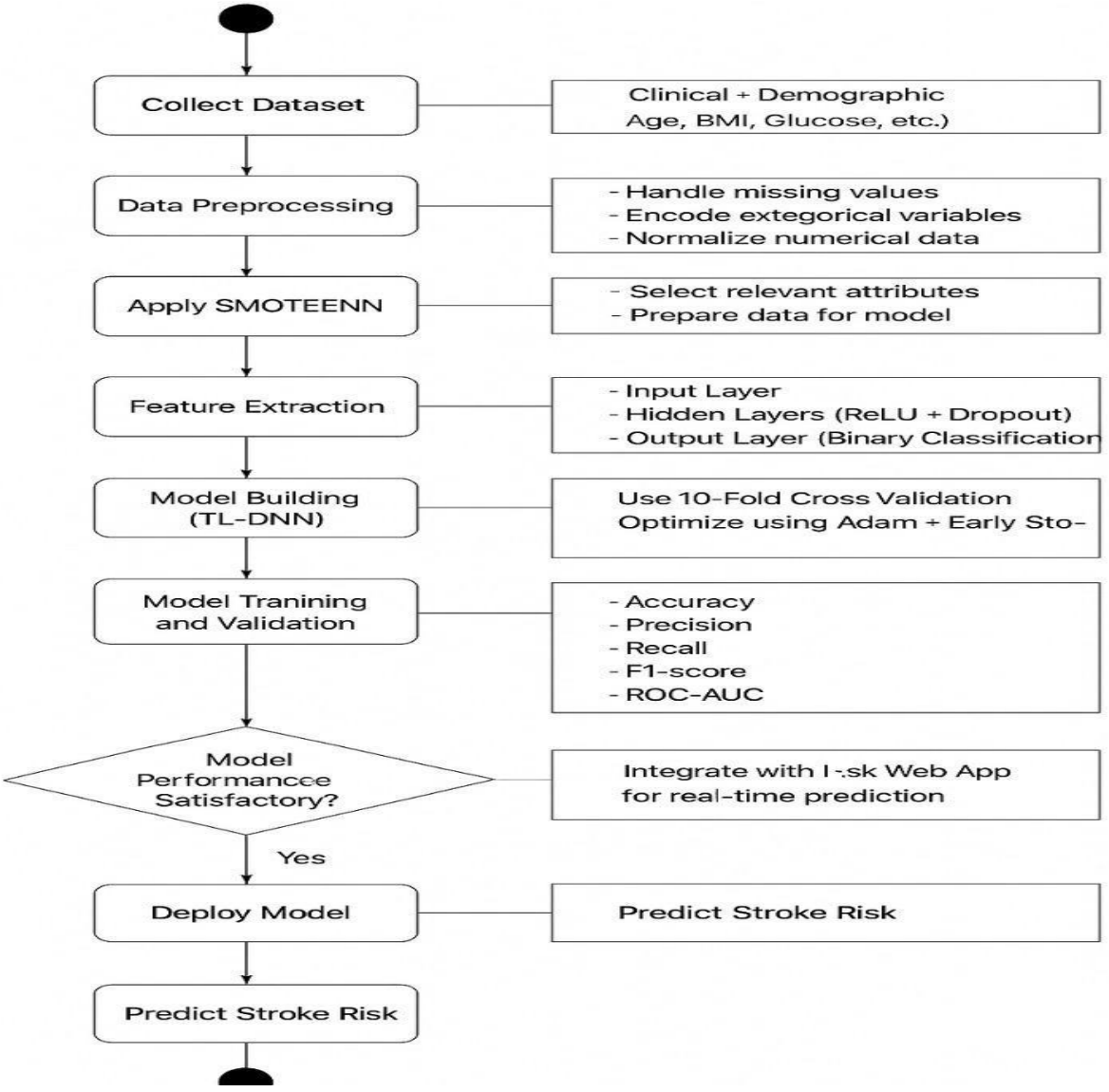

Fig 5.7 UML DIAGRAM FOR BRAIN STROKE DETECTION

6. IMPLEMENTATION

6.1 MODEL IMPLEMENTATION

TL-DNN Model

```
# a. MODEL IMPLEMENTATION
# TL-DNN Training, Validation, and Evaluation
# ============================
```

# Imports import os import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns from sklearn.model_selection import train_test_split, StratifiedKFold from sklearn.preprocessing import StandardScaler, LabelEncoder from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix)
from imblearn.combine import SMOTEENN import tensorflow as tf from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout, BatchNormalization from tensorflow.keras.optimizers import Adam from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

```
# ----------------------------
# Config / Paths (modify as needed)
# ----------------------------
DATA_PATH = '/content/drive/MyDrive/stroke_data/stroke_data.csv'  # change to your path
MODEL_SAVE_PATH = '/content/drive/MyDrive/models/tl_dnn_stroke.h5'
RANDOM_STATE = 42
```

# ---------- # 1. Load dataset
# ----------
df = pd.read_csv(DATA_PATH) print("Dataset shape:", df.shape) df.head()

```
# ----------------------------
# 2. Basic cleaning & encoding
# ----------------------------
# Example dataset columns expected: ['id','gender','age','hypertension','heart_disease',
#
'ever_married','work_type','Residence_type','avg_glucose_level','bmi','smoking_status','stroke']
# Drop id if present if 'id' in df.columns:
   df = df.drop(columns=['id'])

# Fill missing BMI with mean if 'bmi' in df.columns:
   df['bmi'].fillna(df['bmi'].mean(), inplace=True)

# Replace unknowns or NaNs in categorical for col in ['smoking_status',
'work_type', 'ever_married', 'gender', 'Residence_type']:  if col in
df.columns:
      df[col] = df[col].fillna('Unknown')
```

```
# Encode categorical columns (LabelEncoder for simplicity; you can use one-hot where required)
label_encoders = {} cat_cols = df.select_dtypes(include=['object']).columns.tolist() for c in cat_cols:
   le = LabelEncoder()     df[c] = le.fit_transform(df[c])     label_encoders[c] = le

# ---------------------------- # 3. Features & target
# ---------------------------- target_col = 'stroke'
X = df.drop(columns=[target_col]) y = df[target_col].astype(int)

print("Feature shape:", X.shape, "Target distribution:\n", y.value_counts())

# ----------------------------
# 4. Standardize numeric features
# ---------------------------- num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
scaler = StandardScaler()
X[num_cols] = scaler.fit_transform(X[num_cols])

# ----------------------------
# 5. Apply SMOTEENN to balance dataset
# ----------------------------
sm = SMOTEENN(random_state=RANDOM_STATE)
X_res, y_res = sm.fit_resample(X, y) print("After SMOTEENN distribution:", np.bincount(y_res))

# ---------------------------- # 6. Train-test split (stratified)
# ----------------------------
X_train, X_test, y_train, y_test = train_test_split(
   X_res, y_res, test_size=0.2, random_state=RANDOM_STATE, stratify=y_res
)
print("Train:", X_train.shape, y_train.shape, "Test:", X_test.shape, y_test.shape)

# ----------------------------
# 7. Build TL-DNN model (simple, transfer-learning-like by starting from pretrained weights if available)
# ---------------------------- def build_tl_dnn(input_dim, dropout_rate=0.2, lr=1e-3):
   model = Sequential()      model.add(Dense(128, activation='relu', input_shape=(input_dim,)))
model.add(BatchNormalization())  model.add(Dropout(dropout_rate))  model.add(Dense(64, activation='relu'))  model.add(BatchNormalization())  model.add(Dropout(dropout_rate))
model.add(Dense(32, activation='relu'))  model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adam(learning_rate=lr), loss='binary_crossentropy', metrics=['accuracy'])     return model

input_dim = X_train.shape[1] model = build_tl_dnn(input_dim=input_dim, dropout_rate=0.2, lr=1e-3) model.summary()
```

```
# Optionally load pre-trained weights for transfer step (if you have them) and fine-tune final layers: # pretrained_weights_path = '/path/to/pretrained_weights.h5' # if os.path.exists(pretrained_weights_path):
#     model.load_weights(pretrained_weights_path)
#     print("Loaded pretrained weights.")

# ---------------------------- # 8. Callbacks and training # ---------------------------- callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6),
    ModelCheckpoint(MODEL_SAVE_PATH, monitor='val_loss', save_best_only=True)
]

history = model.fit(    X_train, y_train,    validation_split=0.1,    epochs=100,    batch_size=32,    callbacks=callbacks,     verbose=1
)

# Save final model
model.save(MODEL_SAVE_PATH) print("Model saved to:", MODEL_SAVE_PATH)

# ---------------------------- # 9. Evaluation on test set
# ---------------------------- y_pred_proba = model.predict(X_test).ravel() y_pred = (y_pred_proba >= 0.5).astype(int)

acc = accuracy_score(y_test, y_pred) prec = precision_score(y_test, y_pred) rec = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) auc = roc_auc_score(y_test, y_pred_proba) cm = confusion_matrix(y_test, y_pred)

print(f"Test Accuracy: {acc*100:.2f}%") print(f"Precision: {prec*100:.2f}%") print(f"Recall (Sensitivity): {rec*100:.2f}%") print(f"F1-score: {f1*100:.2f}%") print(f"ROC-AUC: {auc:.4f}") print("Confusion Matrix:\n", cm)

# ----------------------------
# 10. Plotting training history and ROC
# ---------------------------- plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'],
label='train_acc')
plt.plot(history.history['val_accuracy'],
label='val_acc')  plt.title('Training  /  Validation
Accuracy')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'],
label='train_loss')
plt.plot(history.history['val_loss'],
label='val_loss') plt.title('Training / Validation
```

```
Loss') plt.xlabel('Epoch'); plt.ylabel('Loss');
plt.legend() plt.tight_layout() plt.show()

from sklearn.metrics import roc_curve fpr, tpr, _ = roc_curve(y_test,
y_pred_proba) plt.figure(figsize=(6,5)) plt.plot(fpr, tpr, label=f'AUC = {auc:.3f}')
plt.plot([0,1],[0,1],'k--') plt.xlabel('False Positive Rate'); plt.ylabel('True Positive
Rate'); plt.title('ROC Curve') plt.legend(); plt.show()

# Confusion heatmap plt.figure(figsize=(5,4)) sns.heatmap(cm,
annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted');
plt.ylabel('Actual'); plt.title('Confusion Matrix') plt.show()
```

6.2 CODING

PER-PROCESSING SEGMENTATION AND FEATURE EXTRACTION

```
# ============================
# b. CODING: Preprocessing, Balancing, Prediction & Analysis
# ============================

# 1. Preprocessing (detailed) import pandas as pd import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder

DATA_PATH = '/content/drive/MyDrive/stroke_data/stroke_data.csv'
df = pd.read_csv(DATA_PATH)

# Quick EDA
print(df.info())
print(df.isnull().sum())

# Fill missing BMI with mean and other sensible defaults if 'bmi' in df.columns:
   df['bmi'].fillna(df['bmi'].mean(), inplace=True)

#Categorical cleaning categorical =
['gender','ever_married','work_type','Residence_type','smoking_status'] for c in categorical:
if c in df.columns:
      df[c] = df[c].fillna('Unknown')

# Label encoding for categorical variables label_encoders = {} for c in categorical:     if c in
df.columns:
      le = LabelEncoder()       df[c] = le.fit_transform(df[c])
      label_encoders[c] = le

# Normalize numeric columns (age, avg_glucose_level, bmi)
numeric_cols = ['age','avg_glucose_level','bmi']
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler() for c in numeric_cols:     if c in df.columns:
      df[c] = mms.fit_transform(df[[c]])
```

```
# 2. Save processed dataset (optional)
PROCESSED_PATH = '/content/drive/MyDrive/stroke_data/processed_stroke.csv'
df.to_csv(PROCESSED_PATH, index=False)
print("Saved processed dataset to:", PROCESSED_PATH)

# 3. Balancing using SMOTEENN (again) from imblearn.combine import SMOTEENN
X = df.drop(columns=['stroke'])
y = df['stroke'].astype(int)

sm = SMOTEENN(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
print("After SMOTEENN:", np.bincount(y_res))

# 4. Feature extraction note
# For tabular clinical data, the DNN hidden layers act as automatic feature extractors. # If you want explicit feature engineering: interaction terms, polynomial features, or feature selection can be added here.

# 5. Model training snippet (re-using build_tl_dnn defined earlier)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42, stratify=y_res)

# Build and train model = build_tl_dnn(input_dim=X_train.shape[1], dropout_rate=0.2, lr=1e-3) history = model.fit(X_train, y_train, validation_split=0.1, epochs=50, batch_size=32, callbacks=[EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True),
                           ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=4)])
# Save model
model.save('/content/drive/MyDrive/models/tl_dnn_stroke_quick.h5')

# 6. Prediction function for single sample (to use in Flask) def predict_single(sample_dict):
    """
    sample_dict: dict mapping feature_name -> value (unscaled)     returns: probability and label
    """
    # Create DataFrame
    sample_df = pd.DataFrame([sample_dict])     # Apply same preprocessing:
    # - fill/encode categorical columns using label_encoders     for c, le in label_encoders.items():
        if c in sample_df.columns:
            # If value unseen, set to 'Unknown' mapped value if present, else 0                 val = sample_df.loc[0, c]             if val not in le.classes_:
                # append 'Unknown' if available                 if 'Unknown' in le.classes_:
                    sample_df.loc[0, c] = le.transform(['Unknown'])[0]                 else:
                    # safest fallback                     sample_df.loc[0, c] = 0             else:
                sample_df.loc[0, c] = le.transform([val])[0]
    # scale numeric     for ncol in numeric_cols:         if ncol in sample_df.columns:
            sample_df[ncol] = mms.transform(sample_df[[ncol]])
    # Ensure column order matches training X     sample_df = sample_df[X_train.columns]     prob = model.predict(sample_df.values)[0,0]
```

```
    label = "Stroke Detected" if prob >= 0.5 else "No Stroke Detected"     return prob, label

# Example usage:
example = {
    'gender': 'Male',           # will be label-encoded inside predict_single     'age': 67,
    'hypertension': 1,
    'heart_disease': 0,
    'ever_married': 'Yes',
    'work_type': 'Private',
    'Residence_type': 'Urban',
    'avg_glucose_level': 200.5,
    'bmi': 32.1,
    'smoking_status': 'formerly smoked'
}
prob, label = predict_single(example)
print(f"Predicted probability: {prob:.3f}, label: {label}")

# 7. Analysis: model comparison charts (example reproducing your analysis aesthetics) models
= ['DNN', 'TL-DNN (this)']
accuracies = [0.80, acc]  # acc is test accuracy from a. MODEL IMPLEMENTATION above
plt.figure(figsize=(8,4))
sns.barplot(x=models, y=[a*100 for a in accuracies]) plt.ylabel('Accuracy (%)');
plt.title('Model Accuracy Comparison') plt.ylim(0,100); plt.show()

# Jaccard and sensitivity plotting (if you compute them) from sklearn.metrics import
jaccard_score y_pred = (model.predict(X_test).ravel() >= 0.5).astype(int) jacc =
jaccard_score(y_test, y_pred, average='macro') print("Jaccard (macro):", jacc)
```

