



Automated Detection of Brain Tumor and Brain Stroke

Abdallah Shehata — *Anas Elgarhy*
Yasmine Osama — Youmna Mustafa
Khloud Tareq — Beshoy Nashat
Abeer Yousief — Asmaa Samir

Supervisors: **Dr. Wafaa Samy** & **TA. Hussein Mohamed**

Department of Information Technology
The Egyptian E-learning University

2024 - 2025



Automated Detection of Brain Tumor and Brain Stroke

Abstract

A brain tumor is a growth of cells in the brain or near it. Brain tumors can happen in the brain tissue. Brain tumors also can happen near the brain tissue. Nearby locations include nerves, the pituitary gland, the pineal gland, and the membranes that cover the surface of the brain.

Your skull, which encloses your brain, is very rigid. Any growth inside such a restricted space can cause problems. Brain tumors can be of two types i.e. cancerous (malignant) or non-cancerous (benign).

Brain cancer is a life-threatening disease, and it affects all the diagnosed people severely. Precise brain tumor classification helps to diagnose brain cancer early, which increases the survival rate of brain cancer patients, but it is quite hard to detect early. It is difficult to evaluate the magnetic resonance imaging images manually. Therefore, there is a need for optimized and fast digital methods for tumor diagnosis with better accuracy.

In this work, we are using a deep learning model based on convolutional neural networks to identify and classify brain cancer and its type from MRI images of patients using a publicly available dataset in Kaggle Hamada (2020).

For brain stroke, is a brain attack and it is a sudden interruption of continuous blood flow to the brain and a medical emergency. A stroke occurs when a blood vessel in the brain becomes blocked or narrowed, or when a blood vessel bursts and spills blood into the brain.

Stroke is a major contributor to death and disability worldwide. Prior to modern therapy, post-stroke mortality was approximately 10% in the acute period, with nearly one-half of the patients developing moderate-to-severe disability.

The most fundamental aspect of acute stroke management is “time is brain”. In acute ischemic stroke, the primary therapeutic goal of reperfusion therapy, including intravenous recombinant tissue plasminogen activator (IV TPA) and/or endovascular thrombectomy, is the rapid restoration of cerebral blood flow to the salvageable ischemic brain tissue at risk for cerebral infarction.

Several landmark endovascular thrombectomy trials were found to be of benefit in selecting patients with acute stroke caused by occlusion of

the proximal anterior circulation, which has led to a paradigm shift in the management of acute ischemic strokes.

In this modern era of acute stroke care, more patients will survive with varying degrees of disability post-stroke. A comprehensive stroke rehabilitation program is critical to optimize post-stroke outcomes. Understanding the natural history of stroke recovery, and adapting a multidisciplinary approach, will lead to improved chances for successful rehabilitation.

Here we are using a CNN model for deep learning multi-image classification which classifies input images into tumor, no tumor, stroke, or no stroke.

Contents

List of Figures	vi
Nomenclature	viii
1 Introduction	1
2 Acknowledgement	3
3 Overview	4
3.1 Scope	7
3.2 Objective	7
3.3 Need	7
3.4 Motivation	7
4 Literature Review	9
4.1 Fundamental Concepts	9
4.1.1 Data	9
4.1.2 Sensor Fusion	9
4.1.3 Machine Learning	10
4.1.4 Supervised Learning	10
4.1.5 Unsupervised Learning	11
4.1.6 Reinforcement Learning	11
4.1.7 Statistics and Probability in Machine Learning	12
4.1.8 Decision Making	12
4.1.9 Deep Learning	12
5 System Requirements Specifications	13
5.1 Introduction and Purpose	13
5.2 Work Scope	13

5.3	User Classes and Characteristics	13
5.3.1	Assumptions and Dependencies	14
5.3.1.1	Assumptions	14
5.4	Functional Requirements	14
5.4.1	First feature of The System (Functional Requirement)	14
5.4.2	System Feature (Functional Requirement)	14
5.5	Connectivity to the Outside World Requirements	15
5.5.1	User Interfaces	15
5.5.2	Hardware Interfaces	15
5.5.3	Software Interfaces	15
5.5.4	Communication Interfaces	15
5.6	Non-functional Requirements	16
5.6.1	Performance Requirements	16
5.6.2	Safety Requirements	16
5.6.3	Security Requirements	16
5.6.4	Software Quality Attributes	17
5.7	System Requirements	17
5.7.1	Requirements of Database	17
5.7.2	Software Requirements (Platform Choice)	17
5.7.3	Hardware Requirements	17
5.8	The SDLC Model	18
6	SYSTEM DESIGN	19
6.1	Architectural Diagram	19
6.2	Processing Steps	19
6.3	K-Means Clustering for Segmentation	20
6.4	Approximate Reasoning	20
6.5	UML Diagrams	20
6.5.1	Use Case Diagram	20
6.6	Sequence Diagram	21
6.7	Activity Diagram	22
6.8	Class Diagram	23
7	Implementation	24
7.1	Brain Tumor detection model	24
7.1.1	Introduction	24
7.1.1.1	Datasets details	24
7.1.2	Dataset Configuration Parameters	24
7.1.3	Dataset Preparation	25
7.1.3.1	Data Shuffling and Splitting	25

7.1.3.2	Edges Cropping	25
7.1.3.3	Cropping Methodology	26
7.1.4	Load and Resizing	27
7.1.4.1	Explanation	27
7.2	Model Building	27
7.2.1	Model parameters	27
7.2.2	The building function	28
7.2.3	Model Compilation	28
7.2.4	Callbacks	28
7.2.5	Summary	29
7.2.6	Training	29
7.2.6.1	Training Configuration	29
7.2.7	Loading and Splitting Data	29
7.2.8	Training Loop	29
7.2.9	Summary	30
7.3	Brain Stroke detection model	31
7.3.1	Introduction	31
7.3.2	Dataset Acquisition and Preparation	31
7.3.3	Directory Structure	31
7.3.3.1	Sorting Configuration	31
7.3.4	Sorting Configuration	32
7.3.5	Dataset Preparation	32
7.3.6	Loading the Dataset	33
7.3.7	Data Augmentation	33
7.3.8	Building the model	34
7.3.8.1	Model Parameters	34
7.3.8.2	Model Building Function	35
7.3.9	Model Training and Evaluation	37
7.3.9.1	Setting the Number of Epochs	37
7.3.9.2	Callback for Model Check pointing	37
7.3.9.3	Training Function	37
7.3.9.4	Conclusion	38
7.4	Overview of the Website	38
7.4.1	Tools, Technologies, and Techniques	38
7.4.1.1	GUI System	39
7.4.1.2	Technologies	40

Bibliography	50
---------------------	-----------

List of Figures

3.1	Human Brain Parts.	4
3.2	Left Brain & Right Brain.	5
3.3	Brain Tumor Example.	6
6.1	The System Architecture	19
6.2	Use case diagram Mohapatra and Rath (2020)	21
6.3	Activity Diagram	22
6.4	Class Diagram	23
7.1	Dataset configuration	25
7.2	Data shuffling	26
7.3	The load data function	27
7.4	The CNN model parameters	28
7.5	The model architecture	28
7.6	CNN model compilation	29
7.7	Setup the callbacks	29
7.8	Training parameters	30
7.9	Loading and Splitting the data	30
7.10	The Training Loop	30
7.11	Dataset Acquisition and Preparation	31
7.12	Data Directory Structure	32
7.13	The normal sorting configuration	32
7.14	The stroke sorting configuration	32
7.15	Count slices Function	33
7.16	Collect Scans Function	33
7.17	Setup the model checkpoint	37
7.18	The training Function	38
7.19	JS First code snippet	41

7.20 JS Secund code snippet	42
7.21 Used Python libraries	43
7.22 Detect and Predict Mask Function	44
7.23 Loop over detictions	45
7.24 Load the trained model	46
7.25 The main API entry point	47
7.26 Pre-process the user input	48
7.27 The prediction API entry point	49

Nomenclature

AI	Artificial Intelligence
CSF	Cerebrospinal Fluid
CT	Computerized Tomography
MRI	Magnetic Resonance Imaging
CNS	Central Nervous System
PNS	Peripheral Nervous System
CSF	Cerebrospinal Fluid
DIP	Digital Image Processing
ML	Machine Learning
CNN	Convolutional Neural Network
FEM	Finite Element Method
DCE	Dynamic Contrast Enhanced
IDE	Integrated Development Environment
UI	User Interface
UX	User Experience

Introduction

A brain tumor is a growth of cells in the brain or near it. Brain tumors can happen in the brain tissue. Brain tumors also can happen near the brain tissue. Nearby locations include nerves, the pituitary gland, the pineal gland, and the membranes that cover the surface of the brain. Your skull, which encloses your brain, is very rigid. Any growth inside such a restricted space can cause problems. Brain tumors can be of two types i.e. cancerous (malignant) or non-cancerous (benign). Brain cancer is a life-threatening disease, and it affects all the diagnosed people severely. Precise brain tumor classification helps to diagnose brain cancer early, which increases the survival rate of brain cancer patients, but it is quite hard to detect early. It is difficult to evaluate the magnetic resonance imaging images manually. Therefore, there is a need for optimized and fast digital methods for tumor diagnosis with better accuracy. In this work, we are using a deep learning model based on convolutional neural networks to identify and classify brain cancer and its type from MRI images of patients using publicly available datasets in Kaggle [†].