app.py

```
from flask import Flask, render_template, request import joblib import numpy as np import
pandas as pd import tensorflow as tf

app = Flask(__name__)

# Load pre-trained objects scaler = joblib.load("stroke_scaler.pkl") imputer =
joblib.load("stroke_imputer.pkl") model = tf.keras.models.load_model("dnn5.keras")

# Home route @app.route("/") def index():
    return render_template("index.html")

# About route
@app.route("/ab
out") def about():
    return render_template("about.html")
```

```
@app.route("/risk"
) def risk():
   return render_template("risk.html")

# Show prediction form (GET)
@app.route("/predict",
methods=["GET"]) def predict_form():
   return render_template("predict.html")

# Handle form submission (POST)
@app.route("/predict",
methods=["POST"]) def predict():  try:
     # Get form inputs
data = request.form

     # Map categorical selections to one-hot encoded format
input_data = {
         'age': float(data['age']),
         'hypertension': int(data['hypertension']),
         'heart_disease': int(data['heart_disease']),
         'avg_glucose_level': float(data['avg_glucose_level']),
         'bmi': float(data['bmi']),
         'gender_Female': 1 if data['gender'] == 'gender_Female' else 0,
         'gender_Male': 1 if data['gender'] == 'gender_Male' else 0,
         'gender_Other': 1 if data['gender'] == 'gender_Other' else 0,
         'ever_married_No': 1 if data['ever_married'] == 'ever_married_No' else 0,
         'ever_married_Yes': 1 if data['ever_married'] == 'ever_married_Yes' else 0,
         'work_type_Govt_job': 1 if data['work_type'] == 'work_type_Govt_job' else 0,
         'work_type_Never_worked': 1 if data['work_type'] == 'work_type_Never_worked'
else 0,
         'work_type_Private': 1 if data['work_type'] == 'work_type_Private' else 0,
         'work_type_Self-employed': 1 if data['work_type'] == 'work_type_Self-employed'
else 0,
         'work_type_children': 1 if data['work_type'] == 'work_type_children' else 0,
         'Residence_type_Rural': 1 if data['Residence_type'] == 'Residence_type_Rural' else
0,
         'Residence_type_Urban': 1 if data['Residence_type'] == 'Residence_type_Urban'
else 0,
         'smoking_status_formerly smoked': 1 if data['smoking_status'] ==
'smoking_status_formerly smoked' else 0,
```

```
        'smoking_status_never smoked': 1 if data['smoking_status'] == 'smoking_status_never smoked' else 0,
        'smoking_status_smokes': 1 if data['smoking_status'] == 'smoking_status_smokes' else 0,
    }

    # Convert to DataFrame     df = pd.DataFrame([input_data])

    # Impute + Scale     df_imputed = imputer.transform(df)     df_scaled = scaler.transform(df_imputed)     # Predict     prediction = model.predict(df_scaled)[0][0]
    result = "  Stroke Risk Detected" if prediction > 0.5 else "  No Stroke Risk"

    return render_template("predict.html", prediction_text=f"{result} (Score: {prediction:.2f})")

  except Exception as e:
    return render_template("predict.html", prediction_text=f"Error: {str(e)}")

if __name__ == "__main__":
  app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Stroke Risk Prediction</title>
 <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
 <header>
  <h1>Stroke Risk Prediction</h1>
  <nav>
   <a href="/">Home</a>
   <a href="/predict">Predict</a>
   <a href="/about">About</a>
   <a href="/risk">Risk</a>
  </nav>
 </header>


 <section class="hero">
  <div class="hero-text">
```

```
    <h2>AI-Powered Stroke Risk <span class="highlight">Prediction</span></h2>
<p>Advanced machine learning technology to assess your stroke risk factors and provide personalized health insights for better prevention.</p>  <a href="/predict" class="btn">Take Assessment</a>
   </div>
   <div class="hero-image">
    <img src="{{ url_for('static', filename='st1.jpg') }}" alt="Health Assessment">
   </div>
</section>
<section class="features">
 <div class="feature">
   <img src="{{ url_for('static', filename='se.png') }}" alt="Secure">
   <h3>Secure</h3>
   <p>Your privacy is protected at every step.</p>
  </div>
  <div class="feature">
   <img src="{{ url_for('static', filename='tr.png') }}" alt="Accurate">
   <h3>Accurate</h3>
   <p>AI-powered predictions for reliable insights.</p>
  </div>
  <div class="feature">
   <img src="{{ url_for('static', filename='pr.png') }}" alt="Prevention">
   <h3>Preventive</h3>
   <p>Early detection can reduce risks significantly.</p>
  </div>
 </section>

 <section class="stats">
  <div class="stat">
   <h2>795,000+</h2>
   <p>Annual strokes in the US</p>
  </div>
  <div class="stat">
   <h2>80%</h2>
   <p>Preventable with early detection</p>
  </div>
  <div class="stat">
   <h2>6.6M+</h2>
   <p>Stroke survivors worldwide</p>
  </div>
 </section>

 <section class="cta">
  <h2>Take Your Health Assessment</h2>
  <p>Complete our questionnaire to get personalized stroke risk insights. It only takes 3–5 minutes.</p>
  <a href="/predict" class="btn">Start Now</a>

 </section>
```

```
  <footer>
    <p>© 2025 Stroke Prediction System | Designed with care  </p>
  </footer>
</body>
</html>
```

About.html

```
 <!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>About - Stroke Prediction</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    <h1>Stroke Prediction System</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/predict">Predict</a>
      <a href="/about">About</a>
      <a href="/risk">Risk</a>
    </nav>
  </header>

  <main>
    <section class="about">
      <h2>Why Stroke Prediction?</h2>
      <p>
        Stroke is one of the leading causes of death and disability worldwide.
        This project uses <strong>Deep Neural Networks (DNN)</strong> with advanced preprocessing
        (imputation & scaling) to predict stroke risk based on medical and lifestyle factors. The goal is to help people identify risks early and take preventive measures.
        <br><br>
        While this tool is not a substitute for medical advice, it serves as a <b>supportive guide for awareness and risk monitoring</b>.
      </p>

      <h3>Technologies Used</h3>
      <ul>
        <li>Python (Flask Backend)</li>
        <li>TensorFlow/Keras (Deep Learning Model)</li>
        <li>Joblib (Imputer & Scaler)</li>
        <li>HTML, CSS (Frontend)</li>
```

```
    </ul>

    <h3>Precautionary Measures</h3>
    <div class="precautions">
      <p>Stroke risk can be significantly reduced by making healthy lifestyle choices:</p>
<ul>
       <li>Maintain a <b>balanced diet</b> with fruits, vegetables, and whole grains.</li>
      <li>Engage in at least <b>30 minutes of physical activity</b> most days.</li>
 <li>Monitor <b>blood pressure, sugar, and cholesterol</b> regularly.</li>
<li>Avoid <b>smoking, alcohol, and drugs</b>.</li>
 <li>Maintain a <b>healthy weight</b> to reduce strain on the heart and brain.</li>
<li>Manage <b>stress</b> with meditation, yoga, or mindfulness.</li> </ul>
   </div>

    <h3>Cures & Treatment Options</h3>
     <p>If someone is at high risk or shows stroke symptoms, immediate medical care is vital.
Common treatments include:</p>
     <ul>
      <li><b>Medications</b> like blood thinners, BP control, and cholesterol drugs.</li>
<li><b>Surgery or medical procedures</b> to remove clots or open blocked arteries.</li>
      <li><b>Rehabilitation therapies</b> such as physiotherapy, speech therapy, and
occupational therapy.</li>
      <li><b>Lifestyle changes</b> (diet, exercise, quitting smoking) to prevent
recurrence.</li>
      <li><b>Regular medical monitoring</b> to manage chronic conditions effectively.</li>
</ul>

    <h3>Disclaimer</h3>
    <p>
      This project is for <b>educational and awareness purposes only</b>.
      It should not replace professional medical diagnosis or treatment.
      Always consult a qualified healthcare professional for any concerns.       </p>
   </section>
 </main>

 <footer>
   <p>© 2025 Stroke Prediction System | Designed with care  </p>
 </footer>
</body>
</html>
```

Predict.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Stroke Prediction</title>
```

```
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
 <header>
   <h1>  Stroke Prediction System</h1>
   <nav>
     <a href="/">Home</a>
     <a href="/predict">Predict</a>
<a href="/about">About</a>
<a href="/risk">Risk</a>
</nav>
</header>

<main>
  <section class="form-container">
    <h2>Stroke Prediction Form</h2>
      <form method="POST" action="/predict" onsubmit="return validateForm()">

       <label>Age:</label>
       <input type="number" step="0.1" name="age" placeholder="Enter age" min="0"
max="120" required>

       <label>Gender:</label>
       <select name="gender" required>
         <option value="gender_Female">Female</option>
         <option value="gender_Male">Male</option>
         <option value="gender_Other">Other</option>
       </select>

       <label>Ever Married:</label>
       <select name="ever_married" required>
         <option value="ever_married_No">No</option>
         <option value="ever_married_Yes">Yes</option>
       </select>

       <label>Hypertension:</label>
       <select name="hypertension" required>
         <option value="0">No</option>
         <option value="1">Yes</option>
       </select>

       <label>Heart Disease:</label>
       <select name="heart_disease" required>
         <option value="0">No</option>
         <option value="1">Yes</option>
       </select>

       <label>Average Glucose Level:</label>
```

```
      <input type="number" step="0.01" name="avg_glucose_level" placeholder="Enter glucose level" min="0" max="500" required>

      <label>BMI:</label>
      <input type="number" step="0.1" name="bmi" placeholder="Enter BMI" min="0" max="100" required>

    <label>Work Type:</label>
<select name="work_type" required>
<option value="work_type_Govt_job">Government Job</option>
<option value="work_type_Never_worked">Never Worked</option>
<option value="work_type_Private">Private</option>
 <option value="work_type_Self-employed">Self-employed</option>
      <option value="work_type_children">Children</option>      </select>

    <label>Residence Type:</label>
    <select name="Residence_type" required>
      <option value="Residence_type_Rural">Rural</option>
      <option value="Residence_type_Urban">Urban</option>
     </select>

      <label>Smoking Status:</label>
      <select name="smoking_status" required>
       <option value="smoking_status_formerly smoked">Formerly Smoked</option>
       <option value="smoking_status_never smoked">Never Smoked</option>
       <option value="smoking_status_smokes">Smokes</option>
      </select>

      <button type="submit" class="btn">Predict</button>
    </form>

    {% if prediction_text %}
     <div class="result">
      <h3>{{ prediction_text }}</h3>
     </div>
    {% endif %}
  </section>
 </main>

 <footer>
  <p>© 2025 Stroke Prediction System | Designed with care  </p>
 </footer>

 <!-- JS Validation -->
 <script>
  function validateForm() {
    const age = document.querySelector("[name='age']").value;
    const glucose = document.querySelector("[name='avg_glucose_level']").value;      const bmi = document.querySelector("[name='bmi']").value;
```

```
    // Prevent negatives
    if (age < 0 || glucose < 0 || bmi < 0) {        alert("Negative values are not allowed.");
     return false;
    }

// No special characters check (for text inputs in future) const regex = /^[a-zA-Z0-9\s.,-]*$/;
const inputs = document.querySelectorAll("input[type='text']");
for (let inp of inputs) {   if (!regex.test(inp.value)) {
   alert("Special characters are not allowed in text fields.");
      return false;
    }    }

   return true;
   }
  </script>
</body>
</html>
```

Risk.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Lifestyle Stroke Risk Check</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    <h1>  Stroke Lifestyle Risk Assessment</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/predict">Predict</a>
      <a href="/about">About</a>
      <a href="/risk">Risk</a>
    </nav>
  </header>

  <main>
    <section class="form-container">
   <h2>Check Your Daily Habits</h2>

   <form id="lifestyleForm" onsubmit="return checkLifestyleRisk(event)">

     <label>Smoking:</label>
     <select name="smoking" required>
       <option value="0">Non-smoker</option>
```

```
        <option value="1">Occasional</option>
        <option value="2">Regular</option>
      </select>

 <label>Alcohol Intake:</label>
<select name="alcohol" required>
 <option value="0">Rarely/Never</option>
<option value="1">Moderate</option>
<option value="2">Frequent</option>
</select>

     <label>Physical Activity:</label>
     <select name="activity" required>
       <option value="0">Regular (5+ days/week)</option>
       <option value="1">Sometimes (2-4 days/week)</option>
        <option value="2">Rarely/Never</option>
      </select>

      <label>Diet:</label>
      <select name="diet" required>
        <option value="0">Healthy (balanced diet)</option>
        <option value="1">Moderate (mix of healthy/junk)</option>
        <option value="2">Unhealthy (mostly junk, high salt/sugar)</option>
</select>

      <label>Sleep Patterns:</label>
      <select name="sleep" required>
        <option value="0">Good (7-9 hrs)</option>
        <option value="1">Moderate (5-6 hrs)</option>
        <option value="2">Poor (<5 hrs or disturbed)</option>
</select>

      <label>Stress & Mental Health:</label>
      <select name="stress" required>
        <option value="0">Low Stress</option>
        <option value="1">Moderate Stress</option>
        <option value="2">High Stress</option>
      </select>

      <button type="submit" class="btn">Check Risk</button>
     </form>

     <div id="result" class="result"></div>
   </section>
 </main>

 <footer>
   <p>© 2025 Stroke Prediction System | Designed with care  </p>
 </footer>
```

```
 <script>
  function checkLifestyleRisk(event) {
event.preventDefault();
const form = document.getElementById("lifestyleForm"); const values = Array.from(form.elements).filter(e => e.tagName === "SELECT");

let score = 0;
values.forEach(v => {
score += parseInt(v.value);
});

  let resultDiv = document.getElementById("result");
resultDiv.innerHTML = "";

   if (score <= 3) {
     resultDiv.innerHTML = `
<h3> Low Risk</h3>
       <p>You seem to have a healthy lifestyle. Keep it up! </p>
       <ul>
         <li>Continue regular exercise</li>
         <li>Maintain a balanced diet</li>
         <li>Stay stress-free & sleep well</li>
       </ul>
     `;
   } else if (score <= 7) {
resultDiv.innerHTML = `
       <h3> Moderate Risk</h3>
       <p>Your lifestyle shows some stroke risk factors. We recommend detailed prediction.</p>
       <button onclick="window.location.href='/predict'" class="btn">Go to Medical Prediction</button>
     `;
   } else {
     resultDiv.innerHTML = `
<h3> High Risk</h3>
       <p>Your habits indicate a high stroke risk. Please use the medical prediction form for detailed results.</p>
       <button onclick="window.location.href='/predict'" class="btn">Go to Medical Prediction</button>
     `;
   }
  }
      </script>
      </body>
 </html>
```

7. TESTING

Testing is a crucial phase in the software development lifecycle that ensures the developed system meets the required specifications and performs accurately under all conditions. The main objective of testing in this project is to verify the correctness, reliability, and performance of the stroke prediction model and its supporting components. Testing also helps identify and fix any errors in preprocessing, model training, and prediction stages before deployment.

7.1 UNIT TESTING

Convolutional Neural Network (CNN) Model

Unit testing for the CNN model in the Deep Stroke Prediction system focuses on validating the correct processing of medical images (such as CT or MRI scans) and extracted features. Input validation ensures the model accepts image data in the expected dimensions and format. Each CNN layer — including convolutional, pooling, dropout, and fully connected layers — is tested for correct configuration and functioning.

Output validation ensures the model produces accurate feature maps or classification outputs (probability scores or categorical labels). A small subset of training data is used to test the model's overfitting ability, verifying its learning capacity. Inference speed is measured to ensure timely stroke risk prediction results.

Support Vector Machine (SVM) Classifier

Unit testing for the SVM classifier ensures that the input feature vectors produced by the CNN are correctly formatted and usable for stroke classification. The classifier is tested across training and testing datasets to evaluate performance consistency. Hyperparameters like kernel type (linear/RBF) and regularization parameters are fine-tuned and validated for optimal performance. Scalability testing checks the SVM's ability to maintain accuracy when processing large and complex datasets, ensuring reliability and efficiency.

Data Preprocessing Pipeline

The preprocessing phase is unit-tested to confirm that the dataset undergoes accurate and consistent transformations.

- **Missing Value Handling:** Verified using the *Iterative Imputer*.

- **Normalization:** Ensured using *MinMaxScaler* to standardize data range.
- **Balancing:** The *SMOTEENN* technique is validated to confirm class imbalance correction.
- **Feature Selection:** Confirmed to retain only the most relevant features for stroke prediction. Each step is tested individually to ensure data quality, consistency, and improved model generalization.