For brain stroke, it is a brain attack and it is a sudden interruption of continuous blood flow to the brain and a medical emergency. A stroke occurs when a blood vessel in the brain becomes blocked or narrowed, or when a blood vessel bursts and spills blood into the brain. Stroke is a major contributor to death and disability worldwide. Before modern therapy, post-stroke mortality was approximately 10nearly one-half of the patients develop moderate-to-severe disability. The most fundamental aspect of acute stroke management is “time is brain”. In acute ischemic stroke, the primary therapeutic goal of reperfusion therapy, including intravenous recombinant tissue plasminogen activator (IV TPA) and/or endovascular thrombectomy is the rapid restoration of cerebral blood flow to the salvageable ischemic brain tissue at risk for cerebral infarc-

[†]<https://www.kaggle.com>

tion. Several landmarks endovascular thrombectomy trials were found to be of benefit in selecting patients with acute stroke caused by occlusion of the proximal anterior circulation, which has led to a paradigm shift in the management of acute ischemic strokes.

In this modern era of acute stroke care, more patients will survive with varying degrees of disability post-stroke. A comprehensive stroke rehabilitation program is critical to optimize post-stroke outcomes. Understanding the natural history of stroke recovery, and adapting a multidisciplinary approach, will lead to improved chances for successful rehabilitation.

Here we are using a CNN model for deep learning multi-image classification which classifies input images into tumor, no tumor, stroke, or no stroke.

Acknowledgement

First and foremost, we thank God for His grace and success. If it were not for His success and generosity toward us, we would not have reached this stage, or achieved this success, and completed this work. we would like to express our heartfelt gratitude and appreciation to our supervisors Dr. Wafaa Sami and TA. Hussein Mohamad For their guidance, continuous support and encouragement For us all the time and for giving us a lot of invaluable experience, which helped us to continue and always try to provide the best and achieve this work and also extend our sincere thanks and appreciation to them for the time they invested in us and benefited us from their knowledge and experience.

We would also like to express our heartfelt, thanks to our families for their efforts They support and encourage us, bear our pressures, and believe in us, and they motivate us.

Once again, thank you for your dedication and commitment to our academic success, and for helping us achieve our goals. We will always be grateful for your role in shaping our future.

Overview

Brain tumor segmentation and identification are the focus of this research. To assess the anatomy of the brain, MRI or CT scans are frequently used. MRI scanned image is utilized throughout this investigation. The MRI scan is less intrusive than the CT scan for assessing a patient's health.

Magnetic fields and radio waves are at the core of this technology. Numerous algorithms have been developed to detect brain tumors. However, detection and extraction may be impeded.

This research includes two alternative segmentation approaches. Fuzzy C mean and K-means clustering techniques. For this reason, it offers an accurate tumor segmentation result. Any part of the body may become a tumor if its tissues develop out of control. Depending on its location, a tumor could be either primary or secondary 3.1. The word "primary" refers to everything that has a beginning. It is stated to be secondary if a piece of the tumor spreads and develops in a new place. CSF is generally altered by brain tumors (Cerebral Spinal Fluid) (Cerebral Spinal Fluid). Strokes are caused by it.

Instead of treating cancer, the doctor offers treatment for strokes. Con-

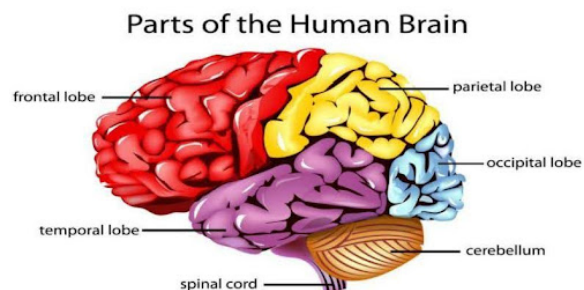


Figure 3.1: Human Brain Parts.

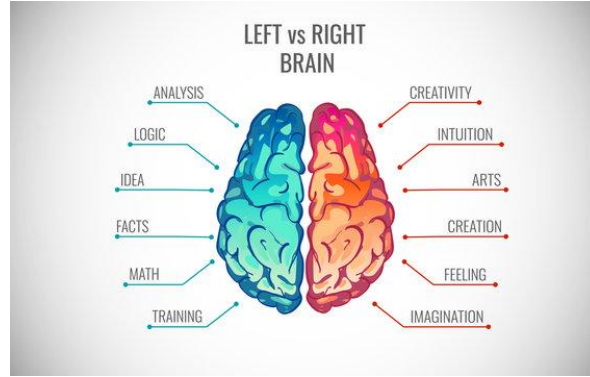


Figure 3.2: Left Brain & Right Brain.

sequently, early detection of malignancies is important to the success of this treatment. If found early enough, a brain tumor may be more efficiently treated, and the patient's life expectancy may be prolonged. That will increase the life of the gadget by about a year or two. T-cells and B-cells are the two most prevalent forms of cancer cells. Both the frequency and the severity of these tumors suggest their malignancy. Mass tumors make the detection of malignant tumors more a particular tumor location and detect malignancy in the brain using this technology. Treatment in most situations is focused on removing or eradicating the tumor 3.2.

If discovered and treated early, most brain tumors may be cured. The number of persons with brain tumors is growing at an alarming pace due to their incapacity to speak.

This project's major purpose is to enable people to learn to build tools. A tool that can tell people whether or not they are in danger of having a brain tumor, as well as how great of a risk they have. Java is the programming language utilized to develop the detecting system.

As a penultimate step, we are designing technologies that can assess the tumor's morphology and stage from a particular tumor location. The illustrations below illustrate the different parts of the brain. Two types of the nervous system are the central and peripheral nervous systems. Most of the nervous system consists of the brain and spinal cord (CNS). The spinal cord and skull are both vital nerves (PNS) 3.3 of the Peripheral Nervous System.

The spinal cord and the brain give rise to the spinal and cranial nerves, respectively. Previously, it described how the Peripheral Nervous Systems (PNS) regulate a variety of biological activities, including respiration, digestion, heart rate, and hormone secretion (PNS).

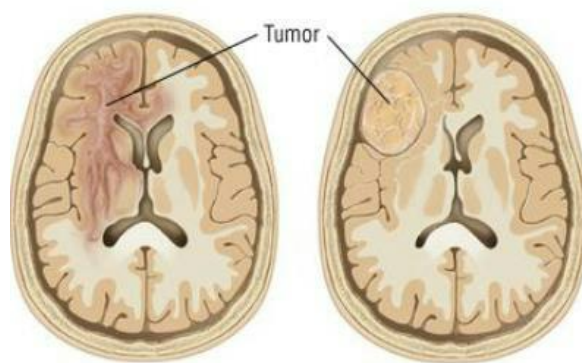


Figure 3.3: Brain Tumor Example.

It consists of the brainstem, cerebrum, and cerebellum as its three principal regions. The cerebrum and cerebellum are connected to the spinal cord via the brain stem. The cerebrum is the biggest area of the brain. It has a right as well as a left hemisphere. The cerebrum oversees all elements of a person's mental and emotional health, as well as their ability to communicate

The cerebellum is the brain's subsidiary structure. The cerebrum must coordinate the movement of the muscles. Many regions of the brainstem include the midbrain, Pons, and medulla. The brain stem is responsible for numerous autonomic functions of the body, such as waking and sleeping cycles, digestion, sneezing, heart rate, breathing, body temperature, coughing, vomiting, and swallowing. The folds on the surface of the cerebrum are referred to as the cortex.

Of the 100 billion nerve cells in the brain, the cortex contains around 70% of them. Grey matter, the component that gives the cortex its gray-brown tone, is composed of nerve cell bodies. As seen in 3.1, the white matter is composed of the long, linked fibers known as axons found in the cortex.

The corpus callosum links the brain's two hemispheres. It facilitates the flow of communication between the two parties. Each hemisphere of the brain is responsible for controlling a distinct side of the body. Weakness or paralysis of the left arm or leg could result from a tumor on the right side of the brain. The left hemisphere of the brain is responsible for thinking, mathematics, speaking, and writing. The right hemisphere oversaw imagination, three-dimensional ability, creativity, and musical aptitude. The left hemisphere dominates hand use and language in around 92% of individuals Ma et al. (2018).

3.1 Scope

Due to its soft tissue contrast and non-invasiveness, MR imaging is a popular option for significant medical imaging of the brain. MR imaging can also be employed to monitor how the size of a brain tumor varies as a result of treatment. A better method for separating tumors is required. Quantifying tumor volumes using MR images is currently impossible as a result of commonly accepted clinical practice procedures.

3.2 Objective

The objectives of this work can be summarized in the set of the following points:

- a) The primary goal is to create a framework that will assist clinical specialists in cross-confirming their predicted brain tumor type analysis results.
- b) Because the current diagnosis process is time-consuming, labor-intensive, and expensive, this deep learning-based tool can detect tumor growth and predict stage.
- c) Because this is an automated tool that uses image processing and AI, it reduces the amount of human effort required to predict the existence of tumor type from images.

3.3 Need

People who have had past brain tumors are more likely to get a new one. If you have two or more close relatives who have brain tumors, your risk of developing one is increased. Even though it is very unusual, members of the same family can be born with a disease that renders the brain more susceptible to growing a tumor. Around 5% of all brain tumors may have a genetic component.

3.4 Motivation

- a) It's obvious that a technique for accurately segmenting tumors would be valuable. Quantitating tumor regions using MR images is currently impossible due to a lack of well-acknowledged clinical practice methods.

- b) It is our major goal in this research to detect if there is a Brain tumor or not and classify the tumor into its types if the tumor is present, such as Glioma, Meningioma, and Pituitary.

Literature Review

To begin research, it is necessary to write a review. It's critical to have access to old publications on topics like space and application, both of which are associated with the kind of work we do. It's not difficult to determine the real demand, the drawbacks of previous labor, and the weight of work after conducting a thorough examination of the past. Using reference papers helps students comprehend the work, calculations, models, and previous research used, as well as pointing them in the right direction for future study. Deep learning and machine learning frameworks are reviewed in this section.

4.1 Fundamental Concepts

4.1.1 Data

Fusion is nothing but a combination of data from various sources; these sources can be homogeneous or heterogeneous. It provides the result in some sense better than the individual input. Input from various sources gives a more accurate result than depending on only a single input in the case of various automatic applications Beddad et al. (2019).

4.1.2 Sensor Fusion

Sensor fusion technology is widely used in automatic applications. The collection of separate data from various sensors is done in sensor fusion. This technology leads to accurate and reliable results than depending on the not-that-accurate result from a single sensor. This technology uses microcontrollers for this work.

Sensor technology is evolving day by day and similarly trying to act like an actual human being. Because of its precision and ability to perform well this technology is used in various applications such as climate monitoring, healthcare, automotive systems, smart mobile devices, industrial control, and oil exploration.

4.1.3 Machine Learning

Throughout the 1980s and 1990s, the use of information-driven Artificial Intelligence (AI) tactics in many building fields, such as voice and picture analysis, grew in popularity. As indexes of structured knowledge grow, using robots to extract relevant instances from the data may be a game-changer.

An inquiry requires a large quantity of information, and it's difficult to examine all the potential linkages and connections with distinct connections of information sources that might produce many outcomes and can be investigated in detail to get desired results.

Artificial intelligence (AI) and Machine learning (ML) are ideal to unlock the full potential of massive data. Artificial intelligence (AI) is fully integrated within machine learning to help systems learn and grow without needing to be directly programmed.

Machine learning occurs when a computer is given a large amount of data, and the computer uses that data to figure things out for itself. Artificial intelligence may benefit from it because of the improved method it gives for its construction. Automated decision-making is a key feature of machine learning, which is why it is so commonly utilized Masood et al. (2020).

4.1.4 Supervised Learning

- a) Data must be prepared for the train machine. This data is known as training data. The technique in which the training data is learned from supervised algorithms and gives some results depending on them was nominated as a supervised learning algorithm.
- b) Supervised learning is of two types; these types are regression and classification.
- c) Sometimes there is a risk of receiving some unrelated input from training data.

- d) Once the machine learns from supervised learning it keeps the record and whenever some data output is needed it gives that output using such previous work.
- e) The Kind of data sets needed to be used in machine learning must be decided first to consider how best-supervised learning works

4.1.5 Unsupervised Learning

- a) There is no requirement to supervise the model in unsupervised machine learning.
- b) If there are unknown patterns present in data then these unknown patterns are recognized by unsupervised machine learning.
- c) There are two types of unsupervised learning: clustering and second is association.
- d) In this unsupervised learning technique four types of clustering methods are used. These clustering methods are exclusive, agglomerative, overlapping, and probabilistic.
- e) There are large databases that are involved these databases have data objects in them. These data objects need association between themselves. To do this association rules are required.
- f) Whenever data is sorted in process it is not possible to get precise information is the drawback of this unsupervised learning.

4.1.6 Reinforcement Learning

- a) Sequence is very important in reinforcement learning. Simply output is dependent on the current input and for the input of the next stage; the output of the previous stage is required.
- b) Dependency is there in reinforcement learning so there is a need to assign labels to each dependent decision.
- c) This technique broadens its area of training by taking help from the environment instead of just depending on the data set provided so that it can increase the chances of better output.

4.1.7 Statistics and Probability in Machine Learning

Machine learning is mainly involved in building intelligent applications. To make intelligent application statistics and probability algorithms are used in terms of providing better vision by learning from data.

To predict upcoming events probability is used and to analyze the events in the past statistics is used.

4.1.8 Decision Making

Decision-making is very important as it provides one conclusive stage to the system. A decision tree has branches and leaves. Branches denote observations and leaves represent conclusions, so the overall decision tree is used as a predictive model. Data mining, statistics, and machine learning need accurate decision making so it uses this approach of predictive modeling. Every leaf of the tree has a class label, and branches connect the leaves in the tree. These branches tend to reach the class label of the leaves. Decision analysis needs a decision tree for decision-making. The decision tree illustrates the decisions that make it easy for decision-making Xia et al. (2017).

4.1.9 Deep Learning

In machine learning and artificial intelligence (AI), a technique known as "deep learning" mimics human learning processes. As part of data science, deep learning is a critical feature. Data scientists benefit greatly from deep learning. must gather, analyze, and interpret massive, large amounts of data; it speeds up and simplifies the process. Predictive analytics may be made more efficient by using deep learning. While most machine learning algorithms are linear, deep learning algorithms are stacked in a hierarchical framework of increasing complexity and abstraction Munappy et al. (2019).

System Requirements Specifications

5.1 Introduction and Purpose

The documents to be searched, as well as the software needed to search, are all listed in this document. To create the system, a list of the different software frameworks is necessary. The following topics are covered in the document.

Introduction and Purpose

5.2 Work Scope

Today's ailments are on the rise as a result of our hectic and stressful lifestyles. All age groups are susceptible to illness, thus early detection is critical. Pneumonia, lung cancer, and brain tumor diseases by using symptoms or reports. Most people throughout the world still lack access to the necessary instruments for the early identification of these illnesses. One of the most hazardous diseases among both genders, early diagnosis and treatment are critical to the health of those sick. For the future, we're going to invent a strong system that can accurately work on a large medical dataset with several lung cancer and brain tumor diseases.

5.3 User Classes and Characteristics

This system contains mainly 2 classes the User and the Admin, the characteristics of these classes can be listed as follows:

- a) Characteristics of User
 - i) Login/Register
 - ii) Upload images
 - iii) Get prediction results
 - iv) Logout
- b) Characteristics of Admin
 - i) Admin login with authentication
 - ii) Stage evaluation
 - iii) Prediction result

5.3.1 Assumptions and Dependencies

5.3.1.1 Assumptions

Pathological tests, such as needle biopsy specimens and analysis by experienced pathologists, are necessary for the accurate identification and classification of skin diseases because they require a human judgment of many factors, and experiences, and a system of decision support is eligible in this case. Doctors are unable to use the current system since it lacks a diagnostic system Abdel-Maksoud et al. (2015). Dependencies: Both structured data and unstructured data are used by the algorithm (from hospitals). To the best of our knowledge, there is an absence of studies on any data type in the field of big data analytics for medicine. Providing a correct diagnosis lowers the mortality rate caused by misdiagnosis.

5.4 Functional Requirements

5.4.1 First feature of The System (Functional Requirement)

- a) Images of lung cancer can be used to identify the condition.
- b) Image-based detection of brain tumors.

5.4.2 System Feature (Functional Requirement)

1. Database
2. Database Requirements SQLite3

5.5 Connectivity to the Outside World Requirements

5.5.1 User Interfaces

Python: The Python interface is currently being worked on. Many algorithms, as well as the functions that make them up or support them, may be found. With an STL container-friendly template interface, Open CV is developed entirely in C++.

Image Processing: Images may be read and written. Images and their characteristics are detected. In a picture, the detection of forms like circles, rectangles, and coins may be accomplished. Recognizing text in photographs is an emerging technology (e.g., taking a look at the plates). Changing the color and resolution of the picture.

5.5.2 Hardware Interfaces

1. To operate our project, we needed a hardware system such as an Intel I3 CPU with 4 GB of RAM and a 20GB hard drive.
2. Standard keyboard, mouse, and LED monitor are also required

5.5.3 Software Interfaces

Microsoft can be used as the operating system platform for the system. Some GUI tools are also employed by the system. PyCharm and higher are required as a Python platform to execute this application. An SQLite3 database is required for data storage.

5.5.4 Communication Interfaces

- a) Diseases Detection System
- b) User disease image data set
- c) Pre-processing unit
- d) Feature vector generation using CNN
- e) Classified results in the form of predictions

f) Open-CV for image processing

5.6 Non-functional Requirements

5.6.1 Performance Requirements

1. Performance: The performance of our system is fast as compared to other systems and response time is quick.
2. Availability: Availability of data is also a requirement for performing any operations.
3. Maintainability: In this system, we can maintain data of the user's images.
4. Security: In this system, user information is stored in the form of images, so our system is secure.
5. Usability: This system is very useful as an assistive tool for the medical sector.

5.6.2 Safety Requirements

This study is carried out to examine the economic impact that the system is going to have on the organization. The funding available to the company for system research and development is limited. Costs must be for clear reasons. As a result, the system was developed within the budget, which was accomplished because almost all the technologies used were free.

5.6.3 Security Requirements

The main thing in our system is, that we must provide end-to-end security for user and provider signs by using proper authentication login credentials. Users have been given the right to upload images and only view the results. The system is fully secure as well as eco-friendly. We implemented this system by considering security aspects, so we divided our system into four different modules to achieve integrity.

5.6.4 Software Quality Attributes

1. Capacity: The project capacity according to data is very small.
2. Availability: The proposed system is available on Python application.
3. Reliability: System is reliable for multi-disease prediction.
4. Security: When users log in to the system the user's mail ID and Password accurately match.

5.7 System Requirements

5.7.1 Requirements of Database

5.7.2 Software Requirements (Platform Choice)

The requirements for the software (platform choice) can be summarized as follows:

1. Operating System: Linux
2. Front End: Python3x
3. Database: SQLite3
4. IDE: PyCharm

5.7.3 Hardware Requirements

The requirements for the hardware can be summarized as follows:

- a) Processor: x64
- b) Speed: 2.5 GHz
- c) RAM: 8 GB (min)
- d) Hard Disk: 20GB (min)

5.8 The SDLC Model

Waterfall Model: The waterfall model is a consecutive model that is used in the software development processes, where the process slowly descends to be phase of requirements, gathering, analyses, design of systems, implementation, testing, Deploying, and finally maintenance Chen et al. (2017).