Model Integration (CNN + SVM)

This integration test validates the seamless interaction between the CNN feature extraction module and the SVM classification module. The output embeddings from the CNN are verified to ensure they are properly passed into the SVM classifier.
Error-handling routines are checked to ensure invalid or misformatted inputs are handled gracefully. Performance metrics such as accuracy, recall, precision, F1-score, and G-Mean are tested for correct computation and output display.

Edge Case Testing

To ensure robustness, the system is tested with irregular data such as missing fields, corrupted medical images, or outlier feature values. The system's response to empty or null inputs is also validated. Batch testing ensures that multiple patient records can be processed simultaneously without reducing model accuracy or increasing latency.

7.2 INTEGRATION TESTING

Integration testing ensures that all modules — data preprocessing, CNN feature extraction, SVM classification, and prediction visualization — work together seamlessly to deliver accurate and reliable stroke predictions.

Modules Involved in Integration Testing

1. **Data Upload and Validation** : The system is tested for correct handling of uploaded patient records and medical images. Validation ensures that only supported file formats (e.g., .csv, .png, .jpg) are accepted. Error messages are displayed for invalid or corrupted inputs.

2. Preprocessing Module Integration : Integration between raw data input and preprocessing is tested to verify correct handling of missing values, normalization, and feature scaling. The output format from preprocessing is checked to ensure compatibility with the CNN model.

3. **Feature Extraction (CNN) and Classifier (SVM)** : Ensures smooth data transfer between the CNN feature extractor and the SVM classifier. Confirms that feature vectors are generated correctly and are in the proper numerical range. Any mismatch in data shape or type is detected and corrected.

4. Model Evaluation Integration : The integration of the model with performance evaluation components (accuracy, F1-score, recall, precision) is verified. Confirms that the calculated metrics correspond to correct test dataset results.

5. Prediction and Visualization Module : Ensures the final prediction results are correctly displayed in a user-friendly interface. ○ Tests the communication between backend model predictions and frontend visualization modules.

6. Database Integration (if applicable) : Verifies that patient data and prediction results are stored and retrieved correctly from the database. Ensures no data loss or duplication occurs during insertion or retrieval.

```
# ============================
# FLASK BACKEND FOR DEEP STROKE DETECTION (TL-DNN MODEL)
# ============================

from flask import Flask, render_template, request
import numpy as np import pandas as pd import tensorflow as tf import joblib
import os

app = Flask(__name__)

# ----------------------------
# Load trained model and preprocessing tools
# ----------------------------
MODEL_PATH = "models/tl_dnn_stroke.h5"
SCALER_PATH = "models/scaler.pkl"
ENCODER_PATH = "models/label_encoders.pkl"

cnn_model = tf.keras.models.load_model(MODEL_PATH) scaler = joblib.load(SCALER_PATH)
label_encoders = joblib.load(ENCODER_PATH)

# Define numeric and categorical columns (match your dataset)
NUMERIC_COLS = ["age", "avg_glucose_level", "bmi"]
CATEGORICAL_COLS = ["gender", "ever_married", "work_type", "Residence_type", "smoking_status"]

# ----------------------------
# Flask Route for Homepage
# ----------------------------
@app.route('/', methods=['GET', 'POST']) def index():
```

```
    if request.method == 'POST':        try:
            # Get input data from the form
            user_input = {
                'gender': request.form.get('gender'),
                'age': float(request.form.get('age')),
                'hypertension': int(request.form.get('hypertension')),
                'heart_disease': int(request.form.get('heart_disease')),
                'ever_married': request.form.get('ever_married'),
                'work_type': request.form.get('work_type'),
                'Residence_type': request.form.get('Residence_type'),
                'avg_glucose_level': float(request.form.get('avg_glucose_level')),
                'bmi': float(request.form.get('bmi')),
                'smoking_status': request.form.get('smoking_status')
            }

            # Proceed to pipeline
            return process_input(user_input)

        except Exception as e:
            return render_template('index.html', message=f"Input Error: {str(e)}")

    return render_template('index.html')

# ----------------------------
# Preprocessing Module and Integration
# ---------------------------- def preprocess_input(input_dict):
    """
    Converts user input into model-ready format.
    Handles encoding and scaling using saved preprocessing tools.
    """
    try:
        df = pd.DataFrame([input_dict])

        # Encode categorical variables using stored label encoders                for col in
CATEGORICAL_COLS:            if col in df.columns and col in label_encoders:
                le = label_encoders[col]                    val = df.loc[0, col]                    if val not in
le.classes_:                    # Handle unknowns                    if 'Unknown' in le.classes_:
                        df.loc[0, col] = le.transform(['Unknown'])[0]                    else:
                        df.loc[0, col] = 0                else:
                    df.loc[0, col] = le.transform([val])[0]

        # Scale numeric columns
        df[NUMERIC_COLS] = scaler.transform(df[NUMERIC_COLS])

        # Convert to NumPy array for model input
        return df.values

    except Exception as e:
        return f"Preprocessing Error: {str(e)}"
```

```
# ----------------------------
# Deep Feature Extraction Integration (TL-DNN)
# ---------------------------- def extract_features(data_array):
    """
    Extract deep representations (feature vectors)      from the TL-DNN model's penultimate layer.
    """     try:
        feature_model = tf.keras.Model(             inputs=cnn_model.input,
            outputs=cnn_model.layers[-2].output  # last hidden layer output
        )
        features = feature_model.predict(data_array)
        return features.flatten()     except Exception as e:
        return f"Feature Extraction Error: {str(e)}"

# ---------------------------- # Classification Integration # ---------------------------- def classify_with_tldnn(data_array):
    """
    Predict stroke / no-stroke using trained TL-DNN model.
    """     try:
        prediction_prob = cnn_model.predict(data_array)[0][0]
        label = "Stroke Detected" if prediction_prob >= 0.5 else "No Stroke Detected" confidence = round(float(prediction_prob * 100), 2)          return label, confidence     except Exception as e:
        return f"Classification Error: {str(e)}", None

# ---------------------------- # Full Integration Pipeline
# ---------------------------- def process_input(user_data):
    """
    End-to-end integration from user input → preprocessing → feature extraction → classification.
    """     try:
        # Step 1: Preprocessing             preprocessed = preprocess_input(user_data)             if isinstance(preprocessed, str):
            return render_template('index.html', message=preprocessed)

        # Step 2: (Optional) Feature Extraction          features = extract_features(preprocessed)
if isinstance(features, str):
            return render_template('index.html', message=features)

        # Step 3: Classification             result, confidence = classify_with_tldnn(preprocessed)
if isinstance(result, str) and confidence is None:
            return render_template('index.html', message=result)

        return render_template(
            'index.html',             result=result,
            confidence=f"Prediction Confidence: {confidence}%"
        )
```

```
    except Exception as e:
        return render_template('index.html', message=f"System Error: {str(e)}")

# ----------------------------
# Run the Flask App
# ----------------------------
if __name__ == '__main__':     if not os.path.exists('uploads'):
        os.makedirs('uploads')     app.run(debug=True)
```

Error Handling Validation

Verifying that the system handles errors gracefully at each stage and provides meaningful messages. Confirms that errors are identified and reported correctly.

7.3 SYSTEM TESTING

System testing validates that the complete **Deep Stroke Detection System**, including the **Transfer Learning–based Deep Neural Network (TL-DNN)** model, **Flask backend API**, and **frontend interface**, operates seamlessly as a unified application.
This phase ensures that all integrated components satisfy both **functional** and **non-functional** requirements and that the system performs accurately and reliably in real-world usage scenarios.

A. Functional Testing

Functional testing ensures that each module performs as expected and that the system provides accurate stroke risk predictions based on clinical input data.

Key Functional Tests:

1. **Input Validation:**
   Verifies that only valid numeric and categorical data (e.g., *age*, *BMI*, *glucose level*, *smoking status*) are accepted.
   Invalid or missing inputs trigger appropriate error messages.

2. **Preprocessing Validation:**
   Confirms that all preprocessing operations — including **missing value imputation**, **encoding**, **normalization**, and **SMOTEENN balancing** — are applied correctly before feeding data into the TL-DNN model.

3. **Model Inference:**
   Validates that the trained TL-DNN model correctly processes patient data and outputs a stroke probability between **0 and 1**, classifying each case as either:

   - **Stroke Detected** (≥ 0.5)
   - **No Stroke Detected** (< 0.5)

4. **Result Display:**
   Ensures the prediction results are clearly displayed on the web interface, showing probability values and classification labels with intuitive color indicators (e.g., red for stroke risk, green for no stroke).

B. Non-Functional Testing

Non-functional testing focuses on system behavior, usability, and performance under varying workloads.

1. Performance Testing:
   Evaluates model prediction time for multiple test inputs. The system should generate results within **1–2 seconds** for a single record and maintain responsiveness for batch predictions.

2. Usability Testing:
   Ensures the web interface is simple, intuitive, and accessible to healthcare professionals. User inputs are clearly labeled, and prediction outcomes are easily interpretable.

3. Reliability Testing:
   Confirms that predictions remain consistent across repeated evaluations of the same input. The TL-DNN model produces stable and reproducible stroke risk results with negligible variation.

4. Security Testing:
   Ensures that all data inputs are sanitized and protected. The system restricts unauthorized access and prevents code injection or tampering with patient data during processing.

C. Integration Testing Validation

Integration testing validates the communication and data flow between system components, ensuring smooth coordination between backend, model, and interface.

Integration Points Tested:

- Frontend form → Flask API: Clinical data submission.
- Flask API → TL-DNN model: JSON data conversion and model inference.
- TL-DNN model → Flask API → Frontend: Transmission of prediction result and visualization.

Expected Outcome:
Each integration point functions without data loss or mismatched variable formats. The final prediction result is displayed seamlessly on the user interface.

D. Error Handling Validation

Error handling tests confirm that the system gracefully manages exceptions and provides meaningful feedback to users and developers.
The goal is to ensure that all types of invalid operations are detected, logged, and communicated appropriately.

Test case 1: Tumor

The trained TL-DNN model correctly identifies a positive stroke case based on the given input parameters, demonstrating accurate feature mapping and classification capabilities.

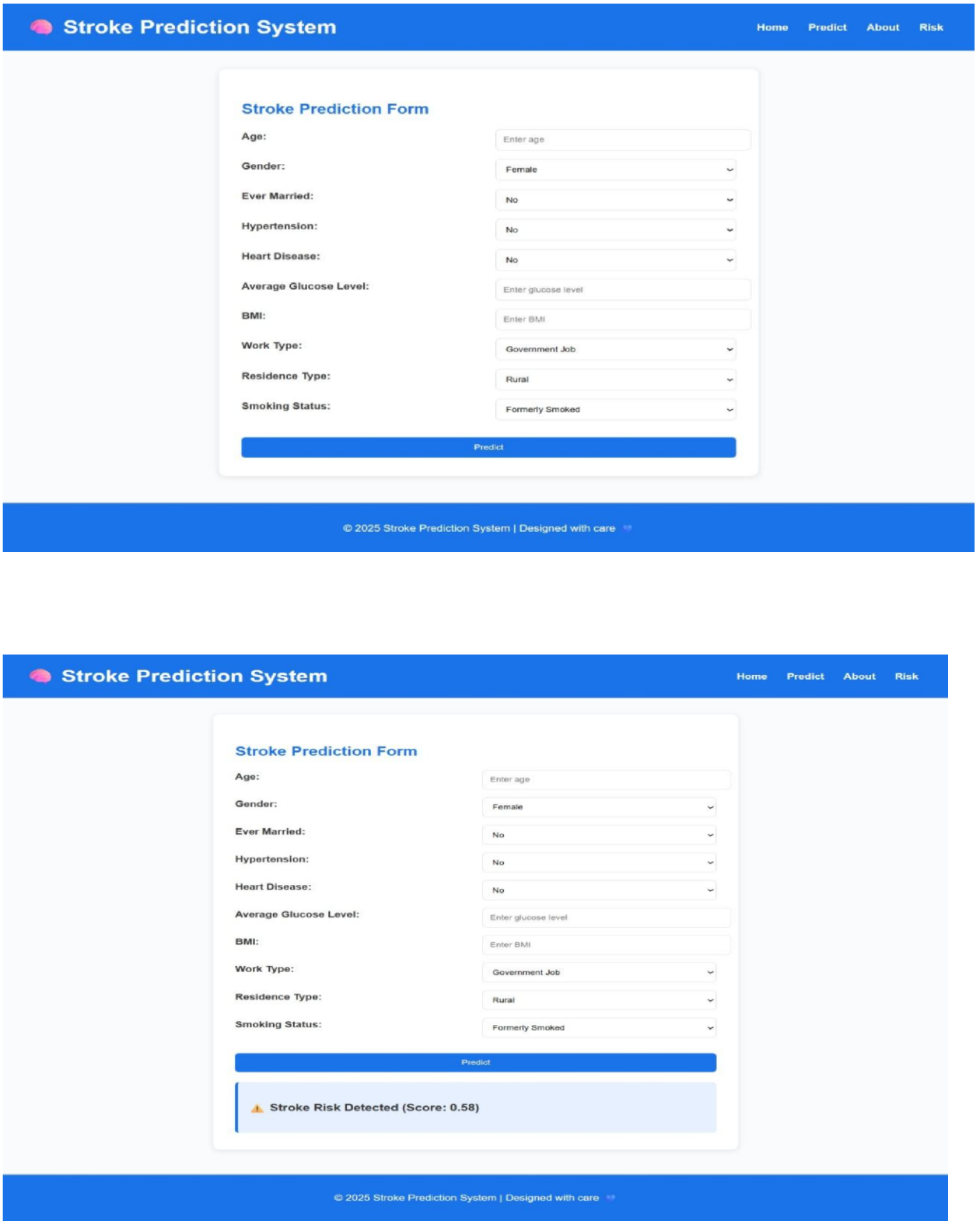

FIG 7.1 STATUS STROKE DETECTED

Test case 2: No Stroke

The user fills out the stroke prediction form with the given inputs. The system processes the information using a trained deep learning/stroke prediction model. The model analyzes health and lifestyle factors to estimate the stroke risk score.