- a) Requirement analysis: Here requirements are gathered means which kind of dataset is required. Then what are the functional requirements of the system? The document is prepared, and then use cases are designed. In our system, we gather all information about the Admin and user and the functionality of each module.
- b) System Design: at this stage, the hardware (HW) and software (SW) required to design A system is decided. It uses the above-mentioned hardware and software requirements. We design the Admin and user modules. Design the according to the functionality of each module.
- c) Implementation: In this stage, the system is developed module-wise. This system consists of mainly 2 modules (1. Admin 2. User).
- d) Testing: In this stage, all developed software is installed, and it is tested in different ways against the system requirements. In this stage, we check whether all these modules are working properly or not with proper authentication. Disease prediction proper or not as well as stage prediction proper or not.
- e) Deployment: in this deployment stage we deployed the new functionality of each module like Crop Yields Predictions, Crop Suggestion to the Farmer; Dynamic Assistance, and Online E-Mart modules. We deploy all systems with proper functions.
- f) Maintenance: According to the software's new version and their use, they need to be updated some predefined machine learning libraries need to be used. This system is easy to maintain.

SYSTEM DESIGN

6.1 Architectural Diagram

In this chapter of the thesis, the proposed system design will be presented and discussed. The architecture of the proposed system design is illustrated in 6.1

6.2 Processing Steps

The first step is pre-processing which is the stage that converts the picture according to the requirements of the following level. It cleans up the picture by removing noise and other blemishes and sharpening the image's edges. RGB to grey conversion and reshaping are carried out through this stage also, it has a noise-removal feature that uses a median filter. Modern MRI scans have very low noise arrival probabilities. It may show up as a result of the heat effect. In this study, the primary focus is on identifying and classifying tumor cells. Noise reduction is required for the full system, though.

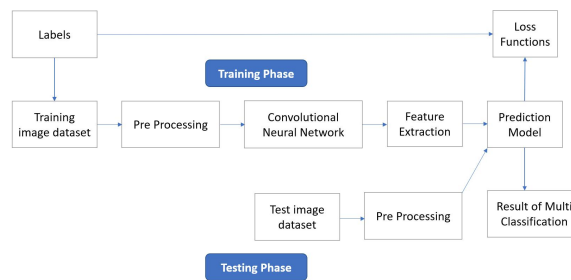


Figure 6.1: The System Architecture

6.3 K-Means Clustering for Segmentation

The steps for k-means clustering are illustrated below:

- a) K is the number of clusters.
- b) Pick the k cluster centers at random.
- c) Calculate the cluster's average or center.
- d) Determine the separation between the center of each cluster and each pixel.
- e) If the distance is near the center, go to the closest cluster to the center.
- f) Move on to the next cluster if you can't find what you need.
- g) Calculate the center of gravity again.
- h) Repeat the operation till the center doesn't shift.

6.4 Approximate Reasoning

To estimate the tumor size, the linearization approach is used in the first stage of deductive reasoning. In other words, it's a picture with just two possible colors: black or white (0 or 1). After this, define the tumor's stage and forecast its prognosis from the supplied tumor area.

6.5 UML Diagrams

6.5.1 Use Case Diagram

The use case diagram, as shown in 6.2, can be used to summarize the data about your system's users (also referred to as actors) and their interaction with it. A useful use case diagram can help your team represent and discuss 6.2:

- a) Representing the objectives of user-system interactions
- b) Defining and structuring a system's functional needs
- c) Defining a system's context and needs

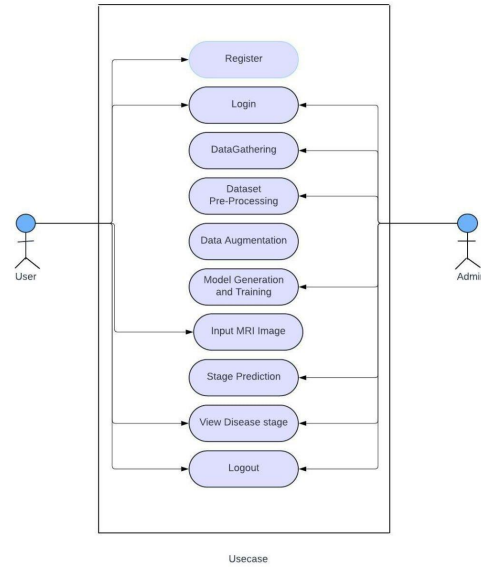


Figure 6.2: Use case diagram Mohapatra and Rath (2020)

d) Modeling the basic flow of events in a use case

The connection between a system and a user is graphically depicted in a use case diagram. So many user types and their interactions with a system could be depicted in a use case diagram. Internal and external factors are taken into account while drawing up use case diagrams. The majority of these criteria are design specifications. Usage cases are produced, and actors are identified when a system's capability is examined to prepare them for use. The following are some possible uses for use case diagrams:

- i) Gathers system requirements.
- ii) To acquire a different perspective on a system.
- iii) Identify the external and internal elements that affect the system.
- iv) Display the performers' interplay.

6.6 Sequence Diagram

A sequence diagram is an interaction diagram that shows how and in what order the processes interact. This is the construction of message sequence diagrams, sometimes called event diagrams, event scenarios, and sequence diagrams.

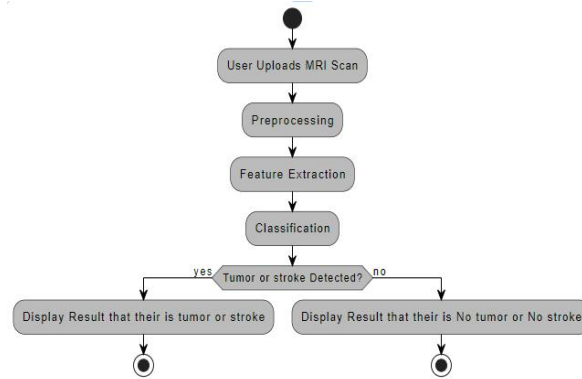


Figure 6.3: Activity Diagram

6.7 Activity Diagram

An activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration, and concurrency. An activity diagram shows the overall flow of control 6.3.

The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.

The order of events is shown by arrows that go from the beginning to the finish. As a result, they might be seen as a type of flowchart. It is difficult to express concurrency in flowcharts because of the absence of structures. This can only be resolved for basic circumstances by using the join and split symbols in activity diagrams; the model's meaning is obscured when they are arbitrarily coupled with choices or loops. Here we upload the image, whether it is of a brain tumor or stroke. It goes through the process of pre-processing, feature extraction, and then classification. Here, in the case of a brain tumor or stroke, the result appears as the presence of the disease, and in the case of the absence of the disease, the result appears as the absence of the disease.

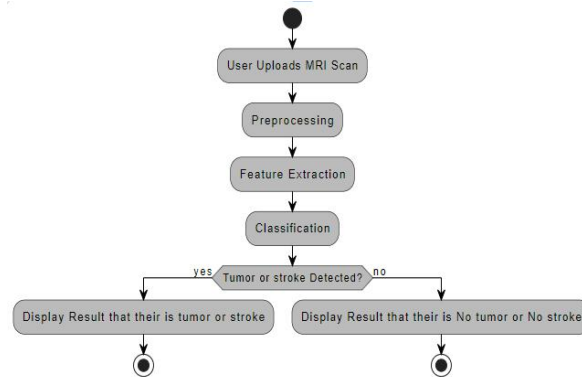


Figure 6.4: Class Diagram

6.8 Class Diagram

Static diagrams, such as the class diagram, are used in textbooks. It's a representation of the application's "static" state. It is possible to create executable code from a class diagram in addition to visualizing, explaining, and documenting many parts of a system. The class diagram is a graphic depiction of a class's attributes, capabilities, and constraints. Class diagrams are frequently used in the design of object-oriented systems as they are the only UML diagrams that may be immediately converted to object-oriented languages. In-class diagrams, relationships, classes, and interfaces may all be viewed. Also displayed are restrictions. Another term for it is a structure diagram. The class diagram serves as a visual representation of an application's static view. 6.4 shows a diagram showing how processes interact with each other and in what order they do so. As the name suggests, it's based on a Message Sequence Chart construct. The interactions between items are displayed chronologically in a sequence diagram. We can identify the objects and classes that are involved as well as the order of messages that need to be passed between them in this diagram to achieve the objectives of the scenario. Logical View use case realizations are generally depicted in sequence diagrams. Events and scenarios are sometimes used interchangeably with sequence diagrams.

Implementation

7.1 Brain Tumor detection model

7.1.1 Introduction

In this model, We utilize two primary datasets, Br35H Hamada (2020) and a dataset from Huggingface [3], which contain a variety of brain MRI images categorized into negative (no tumor) and positive (tumor) samples. The following sections provide a detailed explanation of the dataset selection, preprocessing, and configuration.

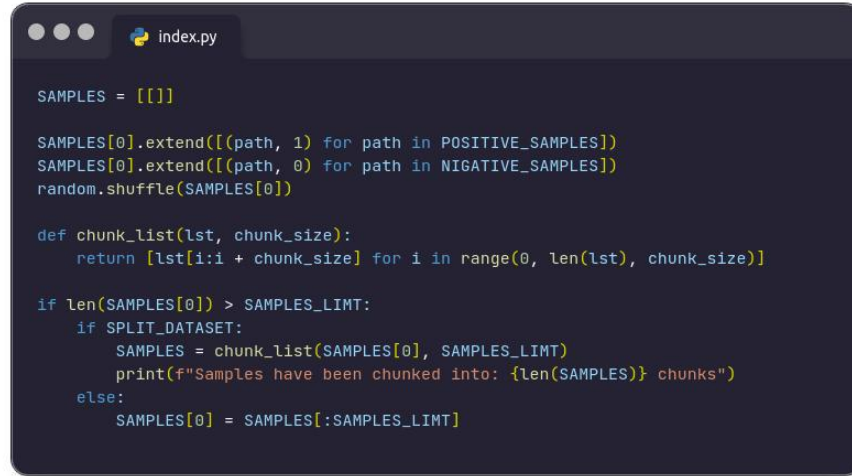
7.1.1.1 Datasets details

The datasets available for this project are:

1. Br35h Hamada (2020): This dataset consists of 1500 negative and 1500 positive samples.
2. Huggingface: This dataset comprises 1312 negative and 4057 positive samples.

7.1.2 Dataset Configuration Parameters

The training script allows the customization of various dataset parameters through configurable options. These parameters include the choice of the dataset, the limit on the number of samples, the option to split large datasets, the percentage of data used for testing, and whether to clean previous runs and augment samples 7.1.



```
SAMPLES = []

SAMPLES[0].extend([(path, 1) for path in POSITIVE_SAMPLES])
SAMPLES[0].extend([(path, 0) for path in NIGATIVE_SAMPLES])
random.shuffle(SAMPLES[0])

def chunk_list(lst, chunk_size):
    return [lst[i:i + chunk_size] for i in range(0, len(lst), chunk_size)]

if len(SAMPLES[0]) > SAMPLES_LIMIT:
    if SPLIT_DATASET:
        SAMPLES = chunk_list(SAMPLES[0], SAMPLES_LIMIT)
        print(f"Samples have been chunked into: {len(SAMPLES)} chunks")
    else:
        SAMPLES[0] = SAMPLES[0][:SAMPLES_LIMIT]
```

Figure 7.1: Dataset configuration

7.1.3 Dataset Preparation

The dataset preparation process includes downloading the chosen dataset, copying samples, and organizing them into positive and negative categories. Based on the dataset selected (Br35H or Huggingface), appropriate functions are called to fetch and prepare the data.

7.1.3.1 Data Shuffling and Splitting

The samples are shuffled, and if the total number of samples exceeds the limit, they are either split into chunks or truncated based on the configuration, to improve the learning performance 7.2.

This concludes the dataset settings and preparation chapter. The next steps involve utilizing these prepared datasets to train and evaluate our AI model for brain tumor detection.

7.1.3.2 Edges Cropping

To enhance the accuracy of brain tumor detection, it is essential to focus on the region of interest (ROI), which in this case is the brain itself. We implemented a cropping technique to isolate the brain from the rest of the image by finding the extreme points of the brain contour. This section details the cropping method, which is based on the approach described in [Finding extreme points in contours with OpenCV.

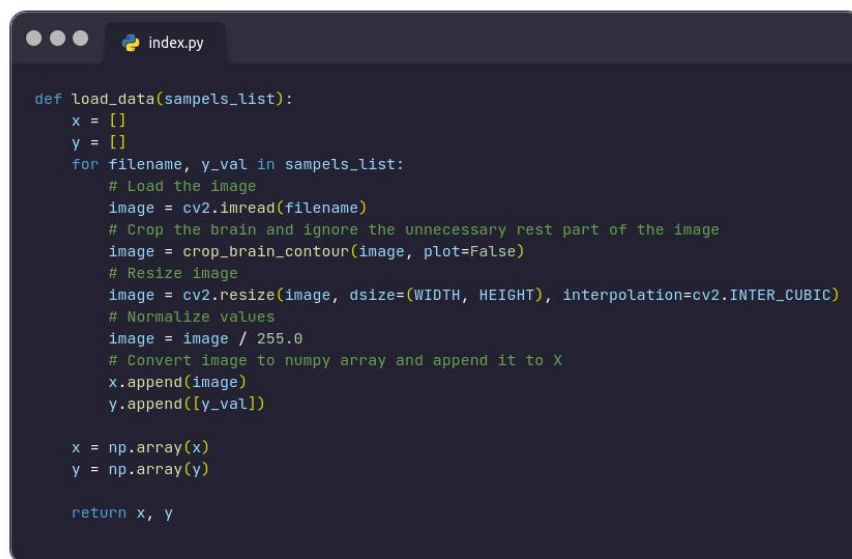


Figure 7.2: Data shuffling

7.1.3.3 Cropping Methodology

i) Convert to Grayscale and Blur:

- The input image is converted to grayscale to simplify the processing.
- A Gaussian blur is applied to the grayscale image to reduce noise and detail, which helps in thresholding.

ii) Thresholding:

- A binary threshold is applied to the blurred image, converting it into a binary image where the brain region becomes white (255) and the background becomes black (0).
- Erosion and dilation operations are performed to remove small noise and smooth the object boundaries.

iii) Finding Contours: Contours are identified in the binary image, and the largest contour is assumed to be the brain region.

iv) Extreme Points Calculation: The extreme left, right, top, and bottom points of the brain contour are identified.

v) Cropping the Image: Using the extreme points, the original image is cropped to extract the brain region.



Figure 7.3: The load data function

7.1.4 Load and Resizing

To feed the images into our AI model, it is crucial to prepare and resize them to a consistent shape. The ‘load_data’ function takes a list of image file paths and labels, preprocesses the images, and returns the data in a format suitable for training 7.3


7.1.4.1 Explanation

1. Loading the Image: Each image is loaded using ‘cv2.imread’.
2. Cropping the Brain Region: The function ‘crop brain contour’ is called to crop the brain region from the image.
3. Resizing the Image: The cropped image is resized to the specified ‘WIDTH’ and ‘HEIGHT’ using cubic interpolation.
4. Normalization: The pixel values of the image are normalized to the range [0, 1] by dividing by 255.0.
5. Appending to Lists: The processed image and its corresponding label are appended to lists ‘x’ and ‘y’, respectively.
6. Conversion to Numpy Arrays: The lists ‘x’ and ‘y’ are converted to NumPy arrays for efficient processing and handling in machine learning models.

7.2 Model Building

7.2.1 Model parameters

We can select the optimizer and set its parameters through the 7.4 code snippet.

A screenshot of a code editor window titled 'index.py'. It contains a Python function 'def build_model(input_shape):'. The function defines a Keras model architecture. It starts with an input placeholder 'X_input = Input(input_shape)'. Then, it applies 'ZeroPadding2D' with padding of 2 on all sides. This is followed by a 'CONV → BN → RELU Block' consisting of 'Conv2D(32, (7, 7), strides=(1, 1), name='conv0')(X)', 'BatchNormalization(axis=3, name='bn0')(X)', and 'Activation('relu')(X)'. Next, there are two 'MAXPOOL' layers: 'MaxPooling2D((4, 4), name='max_pool0')(X)' and 'MaxPooling2D((4, 4), name='max_pool1')(X)'. This is followed by a 'FLATTEN X' layer 'Flatten()(X)' and a 'FULLYCONNECTED' layer 'Dense(1, activation='sigmoid', name='fc')(X)'. Finally, it creates a 'Model' instance 'model = Model(inputs=X_input, outputs=X, name='BrainDetectionModel')' and returns it.

```
def build_model(input_shape):
    # Define the input placeholder as a tensor with shape input_shape.
    X_input = Input(input_shape)

    # Zero-Padding: pads the border of X_input with zeroes
    X = ZeroPadding2D((2, 2))(X_input) # shape=(?, 244, 244, 3)

    # CONV → BN → RELU Block applied to X
    X = Conv2D(32, (7, 7), strides=(1, 1), name='conv0')(X)
    X = BatchNormalization(axis=3, name='bn0')(X)
    X = Activation('relu')(X) # shape=(?, 238, 238, 32)

    # MAXPOOL
    X = MaxPooling2D((4, 4), name='max_pool0')(X) # shape=(?, 59, 59, 32)

    # MAXPOOL
    X = MaxPooling2D((4, 4), name='max_pool1')(X) # shape=(?, 14, 14, 32)

    # FLATTEN X
    X = Flatten()(X) # shape=(?, 6272)
    # FULLYCONNECTED
    X = Dense(1, activation='sigmoid', name='fc')(X) # shape=(?, 1)

    # Create model. This creates your Keras model instance, you'll use this instance to train/test the model.
    model = Model(inputs=X_input, outputs=X, name='BrainDetectionModel')

    return model
```

Figure 7.4: The CNN model parameters

A screenshot of a code editor window titled 'index.py'. It shows the setup for the model architecture. It defines 'WIDTH = 340' and 'HEIGHT = 340' as parameters. Then, it sets 'IMG_SHAPE = (WIDTH, HEIGHT, 3)'. It calls 'model = build_model(IMG_SHAPE)' and 'model.summary()'. Finally, it prints 'Model compiling...' and compiles the model with 'model.compile(optimizer=OPTIMIZER, loss='binary_crossentropy', metrics=['accuracy'])'.

```
WIDTH = 340 #@param {type:"number"}
HEIGHT = 340 #@param {type:"number"}

IMG_SHAPE = (WIDTH, HEIGHT, 3)
model = build_model(IMG_SHAPE)
model.summary()

print("Model compiling...")
model.compile(optimizer=OPTIMIZER, loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 7.5: The model architecture

7.2.2 The building function

The 'build model' function defines the architecture of the CNN, code snippet 7.5.

7.2.3 Model Compilation

After building the model, it needs to be compiled with the chosen optimizer, loss function, and metrics, code snippet 7.6.

7.2.4 Callbacks

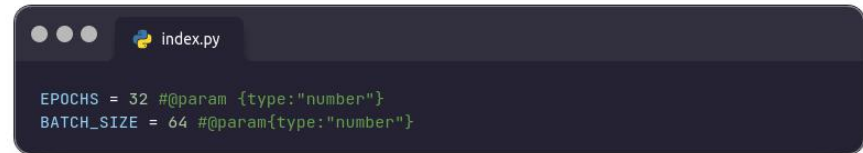
To monitor the training process and save the best model, we use TensorBoard and ModelCheckpoint callbacks, code snippet 7.7