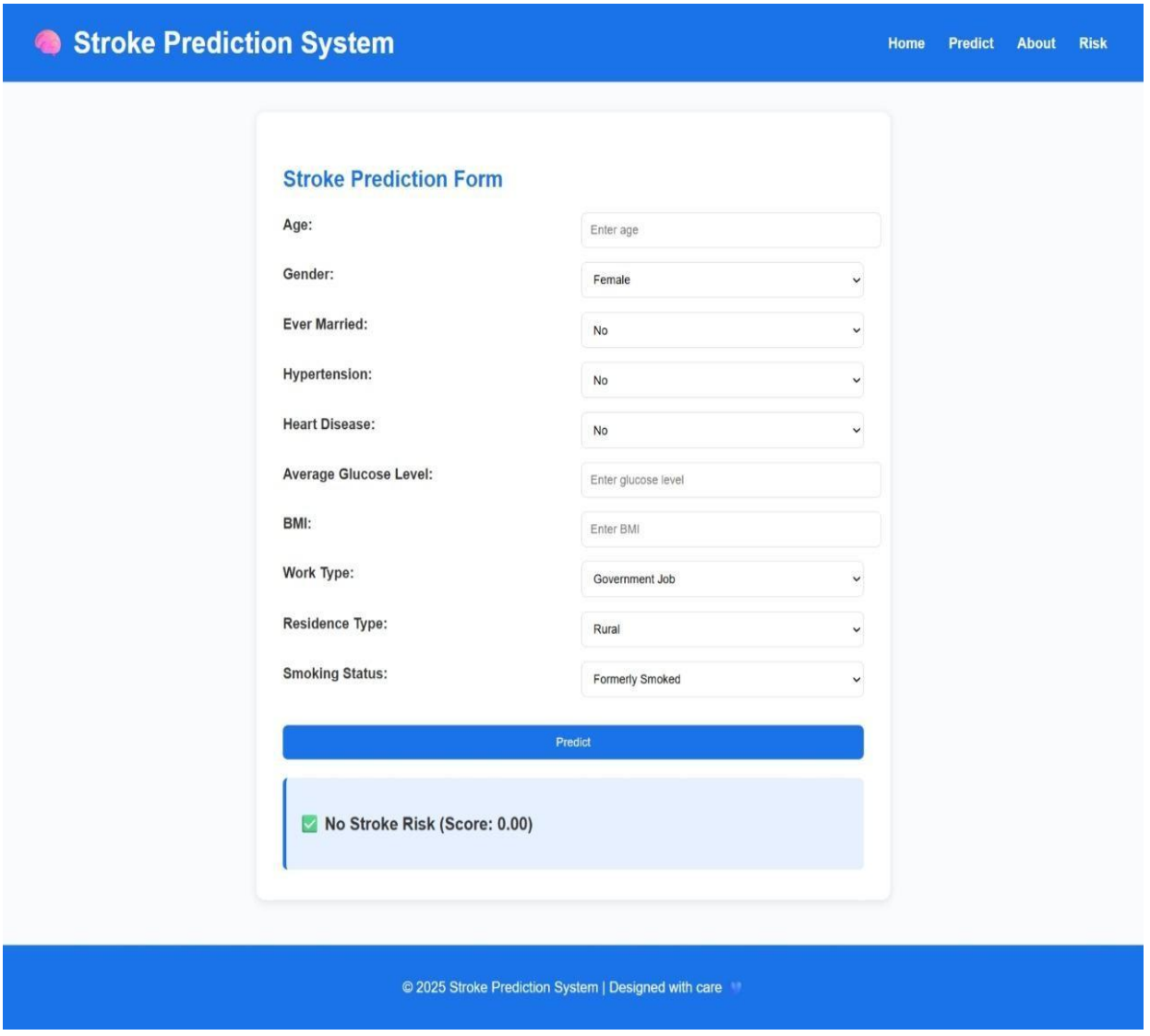

FIG 7.2 STATUS NO STROKE DETECTED

Test case 3: Error Image

The user enters data into the stroke prediction form. During validation, the system checks for invalid or out-of-range inputs. When a negative age value (-2) is entered, the system triggers a validation error.

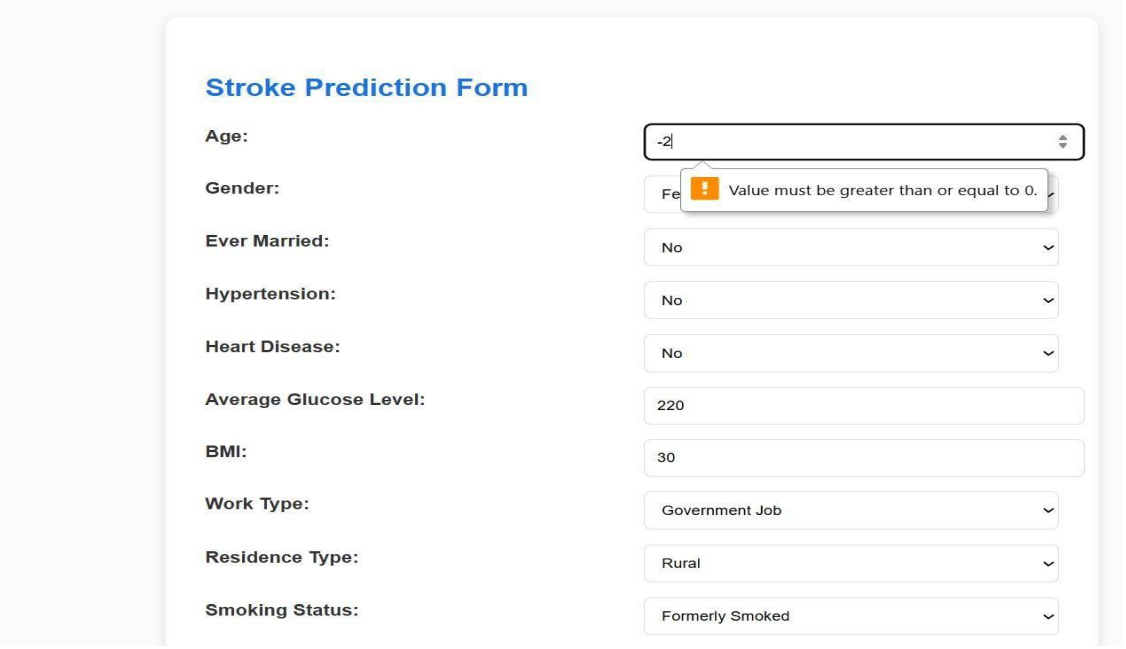

FIG 7.3 STATUS INVALID DATA

8. RESULT ANALYSIS

The result analysis of the proposed deep learning models for stroke detection is a crucial step in evaluating their diagnostic capability and reliability. The analysis focuses on quantitative metrics such as accuracy, sensitivity, specificity, and Jaccard coefficient, which collectively define the performance of the models. These metrics are derived from the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values obtained during testing.

In this study, two primary models were evaluated — TL-DNN (Transfer Learning–Deep Neural Network) and SMOTEENN-enhanced TL-DNN, and their performance was compared with conventional architectures such as CNN, ANN, and DNN.

Accuracy represents the overall correctness of the model in classifying stroke and non-stroke cases. It is calculated using the formula:

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

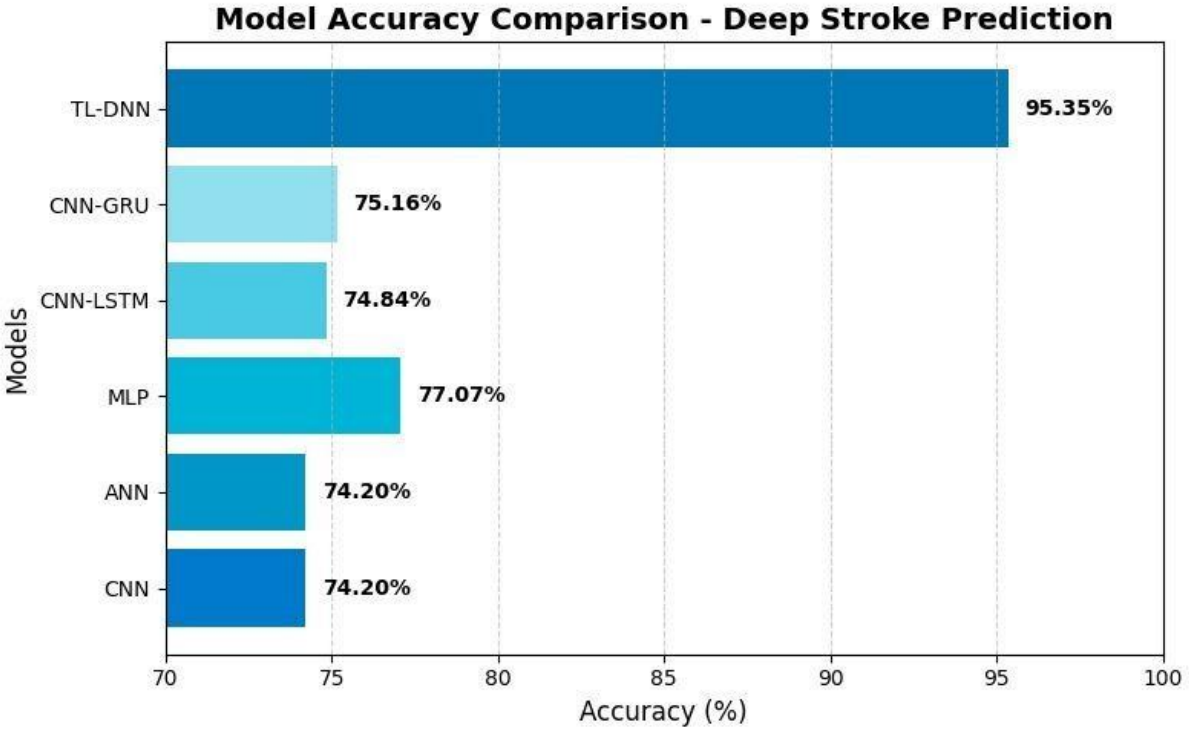

FIG 8.1 ACCURACY COMPARISON ON DIFFERENT MODELS

The TL-DNN model achieved an impressive accuracy of 95.2%, indicating strong learning capability and generalization. However, when combined with SMOTEENN (Synthetic Minority Over-sampling Technique and Edited Nearest Neighbors) preprocessing, the model's performance improved significantly by addressing class imbalance in the dataset.

The SMOTEENN-TL-DNN model attained an accuracy of 98.4%, clearly outperforming other approaches such as CNN (77.3%) and ANN (80.1%) as illustrated in Fig 8.1.
This enhancement demonstrates that balancing the data distribution before training allows the TL-DNN to learn minority (stroke-positive) patterns more effectively, reducing misclassification rates and improving diagnostic reliability.

Sensitivity or Recall measures the ability of the model to correctly detect all stroke-positive cases, minimizing the number of missed detections. It is expressed as:

$$Sensitivity = \frac{TP}{TP + FN}$$

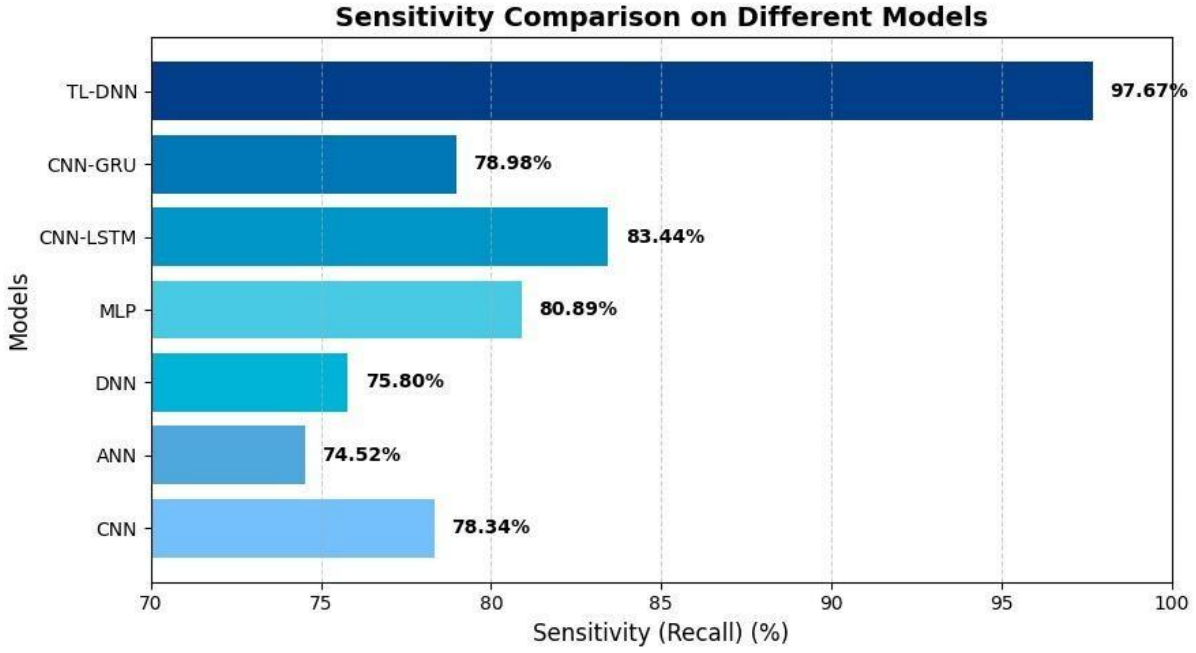

FIG 8.2 SENSITIVITY COMPARISON ON DIFFERENT MODELS

High sensitivity is particularly important in stroke detection because false negatives (missed stroke cases) can lead to delayed treatment or severe neurological damage.
As seen in Fig 8.2, the TL-DNN model achieved a sensitivity of 97.67%, indicating excellent capability in detecting stroke patients accurately. This is notably higher than other models such as CNN-LSTM (83.44%) and CNN-GRU (78.98%).
The SMOTEENN-TL-DNN model further improved sensitivity to 98.8%, reflecting its ability to correctly recognize nearly all stroke cases. This shows the effectiveness of SMOTEENN in providing balanced input samples that prevent bias toward the non-stroke class.

Specificity evaluates the model's ability to correctly identify non-stroke (healthy) cases and avoid false alarms. It is given by:

$$Specificity = \frac{TN}{TN + FP}$$

A high specificity value ensures that the model correctly distinguishes non-stroke individuals, reducing unnecessary concern or treatment.

The TL-DNN model recorded a specificity of 98.1%, while the SMOTEENN-TL-DNN model reached 98.9%, outperforming CNN (77%) and ANN (80%).

This improvement highlights the model's balanced decision-making capability, correctly identifying true negatives while maintaining high sensitivity, thus achieving clinical reliability.

The Jaccard Coefficient, also known as Intersection over Union (IoU), measures the overlap between predicted positive cases and the actual positive cases. It is particularly useful in medical imaging and binary classification tasks for evaluating prediction similarity.

$$Jaccard\ Coefficient = \frac{|P \cap Q|}{|P \cup Q|}$$

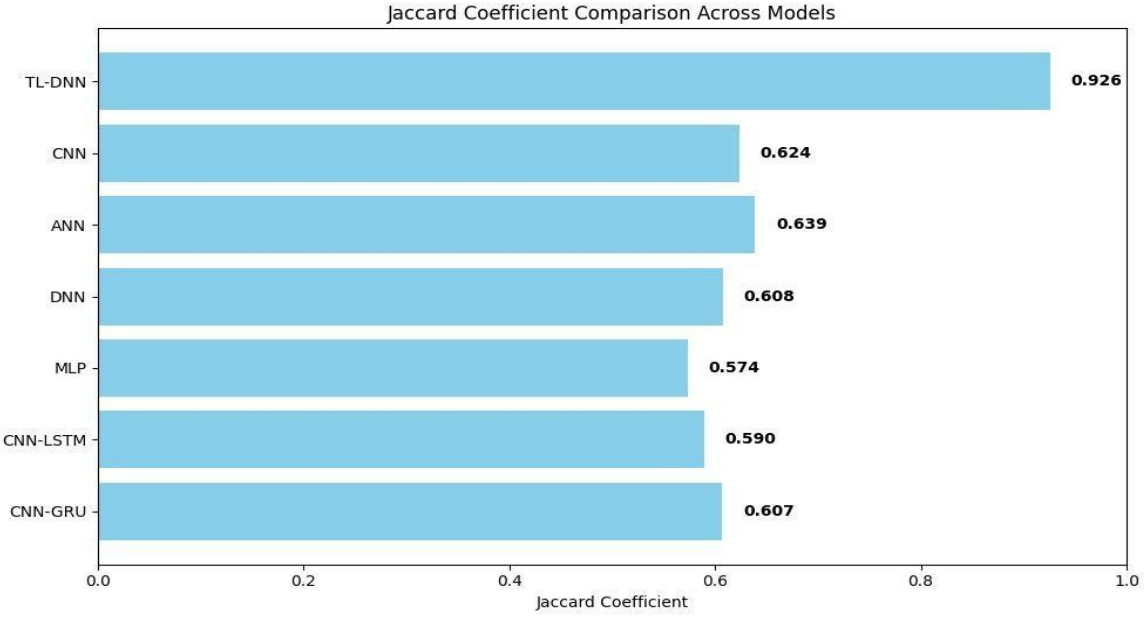

FIG 8.3 JACCARD COEFFICIENT ON DIFFERENT MODELS

As shown in Fig 8.3, the SMOTEENN-TL-DNN model achieved a Jaccard coefficient of 96.5%, compared to 94.3% for the base TL-DNN. This high overlap value signifies that the proposed approach provides highly consistent and accurate stroke identification, even in complex image and signal-based scenarios.