```
# TensorBoard
log_file_name = f'brain_tumor_detection_cnn_{int(time.time())}'
tensorboard = TensorBoard(log_dir=f'logs/{log_file_name}')

# Checkpoint
# Unique file name that will include the epoch and the validation (development) accuracy
filepath = "cnn-parameters-improvement-{epoch:02d}"
# Save the model with the best validation (development) accuracy till now
checkpoint = ModelCheckpoint(f"models/{filepath}.model", monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

Figure 7.6: CNN model compilation



```
EPOCHS = 32 #@param {type:"number"}
BATCH_SIZE = 64 #@param {type:"number"}
```

Figure 7.7: Setup the callbacks

7.2.5 Summary

This section covers the crucial steps of building and compiling the CNN model for brain tumor detection. By allowing us to select the optimizer and its parameters, we ensure flexibility in training. Additionally, callbacks are set up to monitor and save the best-performing model during training. This setup ensures that the model is efficiently trained and can be evaluated using the validation dataset.

7.2.6 Training

7.2.6.1 Training Configuration

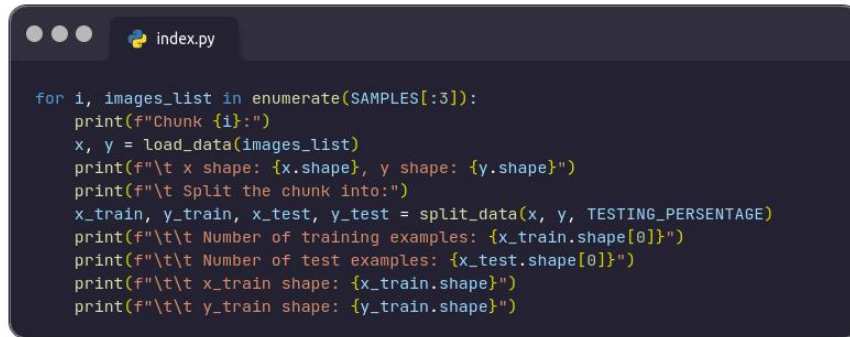
We define the hyperparameters for training, including the number of epochs and batch size, code snippet 7.8.

7.2.7 Loading and Splitting Data

We load and split the data for training and testing. This is done for each chunk of data to handle large datasets efficiently, code snippet 7.9.

7.2.8 Training Loop

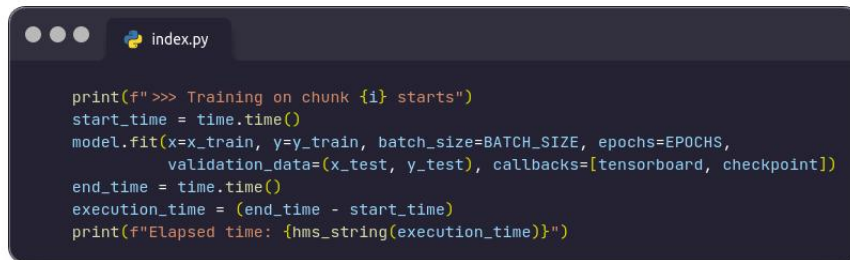
We iterate over each chunk of data, train the model, and measure the time taken for training, code snippet 7.10.



```
index.py

for i, images_list in enumerate(SAMPLES[:3]):
    print(f"Chunk {i}:")
    x, y = load_data(images_list)
    print(f"\t x shape: {x.shape}, y shape: {y.shape}")
    print(f"\t Split the chunk into:")
    x_train, y_train, x_test, y_test = split_data(x, y, TESTING_PERCENTAGE)
    print(f"\t\t Number of training examples: {x_train.shape[0]}")
    print(f"\t\t Number of test examples: {x_test.shape[0]}")
    print(f"\t\t x_train shape: {x_train.shape}")
    print(f"\t\t y_train shape: {y_train.shape}")
```

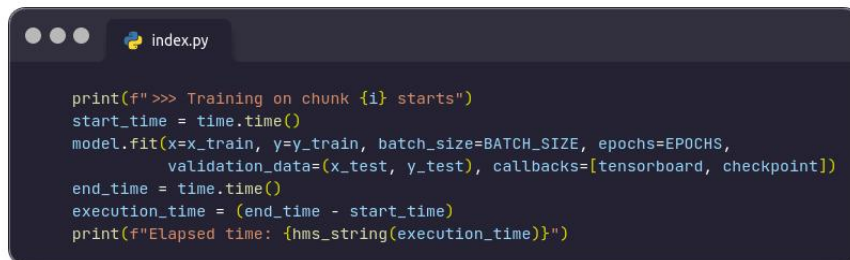
Figure 7.8: Training parameters



```
index.py

print(f">>> Training on chunk {i} starts")
start_time = time.time()
model.fit(x=x_train, y=y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
          validation_data=(x_test, y_test), callbacks=[tensorboard, checkpoint])
end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
```

Figure 7.9: Loading and Splitting the data



```
index.py

print(f">>> Training on chunk {i} starts")
start_time = time.time()
model.fit(x=x_train, y=y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
          validation_data=(x_test, y_test), callbacks=[tensorboard, checkpoint])
end_time = time.time()
execution_time = (end_time - start_time)
print(f"Elapsed time: {hms_string(execution_time)}")
```

Figure 7.10: The Training Loop

7.2.9 Summary

In this section, we detailed the process of training the CNN model for brain tumor detection. We covered data loading, splitting, model training, and performance monitoring.

By using callbacks, we ensured that the best model was saved, and the training process was logged for further analysis. The model's performance can be evaluated on the test set, and additional metrics such as the F1 score can be computed to assess its effectiveness.



```
# Download dataset
url = "https://github.com/Peco602/brain-stroke-detection-3d-cnn/releases/download/v0.0.1/brain_ct_data.zip"
filename = os.path.join(os.getcwd(), "brain_ct_data.zip")
tf.keras.utils.get_file(filename, url)

# Unzip dataset
with zipfile.ZipFile("brain_ct_data.zip", "r") as z_fp:
    z_fp.extractall(".")
```

Figure 7.11: Dataset Acquisition and Preparation

7.3 Brain Stroke detection model

7.3.1 Introduction

In this section, we present the initial setup for our Brain Stroke Detection model, detailing the steps taken to prepare and process the dataset. T

his setup includes downloading the dataset, unzipping the files, and organizing the data into appropriate directories for normal and stroke CT scans. Additionally, we introduce sorting configurations to ensure the CT scan slices are correctly ordered for analysis.

7.3.2 Dataset Acquisition and Preparation

To begin our model, we downloaded a dataset of brain CT scans from a public repository. This dataset is essential for training and evaluating our brain stroke detection model. The following code snippet illustrates the process of downloading and extracting the dataset, code snippet 7.11.

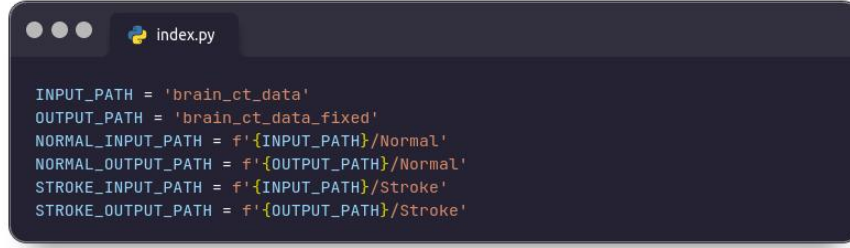
This script utilizes TensorFlow’s get file method to download the dataset from a specified URL. The dataset is then extracted into the current working directory using Python’s ‘zipfile’ module.

7.3.3 Directory Structure

Post extraction, the dataset is organized into separate directories for normal and stroke cases. The directory paths are defined as follows, code snippet 7.12.

7.3.3.1 Sorting Configuration

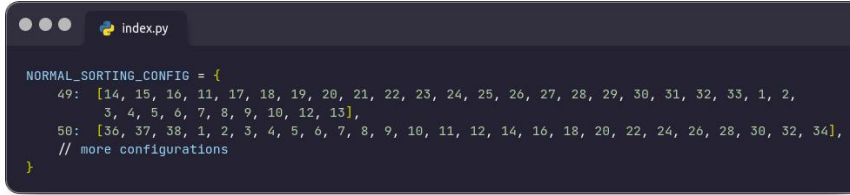
Properly sorting the CT scan slices is crucial for accurate model training. The slices need to be ordered correctly to preserve the spatial information of the



```
index.py

INPUT_PATH = 'brain_ct_data'
OUTPUT_PATH = 'brain_ct_data_fixed'
NORMAL_INPUT_PATH = f'{INPUT_PATH}/Normal'
NORMAL_OUTPUT_PATH = f'{OUTPUT_PATH}/Normal'
STROKE_INPUT_PATH = f'{INPUT_PATH}/Stroke'
STROKE_OUTPUT_PATH = f'{OUTPUT_PATH}/Stroke'
```

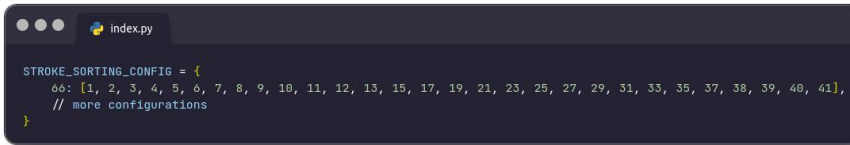
Figure 7.12: Data Directory Structure



```
index.py

NORMAL_SORTING_CONFIG = {
    49: [14, 15, 16, 11, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 1, 2,
        3, 4, 5, 6, 7, 8, 9, 10, 12, 13],
    50: [36, 37, 38, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34],
    // more configurations
}
```

Figure 7.13: The normal sorting configuration



```
index.py

STROKE_SORTING_CONFIG = {
    66: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 38, 39, 40, 41],
    // more configurations
}
```

Figure 7.14: The stroke sorting configuration

scans. We have implemented specific sorting configurations for both normal and stroke datasets. Below is an example of the sorting configuration for normal cases, code snippet 7.13.