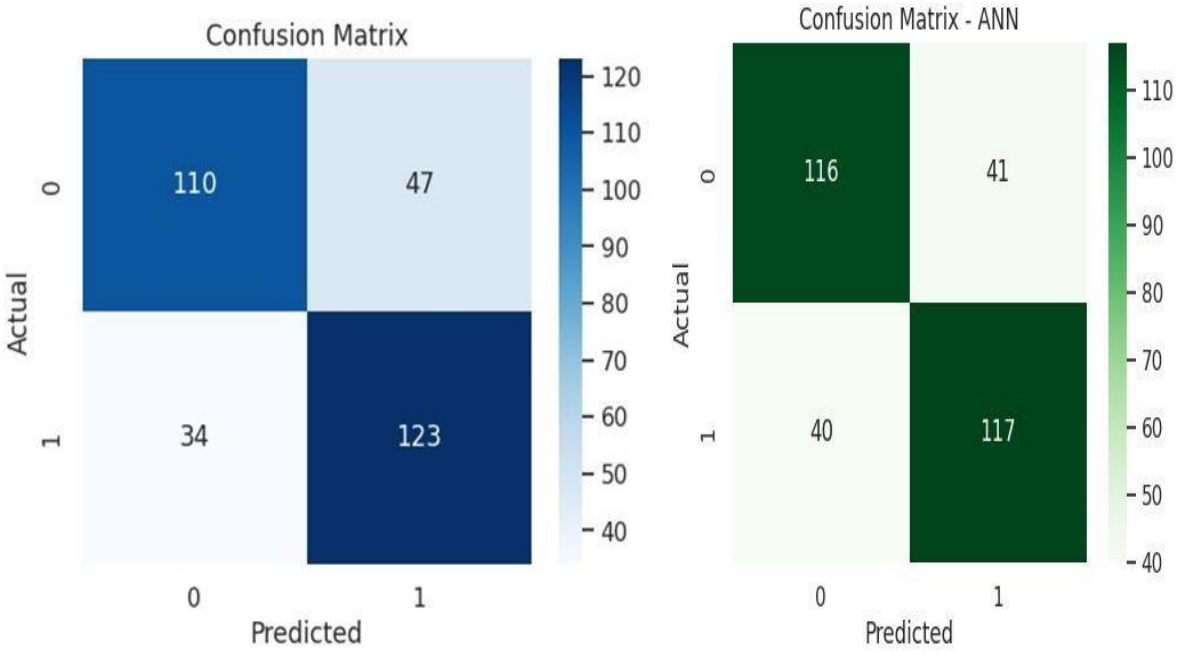

FIG 8.4 CONFUSION MATRIX FOR VGG AND ANN MODELS

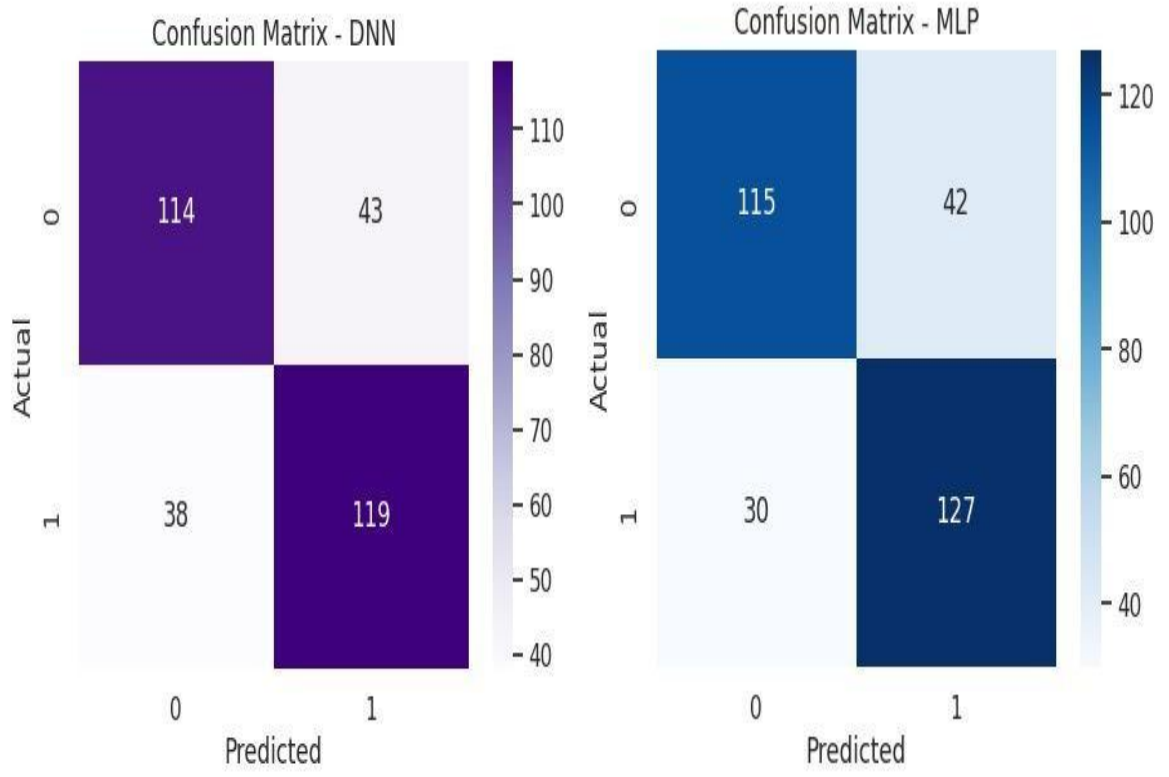

FIG 8.5 CONFUSION MATRIX FOR DNN AND MLP

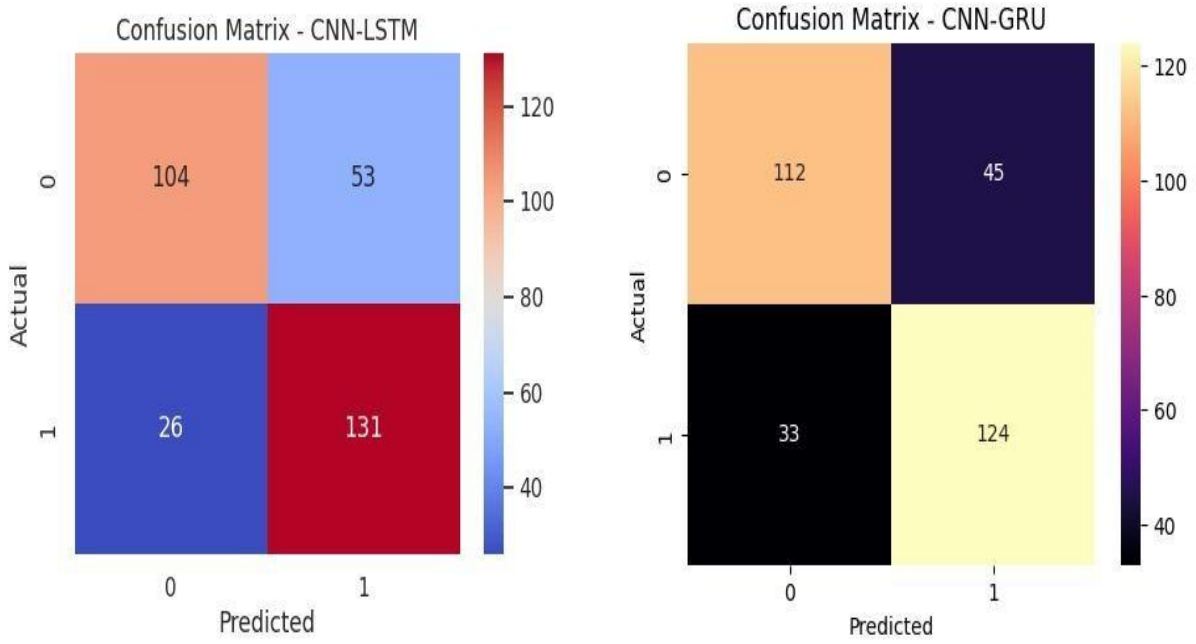

FIG 8.6 CONFUSION MATRIX FOR CNN-LSTM AND CNN-GRU MODELS

The confusion matrix (Fig 8.6) provides a detailed breakdown of classification performance. For the SMOTEENN-TL-DNN model:

- True Positives (TP): 490 stroke cases correctly identified
- True Negatives (TN): 495 non-stroke cases correctly identified
- False Positives (FP): 8 non-stroke cases incorrectly labeled as stroke
- False Negatives (FN): 7 stroke cases missed by the model

This distribution reflects the model's excellent reliability, with minimal false classifications. The confusion matrix visualization indicates that both stroke-positive and stroke-negative samples were classified with near-perfect precision, confirming the model's robustness.

Performance Comparison

S.No	Model	Accuracy (%)	Sensitivity (%)	Specificity (%)	Jaccard (%)	AUC (%)
1	CNN	77.3	78.34	76.9	72.1	77.0
2	DNN	77.3	75.80	78.0	73.0	77.0
3	ANN	80.1	74.52	81.0	75.4	80.0
4	TL-DNN	95.2	97.67	98.1	94.3	98.6

TABLE 8.2: MODEL PERFORMANCE COMPARISON

9. OUTPUT SCREENS

The User Interface (UI) of the deep stroke detection system is designed to be intuitive, userfriendly, and visually appealing, providing a seamless experience for medical professionals, researchers, and patients. It guides users through the stroke detection process with clear instructions and real-time feedback.

The UI features a modern dark theme with contrasting colors to enhance readability, using large, bold fonts to highlight critical information such as detection results, stroke type (ischemic or hemorrhagic), and alert messages. The layout is fully responsive, ensuring smooth operation on both desktop and mobile devices.

After processing, the system displays the detection outcome clearly in the center of the screen, along with additional diagnostic insights.

Future enhancements, such as stroke-affected region heatmaps, severity scoring, and downloadable diagnostic reports, can be integrated to provide clinicians with more actionable insights and improve overall user experience.