7.3.4 Sorting Configuration

The ‘NORMAL SORTING CONFIG’ dictionary maps each patient identifier to a list of slice numbers in the correct order. Similarly, a sorting configuration is maintained for the stroke cases, code snippet 7.14.

Each configuration ensures that the slices for a given patient are processed in the correct sequence, which is vital for maintaining the integrity of the 3D structure of the brain scans.

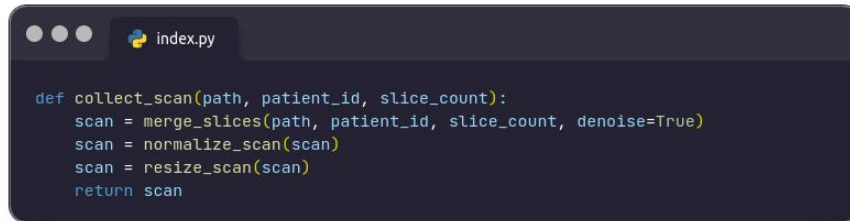
7.3.5 Dataset Preparation

We have the count slices to analyze the slice’s path and return a dictionary with the slice count associated with each patient, code snippet 7.15.

A screenshot of a code editor window titled 'index.py'. The code defines a function 'count_slices(path)' that iterates through a directory and its subdirectories using 'os.walk(path)'. For each file, it extracts the patient ID from the filename and increments a counter in a dictionary 'slice_dict' for that patient ID. The function returns the 'slice_dict' at the end.

```
def count_slices(path):
    slice_dict = {}
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            patient_id = int(filename.split()[0])
            if patient_id not in slice_dict:
                slice_dict[patient_id] = 1
            else:
                slice_dict[patient_id] = slice_dict[patient_id] + 1
    return slice_dict
```

Figure 7.15: Count slices Function

A screenshot of a code editor window titled 'index.py'. The code defines a function 'collect_scan(path, patient_id, slice_count)'. It calls 'merge_slices' with 'denoise=True', then 'normalize_scan', and finally 'resize_scan' before returning the 'scan' variable.

```
def collect_scan(path, patient_id, slice_count):
    scan = merge_slices(path, patient_id, slice_count, denoise=True)
    scan = normalize_scan(scan)
    scan = resize_scan(scan)
    return scan
```

Figure 7.16: Collect Scans Function

and collect scan to load the scan per patient ID and prepare it, code snippet 7.16.

7.3.6 Loading the Dataset

We define a function to load the entire dataset into memory as a 4D array. Then we'll label it as stroke (1) or normal (0) and split the dataset into training and validation sets.

7.3.7 Data Augmentation

We will use data augmentation techniques to increase the diversity of the training data without collecting new data. Augmentation helps improve the generalization capability of the model by artificially enlarging the dataset using random transformations.

The used techniques:

1. **Rotation:** Slightly rotates images to simulate different angles of view.
2. **Flipping:** Vertically flips images to increase variability.

3. **Shifting:** Translates images along the x and y axes, creating different viewpoints.
4. **Zooming:** Zooms in on images to simulate different focal lengths.
5. **Shear Layer:** Distorts the image by slanting its shape, which can simulate perspective distortions.
6. **Brightness Adjustment Layer:** Varies the brightness of the images, making them lighter or darker.
7. **Contrast Adjustment Layer:** Changes the contrast levels of the images, enhancing or reducing the difference between light and dark areas.

7.3.8 Building the model

This chapter details the architecture of the 3D Convolutional Neural Network (CNN) model, including the augmentation layers and the process of compiling the model with appropriate loss functions, optimizers, and performance metrics.

7.3.8.1 Model Parameters

In our model, we have defined several key parameters that govern the architecture and training of our Convolutional Neural Network (CNN). These parameters are crucial for understanding how the model processes input data and how it learns from the data during training. Below is a detailed explanation of each parameter:

1. Image Dimensions (WIDTH and HEIGHT):
 - WIDTH = 128
 - HEIGHT = 128

These parameters define the spatial dimensions of the input images fed into the CNN. Each image is resized or cropped to 128 pixels in width and 128 pixels in height. This standardization ensures that the input data is consistent in size, which is necessary for batch processing in the network. By using a fixed size, we also reduce computational complexity and memory usage.

2. Input Depth (DEPTH) = 64 This parameter specifies the number of channels in the input images. For typical RGB images, the depth would be 3 (corresponding to the red, green, and blue channels). However, in our project, the depth is set to 64, indicating that each input image consists of 64 feature channels. This could result from specific pre-processing steps or the use of specialized input data.
3. Initial Learning Rate (INITIAL LEARNING RATE) = 0.0001
The learning rate is a critical hyperparameter that determines the step size during the optimization process. An initial learning rate of 0.0001 means that during the initial stages of training, the model's weights will be adjusted by this factor concerning the loss gradient. A small learning rate helps in making gradual updates, which can lead to more stable convergence.
4. Learning Rate Decay Steps (DECAY STEPS) = 100000 The decay steps parameter specifies the number of training steps after which the learning rate is decayed. In our model, after every 100000 steps, the learning rate is updated. This mechanism helps in gradually reducing the learning rate, allowing the model to fine-tune the weights as it approaches convergence.
5. Learning Rate Decay Rate (DECAY RATE) = 0.96 The decay rate defines the factor by which the learning rate is multiplied after each decay step. A decay rate of 0.96 means that the learning rate will be reduced by 4% after every 100000 steps. This gradual reduction helps in maintaining a balance between convergence speed and the precision of the weight updates.

7.3.8.2 Model Building Function

This section describes the architecture and implementation details of our CNN model designed for [your specific task, e.g., image classification, medical imaging, etc.]. The model leverages 3D convolutional layers to effectively capture spatial features from the input data.

Model Parameters:

- Input Dimensions:
 - Width: 128 pixels

- Height: 128 pixels
- Depth: 64 channels
- Initial Learning Rate: 0.0001
- Learning Rate Decay: Steps: 100000 – Rate: 0.96

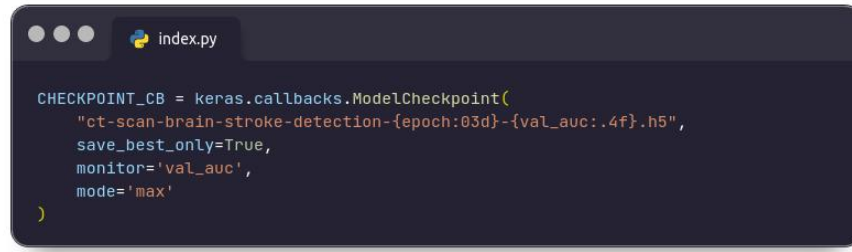
Model Architecture: Our model consists of several layers designed to extract and process features from the input data:

1. Input Layer: Accepts input images of shape (128, 128, 64).
2. Reshape Layer: Adds an extra dimension to make the input suitable for 3D convolutions.
3. Convolutional Layers: Four 3D convolutional layers with increasing filter sizes and ReLU activations.
4. Pooling Layers: 3D max pooling layers to reduce spatial dimensions.
5. Batch Normalization Layers: Normalize activations to improve training stability.
6. Global Average Pooling: Reduces each feature map to a single value.
7. Fully Connected Layer: Dense layer with 512 units and ReLU activation, followed by dropout for regularization.
8. Output Layer: Single unit with sigmoid activation for binary classification.

Learning Rate Schedule: The learning rate decreases exponentially based on the specified decay steps and rate, allowing the model to fine-tune its weights as training progresses.

Compilation: The model is compiled using the Adam optimizer with the defined learning rate schedule and binary cross-entropy loss. The performance is evaluated using specified metrics.

Conclusion: This model is designed to effectively learn from high-dimensional input data and perform accurate binary classification. Further details on training procedures, data preprocessing, and evaluation metrics are discussed in subsequent sections.



```
CHECKPOINT_CB = keras.callbacks.ModelCheckpoint(  
    "ct-scan-brain-stroke-detection-{epoch:03d}-{val_auc:.4f}.h5",  
    save_best_only=True,  
    monitor='val_auc',  
    mode='max'  
)
```

Figure 7.17: Setup the model checkpoint

7.3.9 Model Training and Evaluation

In this section, we cover the final step of the machine learning pipeline: training the model. This involves setting up callbacks for saving the best model, defining the training function, and training the model with different augmentation strategies. We aim to ensure the model learns effectively and generalizes well to unseen data.

7.3.9.1 Setting the Number of Epochs

The number of epochs determines how many times the model will iterate over the entire training dataset. Training for too few epochs might lead to underfitting, while too many epochs might cause overfitting. After careful consideration, we define the number of epochs as follows:

$$\text{EPOCHS} = 150$$

7.3.9.2 Callback for Model Check pointing

To ensure we save the best version of our model during training, we use a callback for model checkpointing. This callback monitors the validation AUC (Area Under the Curve) and saves the model weights whenever there is an improvement. This strategy helps in retaining the model that performs best on the validation set, preventing over-fitting and ensuring better generalization, code snippet 7.17.