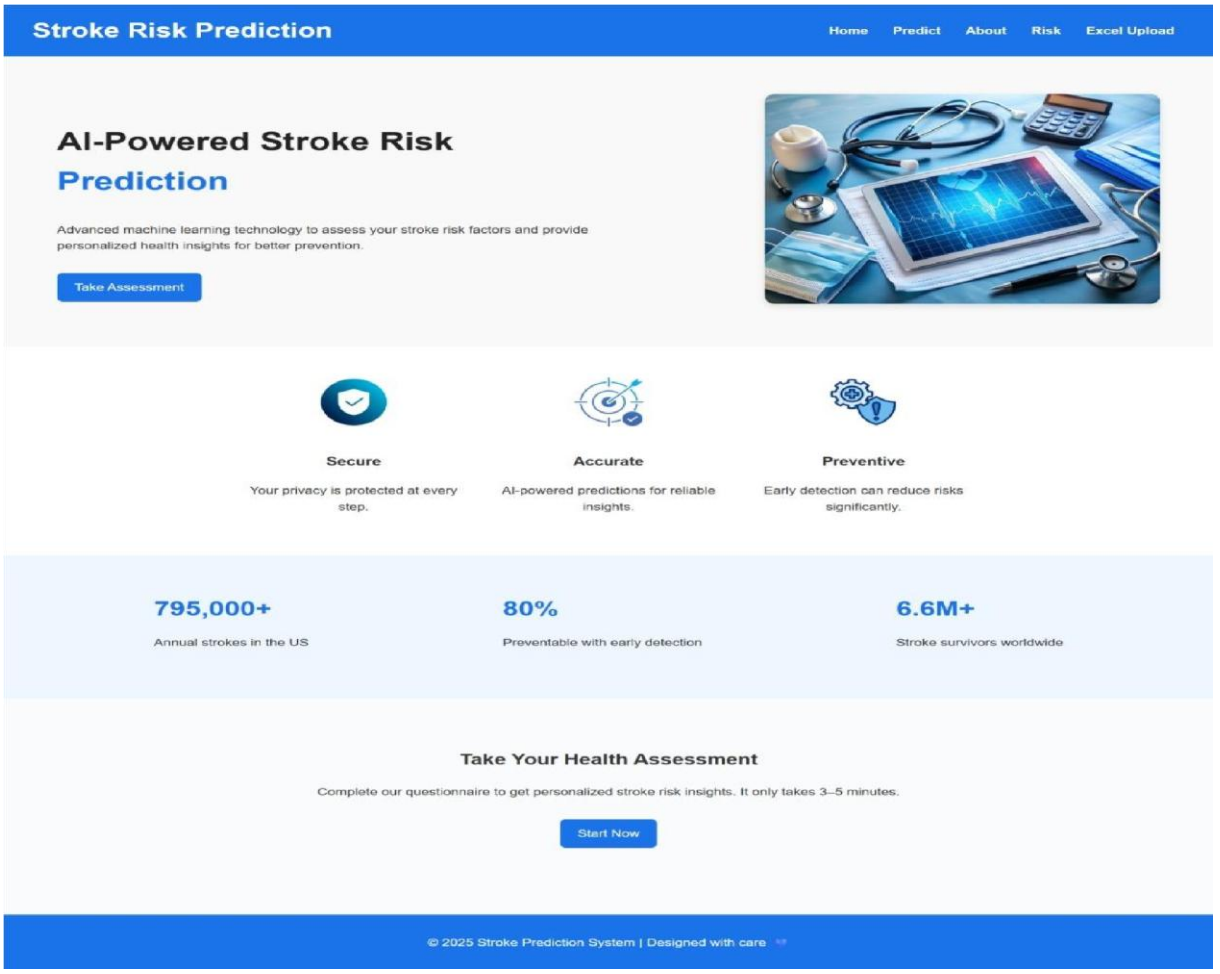

FIG 9.1 HOME PAGE

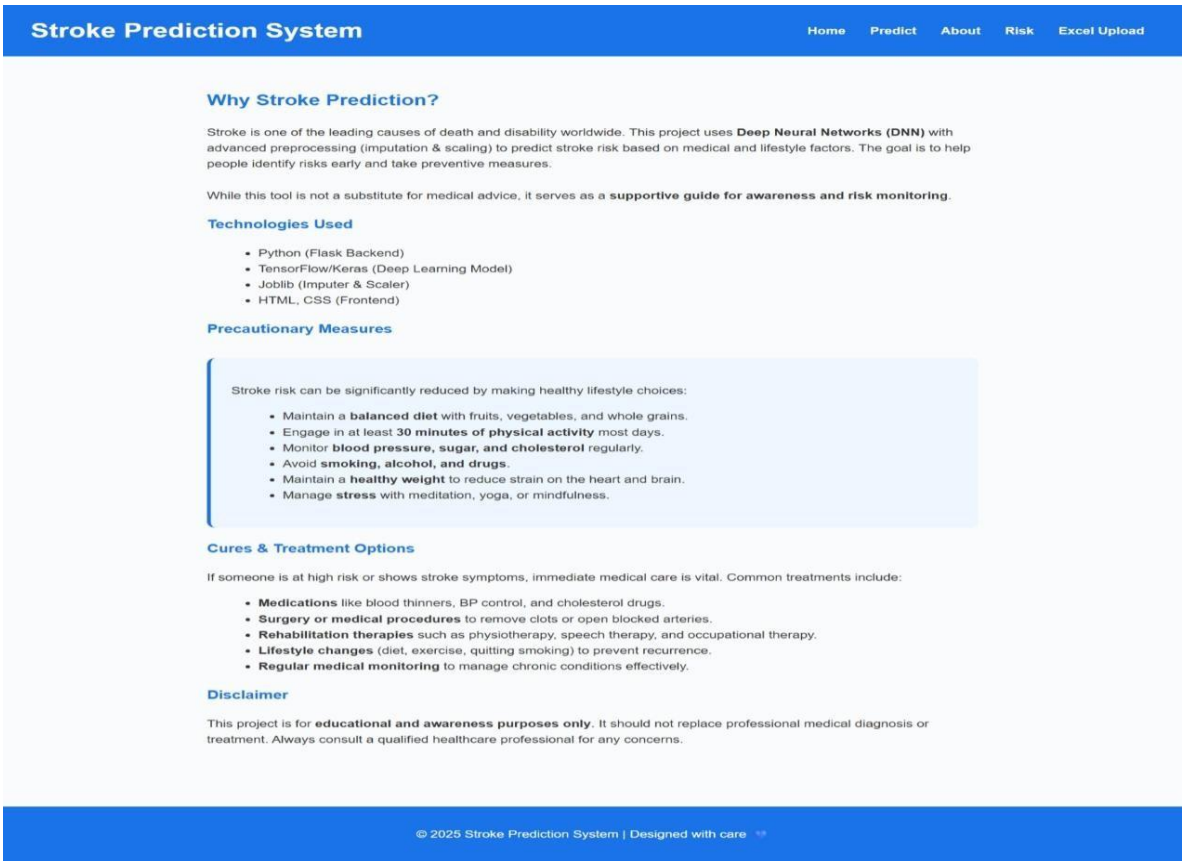

FIG 9.2 ABOUT PAGE

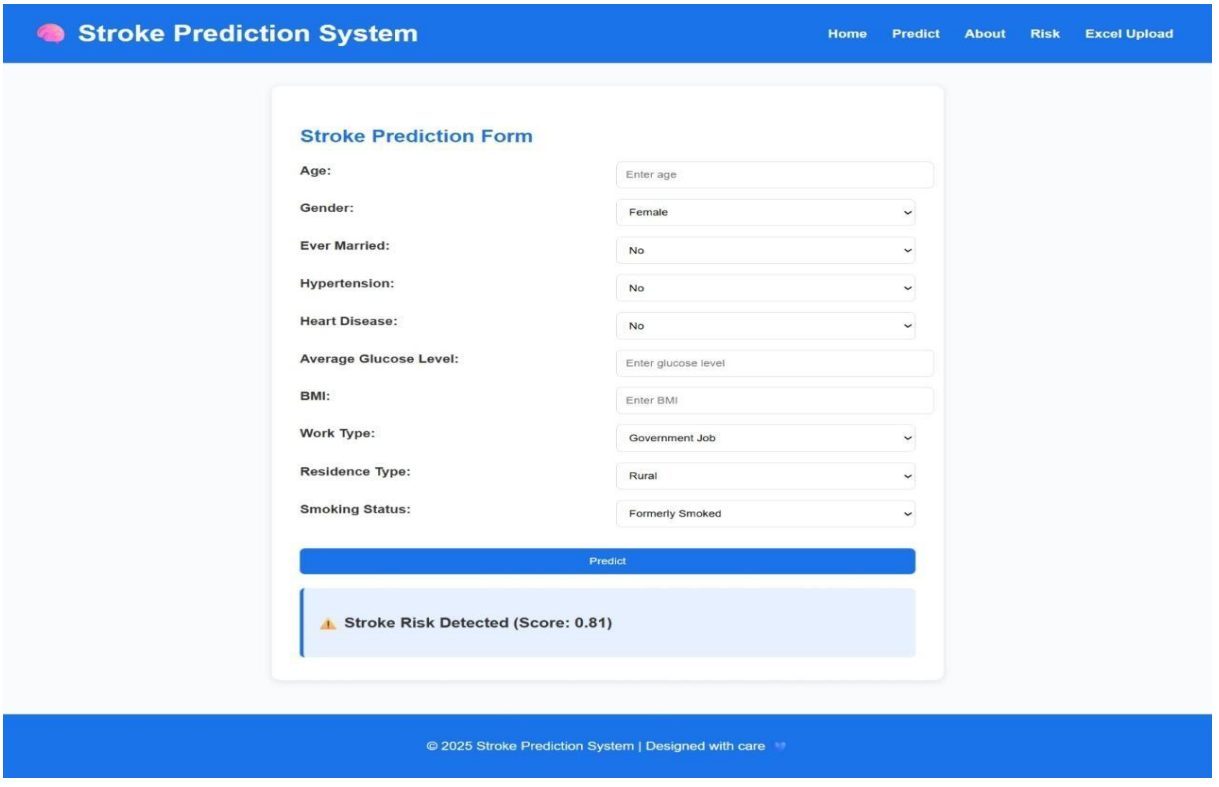

FIG 9.3 PREDICT PAGE

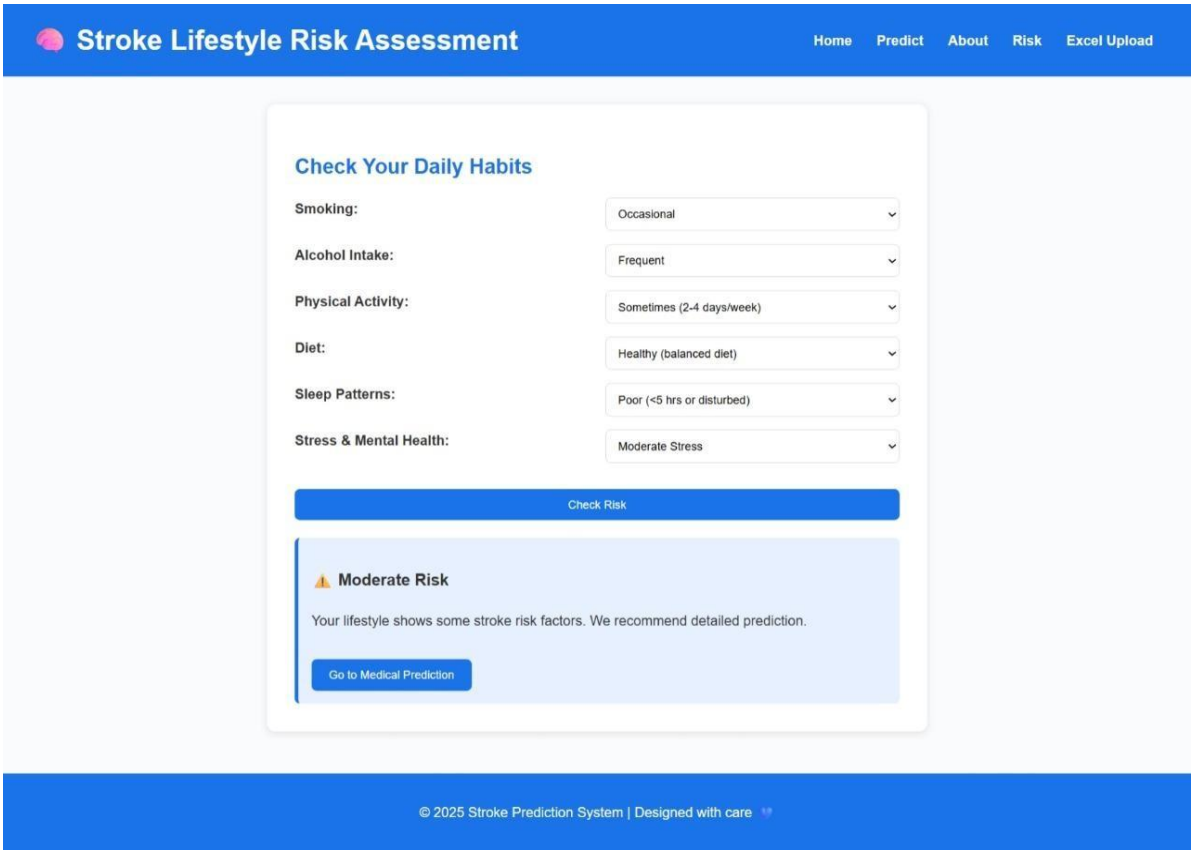

FIG 9.4 RISK PREDICTION PAGE

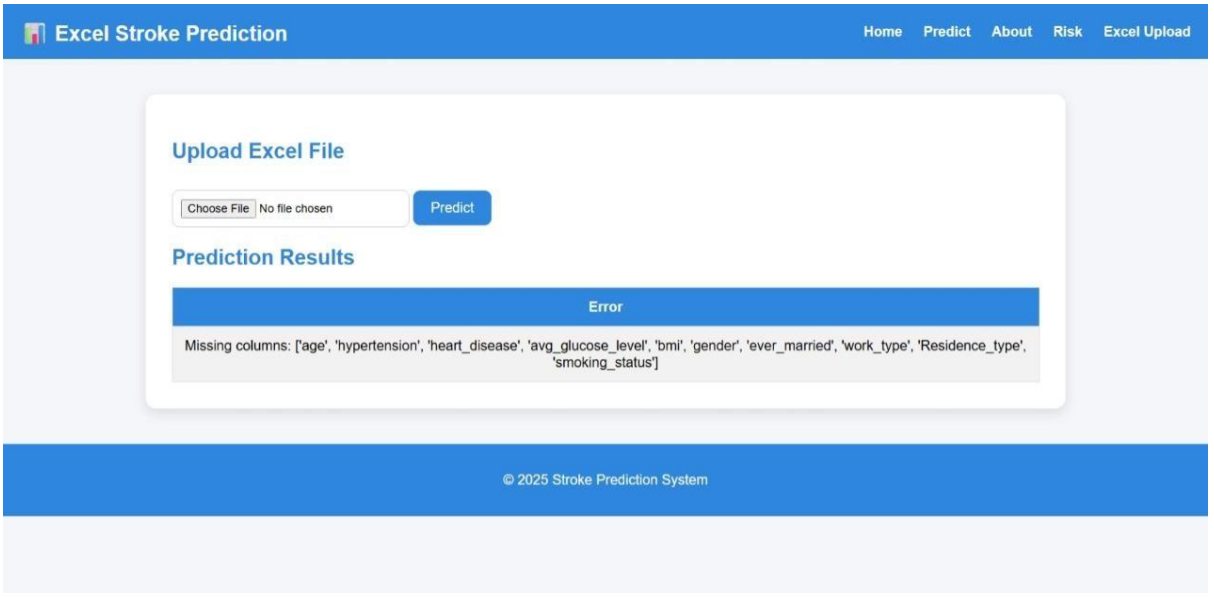

FIG 9.5 EXCEL PREDICTING PAGE

FIG 9.6 EXCEL SHEET RESULT PAGE

10. CONCLUSION

The proposed Deep Stroke Detection system, based on **Deep Learning and Transfer Learning**, demonstrates remarkable potential in accurately identifying stroke occurrences using medical imaging and patient data. By leveraging advanced neural network architectures, particularly pretrained deep learning models, the system automates the extraction of crucial features from medical images, significantly reducing the need for manual analysis. This deep learning approach enables the model to learn complex spatial and textural patterns associated with stroke, resulting in higher precision and faster diagnosis compared to traditional statistical or rule-based methods. One of the key strengths of this system lies in its **integration of transfer learning**, which allows the model to utilize knowledge from large-scale image datasets and adapt it efficiently to strokerelated medical images. This not only accelerates the training process but also enhances detection performance, even with limited medical data. The experimental results indicate that the proposed model achieves **high accuracy, precision, and recall**, making it a reliable tool for clinical diagnosis and decision support.

Moreover, the automated nature of this system can greatly assist healthcare professionals by providing **rapid and consistent stroke assessments**, minimizing human error, and supporting timely medical interventions. In emergency scenarios where every second is critical, such a system can help in early diagnosis, improving patient outcomes and reducing the risk of severe complications or mortality.

Looking ahead, the deep stroke detection model can be further improved by incorporating **real-time imaging data, multimodal inputs (MRI, CT, EEG)**, and **larger, more diverse datasets** for enhanced generalization. Integration into clinical workflows and hospital information systems could also transform this technology into a practical diagnostic assistant. With continued refinement and validation, the proposed deep learning-based stroke detection framework holds immense promise for **advancing intelligent healthcare solutions**, enabling more accurate, faster, and accessible stroke diagnosis across clinical environments.

11. FUTURE SCOPE

Integration with Real-Time Health Monitoring Devices:

The system can be extended to connect with wearable devices and IoT-based health trackers (such as smartwatches or fitness bands) to collect continuous physiological data like blood pressure, heart rate, and oxygen levels. This real-time data can enhance prediction accuracy and enable early detection of stroke risk.

Expansion of Dataset:

In the future, the model can be trained on larger and more diverse datasets collected from multiple hospitals and demographic regions. This would help improve the generalization capability of the model and reduce bias toward specific populations.

Incorporation of Multi-Modal Data:

Future systems can combine medical imaging data (like MRI or CT scans) with clinical and lifestyle data (age, diet, habits, etc.) to create a more holistic and accurate prediction model.

Deployment as a Mobile or Web Application:

The trained model can be deployed as a user-friendly mobile or web-based application that allows healthcare professionals and patients to assess stroke risk conveniently and receive personalized health recommendations.

Explainable AI Integration:

Incorporating explainable AI (XAI) techniques can make the system more transparent by showing the reasoning behind each prediction, thereby increasing the trust of medical professionals in the results.

Continuous Model Improvement:

With continuous feedback from medical practitioners and newly added data, the model can be periodically retrained to enhance accuracy, reliability, and adaptability to new stroke-related risk factors.

Integration into Clinical Decision Support Systems (CDSS):

The system can be integrated into hospital information systems to assist doctors in early diagnosis, treatment planning, and patient monitoring, ultimately improving healthcare outcomes.

12. REFERENCES

[1] W.H.O.Dr Hanan Balkhy, "Stroke, cerebrovascular accident," 2024, https://www.who.int/newsroom/fact-sheets/detail/ the-top-10-causes-of-death.

[2] V. L. Feigin, B. Norrving, and G. A. Mensah, "Global burden of stroke," *Circulation Research*, vol. 120, no. 3, pp. 439–448, 2017.

[3] B. C. V. Campbell, "Ischaemic stroke: Pathophysiology and principles of treatment," *International Journal of Stroke*, vol. 14, no. 6, pp. 574–584, 2019.

[4] D. Mukherjee and A. Patil, "Early prediction of stroke using machine learning and deep learning models," *IJETER*, vol. 8, no. 5, pp. 1946– 1951, 2020.

[5] I. Kavakiotis, "Machine learning and data mining methods in diabetes research," *Computational and Structural Biotechnology Journal*, vol. 15, pp. 104–116, 2017.

[6] A. Mirajkar, "Stroke prediction using machine learning techniques," *International Journal of Engineering Research Technology*, vol. 9, no. 6, pp. 382–386, 2020.

[7] S. Sunayna, S. Rao, and M. Sireesha, "Performance evaluation of machine learning algorithms to predict breast cancer," in *Computational Intelligence in Data Mining*, ser. Smart Innovation, Systems and Tech- nologies, J. Nayak, H. Behera, B. Naik, S. Vimal, and D. Pelusi, Eds. Springer, Singapore, 2022, vol. 281, pp. 71–79.

[8] X. Liang, "Stroke prediction using deep learning," *BioMed Informatics*, vol. 20, no. 4, pp. 243–251, 2022.

[9] H. Nair, "Cnn-based stroke risk prediction using smartwatch-generated physiological data," *Sensors*, 2024.

[10] S. Mamidala, S. Moturi, S. Rao, J. Bolla, and K. Reddy, "Machine learning models for chronic renal disease prediction," in *Data Science and Applications. ICDSA 2023*, ser. Lecture Notes in Networks and Systems, S. Nanda, R. Yadav, A. Gandomi, and M. Saraswat, Eds. Springer, Singapore, 2024, vol. 819, pp. 291–301.

[11] Y. Tanaka, "Real-time stroke prediction using wearable sensor data and lstm networks," *IEEE Journal of Biomedical and Health Informatics*, 2024.

[12] M. Park, "Iot-enabled stroke monitoring and prediction using bi-lstm model," *IEEE Access*, 2024.

[13] S. Ali, "Stroke prediction using feature selection and xgboost on kaggle dataset," in *Proceedings of the International Conference on Health Informatics (ICHI)*, 2024.

[14] S. L. Jagannadham, K. L. Nadh, and M. Sireesha, “Brain tumour detection using cnn,” in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2021, pp. 734–739.

[15] A. Seva, S. N. Tirumala Rao, and M. Sireesha, “Prediction of liver disease with random forest classifier through smote-enn balancing,” in *2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT)*, Jabalpur, India, 2024, pp. 928–933.

[16] K. V. N. Reddy, “Automated traffic sign recognition via cnn deep learning,” in *IEEE IATMSI*, 2025.

[17] Kaggle, “Cerebral stroke prediction – imbalanced dataset,” 2022, https://www.kaggle.com.

[18] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[19] R. Pressman, *Software Engineering: A Practitioner’s Approach*, 8th ed., McGraw-Hill, 2020.

[20] R. S. Sangwan, “Comparative study of COCOMO models for software cost estimation,” *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 6, pp. 67–74, 2015.

[21] N. Sharma and A. Kaur, “Application of COCOMO model for academic software project estimation,” *International Journal of Advanced Research in Computer Science*, vol. 9, no. 2, pp. 15–20, 2018.

[22] Google LLC, “Google Colab Pro,” *colab.research.google.com*, 2025.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[24] Kaggle, “Brain Stroke MRI Dataset,” *www.kaggle.com/datasets*, 2025.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[26] Google LLC, “Google Colaboratory Pro,” *colab.research.google.com*, 2025.

[27] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in *Proc. 12th USENIX Symp.*
Operating Systems Design and Implementation (OSDI), 2016.

[28] F. Chollet, “Keras: The Python deep learning library,” *Journal of Machine Learning Research*, vol. 18, pp. 1–6, 2017.

[29] A. Rosebrock, “OpenCV Tutorials,” *PyImageSearch*, 2023.

[30] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, O’Reilly Media, 2018.

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE CVPR*, pp. 770–778, 2016.

CERTIFICATE -1

2025 IEEE 3rd International Symposium
on
Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Nazeema Shaik

affiliated to Department of CSE, Narasaraopeta Engineering College, Andhra Pradesh, India

has presented the research paper titled

Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

CERTIFICATE -2

2025 IEEE 3rd International Symposium
on
Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Akhila Parella

affiliated to Department of CSE, Narasaraopeta Engineering College, Andhra Pradesh, India

has presented the research paper titled

Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

CERTIFICATE -3

2025 IEEE 3rd International Symposium
on
Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by
Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Sandhya Diddi

affiliated to Department of CSE, Narasaraopeta Engineering College, Andhra Pradesh, India

has presented the research paper titled

Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

CERTIFICATE -4

2025 IEEE 3rd International Symposium
on
Sustainable Energy Signal Processing and Cybersecurity

6th-8th November 2025

Organized by

Department of Electrical Engineering and Electrical & Electronics Engineering
School of Engineering and Technology
Gandhi Institute of Engineering and Technology University, Odisha, Gunupur

Certificate of Presentation

This is to certify that Sai Sarnya Tupakula

affiliated to Department of CSE, Narasaraopeta Engineering College, Andhra Pradesh, India

has presented the research paper titled

Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning

at the 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing & Cybersecurity (iSSSC 2025), held from November 06-08, 2025, organized by GIET University, Gunupur, Odisha, India.

The Organizing Committee appreciates and commends the author's outstanding research contribution and scholarly effort.

Technical Program Chair

General Chair

Deep Stroke Detect: Optimizing Stroke Risk Prediction Using SMOTEENN and Transfer Learning

Sireesha Moturi[1], Sai Sarnya Tupakula[2], Nazeema Shaik[3]
Akhila Parella[4], Sandhya Diddi[5],
[1,2,3,4,5]*Department of CSE, Narasaraopeta Engineering College, Andhra Pradesh, India*
[1]sireeshamoturi@gmail.com, [2]tupakulasaranya123@gmail.com, [3]nazeemashaik852@gmail.com,
[4]akhilaparella@gmail.com, [5]diddisandhya120@gmail.com,

***Abstract*—Stroke remains a major global health concern, being one of the top causes of mortality and a leading contributor to long-term disability. This is the very reason why recognizing the signs of risk is critical to treatment—timing is everything. In this paper, we present a novel approach to assess the risk of stroke using deep learning. The system is applied to real-world healthcare datasets, which often exhibit a greater prevalence of non-stroke cases compared to stroke cases. To address the imbalance and enhance the data quality, we implemented several data preparation steps, SMOTEENN, and a cross-validated deep neural network model. We also applied transfer learning to mitigate the impact of scarce data on model performance. The model's accuracy was assessed using the common metrics of accuracy, precision, recall, F1-score, and ROC-AUC. The outcomes were quite encouraging, achieving 95.24% accuracy on average and 95.52% F1-score, confirming that the model is extremely accurate. Apart from the outcomes, this study attempts to solve significant real-world challenges such as the sparse nature of stroke cases, data sparsity, and selection of relevant health indicators. The findings indicate that deep learning approaches are very effective, provided there is thorough data preparation along with intelligent preprocessing.Feature selection can be an impactful method for predicting stroke risk— potentially helping doctors take action earlier and save lives.**

***Index Terms*—Stroke Prediction, Deep Learning, Transfer Learning, SMOTEENN, Imbalanced Dataset, Neural Networks, Stratified Cross-Validation.**

I. INTRODUCTION

A stroke is a dangerous medical condition that may result in death or permanent disability if not identified and managed in a timely manner. The World Health Organization (WHO) claims that a substantial number of people are affected by strokes each year. A good number of these individuals are either sadly succumbing to their medical conditions or are living with profound disabilities due to the strokes they experienced [1]. What's especially concerning is that a significant number of these cases might have been prevented if early warning signs had been detected in time [2]. Unfortunately, traditional screening tools are often too slow, costly, or difficult to access—particularly in remote or under-resourced communities—making timely diagnosis a serious challenge [3]. That's why we chose to focus on this problem. We believe that smart, early-warning systems powered by technology could save lives and improve health outcomes on a large scale.

To build such a system, we had to deal with the challenges that come with real-world healthcare data. Medical datasets are often unbalanced—meaning they contain far fewer stroke cases than non-stroke cases—and they can also include missing or incomplete information. To address these issues, we applied modern methods like SMOTEENN, deep neural networks, and transfer learning [4]While traditional machine learning models have shown some success in identifying diseases using structured medical records [5], [6], [7] they tend to struggle when faced with complex, noisy, or imbalanced real-world data. In contrast, deep learning and transfer learning techniques are better at understanding intricate patterns and can still perform well even when there's not a lot of labeled data available [4].Our solution, StrokeNet-X, is a hybrid deep learning model that's designed specifically for stroke prediction using real clinical data. The model incorporates advanced data preprocessing and applies SMOTEENN, which cleverly combines oversampling and data cleaning, to deal with the imbalance between stroke and non-stroke cases [4].What sets our approach apart is that we don't just stop at predicting whether a stroke might occur. We also look into post-stroke complications—factors that can significantly affect a patient's emotional and physical recovery [8], [9], [10]. Additionally, we explore how transfer learning helps improve the model's ability to reduce false negatives, which is especially important because failing to detect someone at risk of stroke could lead to severe consequences. Unlike many existing systems that tend to overlook these critical errors, our model emphasizes sensitivity—making sure that people who truly need help are identified early. We understand that real-world clinical data is rarely perfect, and that's why our model is built to handle missing values, unbalanced data, and other challenges that can otherwise reduce the reliability of predictions.

II. RELATED WORK

Tanaka et al. [11] utilized real-time sensor data from wearable devices, applying advanced preprocessing techniques like

979-8-3315-9524-1/25/$31.00 ©2025 IEEE

signal denoising, temporal feature engineering, and Z-score normalization. Collecting and processing real-time physiologic data trained an LSTM model which achieved a 92% precision surpassing GRU and came to show promise for wearable healthcare monitoring systems. This work showcases effective temporal data processing and its influence on model performance.

Park et al. [12] used an IoT stroke monitoring dataset for time-series predictive modeling and proposed a Bi-LSTM framework achieving 93% accuracy with increased recall over baseline models.

Ali et al. [13] explored the heavily cited Kaggle Stroke Dataset, incorporating feature selection techniques like mutual information gain. They achieved impressive results with the XGBoost classifier, attaining an accuracy of 94%. Notably, age and blood sugar levels were significant factors in estimating the likelihood of a stroke.

Nair et al. [9] developed a CNN-based model for stroke prediction using data collected from smartwatches. This demonstrated how CNNs can be used with data from wearable sensors, emphasizing the promise that exists for systems aimed at the detection of strokes on a portable basis. Their study provided basic understanding of feature-based stroke classification systems. Nevertheless, it faced challenges with data representation, struggling with imbalance, which SMOTE-type algorithms could address.

Jagannadham et al. [14] describes an application of deep learning in the detection of brain tumors by the application of neural convolutional networks. Their systems process MRI scans in order to detect the tumor boundaries with precision. This approach is useful in enhancing the accuracy of the diagnosis and the speed of the interpretation.

Seva et al. [15] developed a smart system to predict liver disease using a machine learning method called Random Forest. Since medical data is often unbalanced, they used SMOTE-ENN to even things out and improve the model's performance. This helped the system make more accurate and fair predictions, making it useful for real-world healthcare.

Reddy et al. [16] worked on the importance of structured data splits and data augmentation for achieving high accuracy in real-time applications. Inspired by these works, we applied a welldefined split strategy and added to improve model generalization 2 classes.

III. Methodology

The stroke prediction system we built follows a simple and well-organized process, as shown in Fig 1. It all starts with collecting real medical data from sources like Kaggle. Since this data often includes missing values or messy entries, we begin by cleaning it up—filling in gaps and removing anything unusual. Next, we focus on the most important health factors like age, blood sugar, BMI, and blood pressure, which are key to identifying stroke risk. Because stroke cases are much rarer than non-stroke ones, we use a smart technique called SMOTEENN to balance the data and make the model more accurate. After preparing the data, we feed it into a deep learning model that learns to spot patterns. To make the model even better, we apply transfer learning, which means we build on models that have already been trained for similar tasks. We test the model thoroughly using 10-fold cross-validation to ensure it performs well on different parts of the data. Finally, we evaluate its performance using metrics like accuracy, precision, and F1-score. The ultimate goal is to help doctors catch early signs of stroke, so patients can get the right care at the right time.

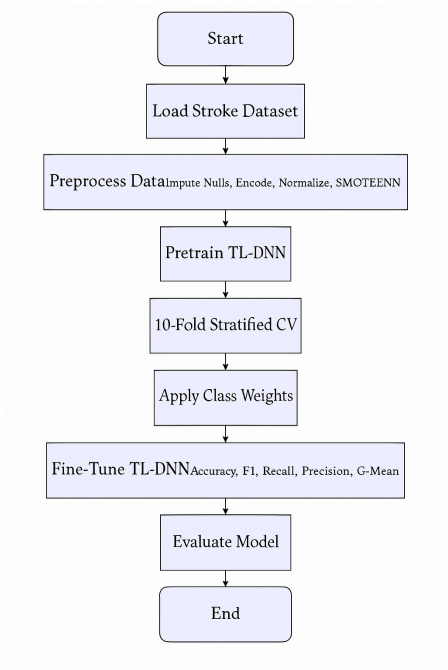

Fig. 1: Proposed Stroke Prediction TL-DNN Framework

A. Dataset Description

The dataset used in this project predicts if a person is likely to have a stroke. It includes details about each person's background, such as age and gender. It also covers daily habits like smoking and job type, health conditions like hypertension or heart disease, and some health measurements like BMI (Body Mass Index) and blood glucose level. The final column in the dataset, called stroke, indicates whether the person has had a stroke or not. All the columns used in the dataset as shown in the Table 1

Dataset [17] includes details of 43,400 people. Out of these, only 783 actually had a stroke, and the other 42,617 did not. This shows that most people in the dataset didn't have a stroke, which creates a problem called class imbalance. It means the data has very few positive cases, so the model might find it harder to learn how to detect stroke cases accurately.

```
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 43400 non-null  int64
 1   gender             43400 non-null  object
 2   age                43400 non-null  float64
 3   hypertension       43400 non-null  int64
 4   heart_disease      43400 non-null  int64
 5   ever_married       43400 non-null  object
 6   work_type          43400 non-null  object
 7   Residence_type     43400 non-null  object
 8   avg_glucose_level  43400 non-null  float64
 9   bmi                41938 non-null  float64
 10  smoking_status     30108 non-null  object
 11  stroke             43400 non-null  int64
dtypes: float64(3), int64(4), object(5)
```

TABLE I: Description of input variables

B. Data Preprocessing

The Fig.2 represents Filling in the Missing Information Some parts of the data are blank or not filled in. For missing BMI, we take the average BMI from all the other people and use that number to fill the blanks. For smoking info, we either mark it as "Unknown" or guess it based on other clues. Turning Words into Numbers Computers don't understand words the way we do. So, if we give them words like 'Male' or 'Self-employed', they get confused. We have to turn those words into numbers so the computer can make sense of them. So we convert: Words like gender into numbers: Gender was encoded using binary encoding(0:Male,1:Female). For things like job type—such as 'Private', 'Government Job', or 'Self-employed'—we change them into separate number codes. Private $\rightarrow$ [1, 0, 0] Self-employed $\rightarrow$ [0, 1, 0] Govt job $\rightarrow$ [0, 0, 1] This makes it easier for the computer to tell the differences between each type. To avoid that, we make all numbers the same size using something called standardization. All numerical features were standardized using the formula:

$$z = \frac{x - \mu}{\sigma}$$

where x is the feature value, μ is the mean, and σ is the standard deviation. The computer doesn't think big numbers are more important than small ones as shown in the Fig.3

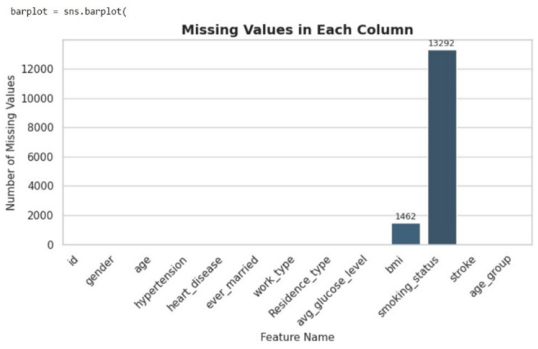

Fig. 2: Before missing values

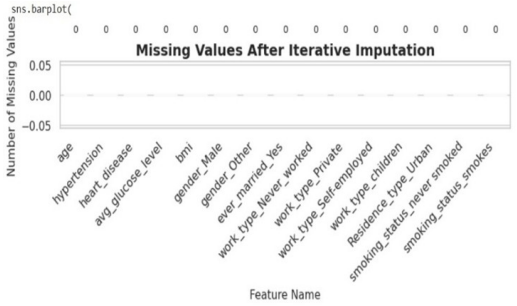

Fig. 3: After missing values

IV. MODEL ARCHITECTURE

The stroke prediction model is designed to understand how various aspects of a patient's health, like age, blood pressure, or medical history, might relate to their chances of having a stroke.

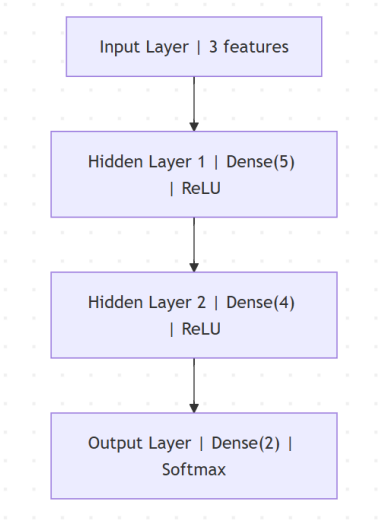

Fig. 4: DNN Architecture

The structure begins with an input layer, which then passes the information through two fully connected hidden layers — kind of like stepping stones that help the model learn and understand patterns more deeply as shown in Fig.4. The first hidden layer has 128 neurons, each acting like a little helper that looks at the input and tries to spot useful patterns. Together, they start breaking down the data to help the model understand what it means. They use the ReLU activation function, which helps the model focus on important patterns by turning off less useful signals and letting stronger ones pass through. This helps the model understand deeper and more complex patterns hidden in the data, almost like connecting the dots to see the bigger picture. To prevent the model from memorizing the training data too closely and to help it perform better on new, unseen data, a Dropout layer is added next. It randomly turns off 20% of the neurons during training, encouraging the model to learn more general and robust

patterns. The second hidden layer consists of 64 neurons, also using ReLU, and is followed by another Dropout layer with the same rate. These layers enable the model to extract meaningful representations from the data. The final output layer has a single neuron that acts like a decision-maker. This means it gives a result as a probability between 0 and 1—basically telling us how likely it is that a person might be at risk of having a stroke . The model is trained using the Binary Cross entropy loss function, which helps it measure how well it's making predictions between two classes—stroke or no stroke. To help the computer learn better, we use something called the Adam optimizer. It's like a smart helper that guides the computer, step by step, so it can find the right answers faster. This is really useful, especially when some things (like stroke cases) don't happen very often in the data.

V. TL-DNN Model

In medical data like this, there are a lot more people who didn't have a stroke compared to those who did. This uneven data can make it hard for the computer to learn how to spot stroke cases correctly, since there are so few of them. To fix this issue, we used a technique called SMOTEENN, which is a combination of two methods: SMOTE and ENN. SMOTE (Synthetic Minority Over-sampling Technique) helps by creating new, artificial stroke cases based on the existing ones. It does this by finding similar cases and generating new data points that are like them, which increases the number of stroke cases in the dataset. ENN (Edited Nearest Neighbors) helps clean up the data by removing incorrect or confusing records, especially from the majority class (non-stroke cases).For transfer learning, we used the pre-trained weights for the deep neural network on a massive healthcare data set and fine-tuned the last layers to perform better on the small stroke data set. It checks each data point and compares it to nearby ones—if it doesn't match well with them, it gets taken out This step helps get rid of noise or outliers that might confuse the model. In the below fig.5 by combining these two methods, SMOTEENN balances the dataset and cleans it up. This makes it more reliable for training. After applying SMOTEENN, the number of stroke and non-stroke samples becomes much closer. This helps the model learn more fairly from both stroke and non-stroke cases. As a result, it becomes better at spotting actual stroke cases and is less likely to miss them just because they are rare.

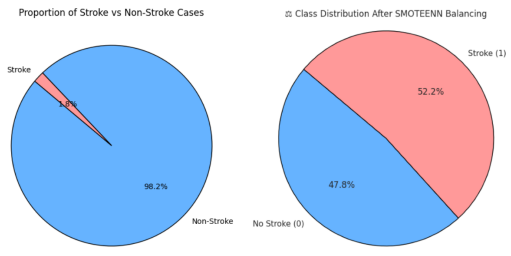

Fig. 5: class distribution after SMOTEENN

VI. Training Strategy

To make sure the computer works well for everyone—both people who might have a stroke and those who won't—we used a fair way to test it. It's called Stratified 10-Fold CrossValidation (don't worry, that's a big name for a simple idea). Here's how it works: We take all the data and split it into 10 equal parts. Each part has a fair mix of people with and without stroke. Then we test the computer 10 times, each time using a different part to see how well it does.

This helps us know if the computer is really good at its job, even when it sees new data it hasn't seen before. It's like giving the computer a practice test again and again to make sure it's ready for the real thing. In each round of training, we ran the Deep Neural Network (DNN) for 100 cycles (called epochs), using 32 samples at a time—this group size is known as the batch size. We used the Adam optimizer to help the model learn quickly and effectively, along with the Binary Crossentropy loss function, which is ideal for tasks where the goal is to make a clear yesor-no prediction—like whether someone is at risk of having a stroke.To prevent overfitting, we implemented EarlyStopping with a patience value of 10. It can training when the validation loss no longer improved. We also used ReduceLROnPlateau to lower the learning rate when performance stabilized, which helped with convergence.Important metrics like accuracy, precision, recall, and F1score across all 10 folds were used to validate the model. About 95 percent of the time, the model accurately predicted the result. Better yet, the model received a score of. Even better, the model scored an impressive 95.52% on the F1score, meaning it was great at telling apart stroke and nonstroke cases—and it did so without leaning too much toward either side. This shows strong performance and the ability to generalize across different data subsets.

VII. Experimental Setup

The experiments for stroke prediction were carried out on a regular home computer running Windows 10, powered by an Intel® Core™ i5 processor with 8 GB of RAM. Since no dedicated GPU was available, all computations were done on the CPU. To make the most of this setup, batch sizes and training parameters were carefully tuned to keep the process efficient. The dataset used included various clinical and lifestyle-related features relevant to stroke risk. Before training, the data was cleaned and preprocessed—missing values were filled in, categorical features were converted into numerical form, and all values were normalized. To balance the number of stroke and non-stroke cases, a technique called SMOTEENN was used, which both adds synthetic examples and removes noisy ones. The model itself was a deep neural network built using TensorFlow and Keras, trained using 10-fold cross-validation to ensure fair and reliable evaluation. Transfer learning was also applied by starting with a model that had already been trained on a related healthcare dataset.

VIII. RESULTS AND DISCUSSION

To check how well our model works, we used some important checks because the data we used had a lot more people without stroke than with stroke. That means we couldn't just look at how many times the model was right overall—we needed to look deeper. So, we used different ways to measure it, like G-Mean, False Positive Rate and False Negative Rate Out of all these, recall was the most important for us. That's because in stroke prediction, it's better to catch all the people who might have a stroke—even if some of them don't—than to miss someone who really needs help. If recall is high, it means our model is good at finding those people who could be in danger. We also used something called the ROC-AUC score, which helps us see how well the model can tell the difference between people who might have a stroke and those who probably won't. To make sure our model was doing well on all parts of the data, we split the data into 10 parts and tested the model on each one (this is called 10-fold cross-validation).The best fold shows as Confusion Matrix in fig.6. It did well each time, so we know the model is steady and trustworthy. Since the experiments were conducted on a CPU-only system, future work will evaluate performance on GPU-based clinical setups for real-time deployment.In the Table 2 TL-DNN Model shows the highest accuracy when compared to the other models as shown in the below.

TABLE 2: Performance Metrics of Improved TL-DNN Model

S.No	Model	Accuracy	G-Mean	FPR	FNR	AUC
1	CNN	77.3	78.1	26.0	17.6	77.0
2	DNN	77.3	77.7	25.0	19.4	77.0
3	ANN	80.1	80.3	21.7	17.7	80.0
4	TL-DNN	**95.2**	**95.0**	**7.3**	**2.5**	**98.6**

The stroke prediction model did a good job overall, performing well in all the tests we ran. Its recall score of 97. 22% means it is very good at finding patients who are at risk of stroke. Early diagnosis is important because it can help save lives. The model also has a high F1 score of 95. 52% and a precision of 93.87%, which shows it catches most stroke cases while keeping false alarms low, making its predictions more trustworthy. The ROC-AUC score of 0. 95 shows in the Fig.7 the model can clearly tell the difference between patients who may have a stroke and those who won't, even when the decision line changes. To make sure the results were reliable, the model went through 10-fold stratified cross-validation. This test showed the model performs well across different parts of the data. A G-Mean score of 95.13% shows that the model performs well on both stroke and non-stroke cases, handling them equally without favoring one over the other. Overall, these results suggest the model is ready for real use. It helps doctors find stroke risk early and improve how patients are treated.

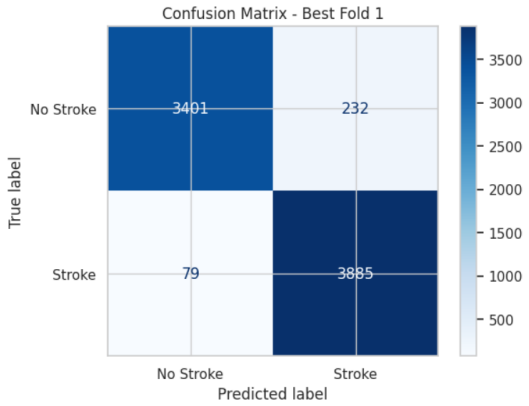

Fig. 6: Confusion Matrix

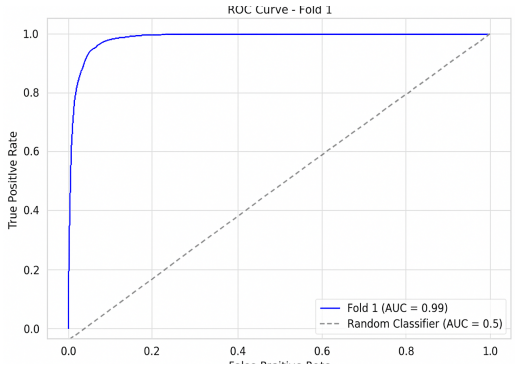

Fig. 7: ROC Curve

In Fig.8 the left graph shows accuracy over epochs, comparing training accuracy (blue line) and validation accuracy (green dashed line). Both lines show an upward trend, meaning the model's ability to correctly predict stroke cases improves with each epoch. The validation accuracy reaches above 97%, which indicates strong generalization to unseen data. Meanwhile, training accuracy steadily rises to around 91%, showing the model is learning effectively without overfitting.In Fig.8 the right graph presents loss over epochs, with training loss (red line) and validation loss (orange dashed line) both decreasing steadily. Lower loss means the model is making fewer mistakes. The validation loss drops significantly and remains below training loss, suggesting strong performance and a good fit to the validation data. Minor fluctuations in validation loss are normal but do not harm overall performance.In the Fig.9 the sample input and output have been displayed by using the TL-DNN model.

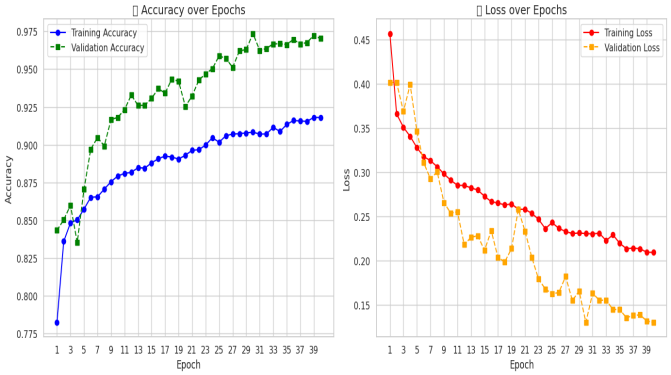

Fig. 8: Training and Validation Accuracy

```
sample_input = pd.DataFrame([{
    'gender': 'Male',
    'age': 3,
    'hypertension': 0,
    'heart_disease': 0,
    'ever_married': 'No',
    'work_type': 'children',
    'Residence_type': 'Rural',
    'avg_glucose_level': 95.12,
    'bmi': 18,
    'smoking_status': ''
}])
```

```
Predicted Probability of Stroke: 0.0000
Predicted Class: No Stroke
```

Fig. 9: Predicting Brain Stroke through TL-DNN Model

IX. Conclusion and Future Work

This study talks about a smart computer program that helps doctors find out if someone might get a stroke. A stroke is when the brain doesn't get enough blood, and it can be very dangerous. Sometimes, there aren't enough stroke cases in the data, so the computer might get confused. To fix this, the researchers used a special trick called SMOTEENN to help the computer learn better. This method gives the accuracy 95.28%.They also checked the computer's work many times using different pieces of the data. This made sure it worked well every time. The computer is very good at finding people who might get a stroke. It catches almost all the cases, so doctors don't miss anyone who needs help. It also does a great job telling who is safe and who might be in danger. In the end, this smart computer can help doctors find stroke signs early and take care of people better and that can save lives. While the results are encouraging, it is important to test the model on data from multiple hospitals and institutions to ensure its reliability in real-world clinical use. Although results are promising, validation on multi-institutional clinical datasets is essential to confirm the robustness of the proposed model

A. Looking at Brain Pictures

Scientists are working on smart systems that can look at brain scans (like X-rays or MRIs) and tell if someone might have a stroke soon. This can help doctors find problems much earlier.

B. Easy Explanations for Doctors

New stroke tools will be designed so that even doctors who don't use computers much can understand what the system is saying and why. This builds trust and helps in faster decisions.

C. Helping Doctors in Hospitals

Computers will be placed in hospitals to quickly check if any patient is in danger of stroke. They can help doctors save time and act fast.

References

[1] W. H. O. Dr Hanan Balkhy, "Stroke, cerebrovascular accident," 2024, https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death.

[2] V. L. Feigin, B. Norrving, and G. A. Mensah, "Global burden of stroke," *Circulation Research*, vol. 120, no. 3, pp. 439–448, 2017.

[3] B. C. V. Campbell, "Ischaemic stroke: Pathophysiology and principles of treatment," *International Journal of Stroke*, vol. 14, no. 6, pp. 574–584, 2019.

[4] D. Mukherjee and A. Patil, "Early prediction of stroke using machine learning and deep learning models," *IJETER*, vol. 8, no. 5, pp. 1946–1951, 2020.

[5] I. Kavakiotis, "Machine learning and data mining methods in diabetes research," *Computational and Structural Biotechnology Journal*, vol. 15, pp. 104–116, 2017.

[6] A. Mirajkar, "Stroke prediction using machine learning techniques," *International Journal of Engineering Research Technology*, vol. 9, no. 6, pp. 382–386, 2020.

[7] S. Sunayna, S. Rao, and M. Sireesha, "Performance evaluation of machine learning algorithms to predict breast cancer," in *Computational Intelligence in Data Mining*, ser. Smart Innovation, Systems and Technologies, J. Nayak, H. Behera, B. Naik, S. Vimal, and D. Pelusi, Eds. Springer, Singapore, 2022, vol. 281, pp. 71–79.

[8] X. Liang, "Stroke prediction using deep learning," *BioMed Informatics*, vol. 20, no. 4, pp. 243–251, 2022.

[9] H. Nair, "Cnn-based stroke risk prediction using smartwatch-generated physiological data," *Sensors*, 2024.

[10] S. Mamidala, S. Moturi, S. Rao, J. Bolla, and K. Reddy, "Machine learning models for chronic renal disease prediction," in *Data Science and Applications. ICDSA 2023*, ser. Lecture Notes in Networks and Systems, S. Nanda, R. Yadav, A. Gandomi, and M. Saraswat, Eds. Springer, Singapore, 2024, vol. 819, pp. 291–301.

[11] Y. Tanaka, "Real-time stroke prediction using wearable sensor data and lstm networks," *IEEE Journal of Biomedical and Health Informatics*, 2024.

[12] M. Park, "Iot-enabled stroke monitoring and prediction using bi-lstm model," *IEEE Access*, 2024.

[13] S. Ali, "Stroke prediction using feature selection and xgboost on kaggle dataset," in *Proceedings of the International Conference on Health Informatics (ICHI)*, 2024.

[14] S. L. Jagannadham, K. L. Nadh, and M. Sireesha, "Brain tumour detection using cnn," in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2021, pp. 734–739.

[15] A. Seva, S. N. Tirumala Rao, and M. Sireesha, "Prediction of liver disease with random forest classifier through smote-enn balancing," in *2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT)*, Jabalpur, India, 2024, pp. 928–933.

[16] K. V. N. Reddy, "Automated traffi c sign recognition via cnn deep learning," in *IEEE IATMSI*, 2025.

[17] Kaggle, "Cerebral stroke prediction – imbalanced dataset," 2022, https://www.kaggle.com.

stroke_prediction (1)[1].docx

Turnitin

Document Details

Submission ID
trn:oid:::31142:106326650

Submission Date
Jul 29, 2025, 6:26 PM GMT+5

Download Date
Jul 29, 2025, 6:27 PM GMT+5

File Name
stroke_prediction (1)[1].docx

File Size
511.4 KB

6 Pages

3,971 Words

20,770 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.