7.3.9.3 Training Function

We define a function 7.18 to encapsulate the training process. This function takes the model, training dataset, validation dataset, number of epochs, and



```
def train_model(model, training_dataset, validation_dataset, epochs=EPOCHS, callbacks=[CHECKPOINT_CB]):  
    history = model.fit(  
        training_dataset,  
        validation_data=validation_dataset,  
        epochs=epochs,  
        shuffle=True,  
        verbose=1,  
        callbacks=callbacks  
    )  
    return history
```

Figure 7.18: The training Function

any callbacks as input parameters. It then trains the model, performs validation at the end of each epoch, and returns the training history for further analysis.

7.3.9.4 Conclusion

This chapter outlines the comprehensive steps involved in training and evaluating our CNN model. By setting up proper callbacks, defining a robust training function, and applying data augmentation techniques, we aim to develop a model that not only performs well on the training data but also generalizes effectively to new, unseen data. The next chapter will delve into the results of our training process and discuss the model's performance on the test dataset.

7.4 Overview of the Website

7.4.1 Tools, Technologies, and Techniques

There are numerous tools and techniques available to aid in the completion of tasks and the execution of processes.

Integrated Development Environment (IDE) in which the source code can be implemented, such as Android Studio IDE, which will be used to develop the system using the Java programming language, as well as to design the Graphical User Interface (GUI) allowing interaction processes among system components.

Adobe Photoshop (UI) for designing the app's user interface, buttons, and layout. Sketch (UX) to improve the user experience.

7.4.1.1 GUI System

A **GUI (Graphical User Interface)** is a system of interactive visual components for computer software. A GUI displays objects that convey information and represent actions that can be taken by the user. The objects change color, size, or visibility when the user interacts with them.

GUI overview: A GUI includes GUI objects, like icons, cursors, and buttons. These graphical elements are sometimes enhanced with sounds, or visual effects like transparency and drop shadows. Using these objects, a user can use the computer without having to know commands.

What are the elements of a GUI? To make a GUI as user-friendly as possible, there are different elements and objects that the user uses to interact with the software. Below is a list of each of these with a brief description.

- **Button:** A graphical representation of a button that acts as a program when pressed.
- **Dialog box:** A type of window that displays additional information and asks a user for input.
- **Icon:** Small graphical representation of a program, feature, or file.
- **Menu:** List of commands or choices offered to the user through the menu bar.
- **Menu bar:** Thin, horizontal bar containing the labels of menus.
- **Ribbon:** Replacement for the file menu and toolbar that groups programs' activities together.
- **Tab:** Clickable area at the top of a window that shows another page or area.
- **Toolbar:** Row of buttons, often near the top of an application window that controls software functions.
- **Window:** Rectangular section of the computer's display that shows the program currently being used.

How does a GUI work? A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files. Although a GUI operating system is primarily navigated using a mouse, a keyboard can also be used via keyboard shortcuts or arrow keys.

For example, if you want to open a program on a GUI system, you would move the mouse pointer to the program's icon and double-click it. With a command line interface, you need to know the commands to navigate to the directory containing the program, list the files, and then run the file.

What are the benefits of GUI? A GUI is more user-friendly than a text-based command-line interface, such as MS-DOS, or the shell of Unix-like operating systems.

Unlike a command-line operating system or CUI, like Unix or MS-DOS, GUI operating systems are easier to learn and use because commands do not need to be memorized. Additionally, users do not need to know any programming languages. Because of their ease of use and more modern appearance, GUI operating systems have come to dominate today's market.

7.4.1.2 Technologies

Front End:

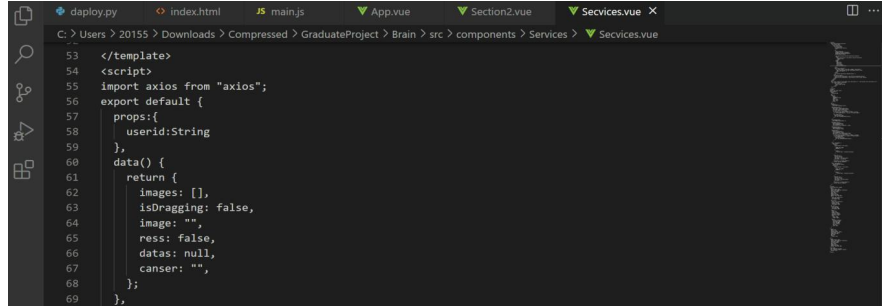
- **HTML**

HTML is essential for creating the structure of web pages. By using tags and attributes, you can create headings, paragraphs, links, images, and lists. It forms the foundation of web development, often used with CSS for styling and JavaScript for interactivity.

- **CSS**

CSS is used to style HTML documents by defining rules that specify how elements should look. It enhances the visual presentation of web pages, allowing for the separation of content (HTML) from design (CSS). This separation makes it easier to maintain and update web pages.

The framework used is Bootstrap, a versatile and powerful front-end framework that simplifies the process of designing responsive and modern web pages. Its grid system, pre-designed components, and utility classes allow developers to create attractive layouts quickly, while its JavaScript plugins add interactivity and enhance the user experience.



```
53 </template>
54 <script>
55 import axios from "axios";
56 export default {
57   props: {
58     userid: String
59   },
60   data() {
61     return {
62       images: [],
63       isDragging: false,
64       image: "",
65       res: false,
66       datas: null,
67       cancer: "",
68     };
69   },
70 }
```

Figure 7.19: JS First code snippet

- **JavaScript**

JavaScript is a versatile language essential for adding interactivity to web pages. It works alongside HTML and CSS to create dynamic user experiences, handle events, manipulate the DOM, and perform various client-side tasks. It allows developers to enhance the user experience by making web pages more engaging and responsive.

Vue.js is a flexible and powerful framework for building interactive web interfaces. Its reactivity system, component-based architecture, and support for single-file components make it a popular choice for developers looking to create maintainable and efficient applications.

JAVASCRIPT code with explanation: As you see in 7.19 and 7.20, we did:

- **import axios from "axios";**: This line imports the Axios library, which is a popular JavaScript library used to make HTTP requests.
- **export default ...** : This is the default export of the Vue component. Everything inside the curly braces defines the properties and behavior of the component.
- **props**: This is an object where we define the properties that the component accepts from its parent component.
- **userid: String**: This line defines a prop named user-id that is expected to be a string. This prop allows the parent component to pass a user-id value to this component.
- **data()**: This is a function that returns an object. This object contains the reactive data properties of the component.

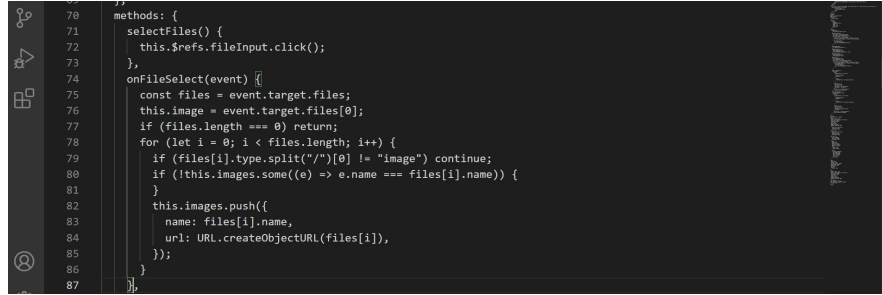


Figure 7.20: JS Secund code snippet

- **images:** `[]`: An empty array to hold images.
- **isDragging** `false`: A boolean flag that can be used to indicate if an item is being dragged (likely used in drag-and-drop functionality).
- **image:** `""`: A string to hold the URL or data of a single image.
- **ress:** `false`: A boolean flag, its purpose isn't clear from this snippet alone but it's likely used to indicate some sort of state.
- **datas:** `null`: A variable to hold some data, initially set to null.
- **canser:** `""`: A string variable, possibly meant to be canceled or something similar, but it's unclear without further context.

Purpose : This method programmatically triggers a click event on a file input element. Usage: It simulates a user clicking on a file input element to open the file picker dialog.

Back End:

- **Java** Java is a widely used, object-oriented programming language designed for building platform-independent applications. Its object-oriented nature, platform independence, and extensive libraries make it a popular choice for developers in various domains, from web and mobile development to enterprise systems.

The framework used is Spring Boot, a powerful framework for building Spring-based applications with minimal setup and configuration. Its features like embedded servers, dependency management, and production-ready tools make it a popular choice for developing scalable and robust applications quickly.

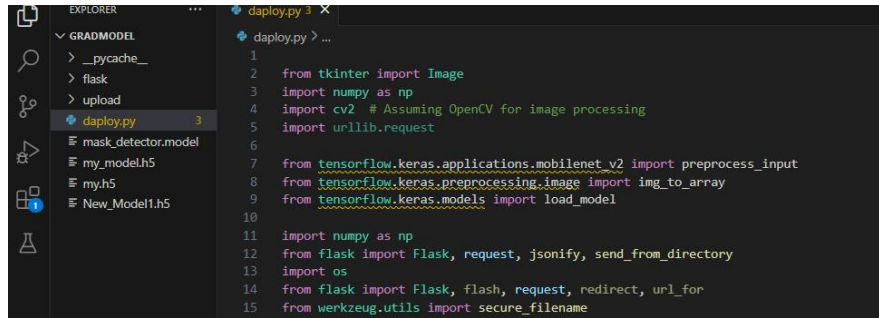


Figure 7.21: Used Python libraries

- **Python** Python is a versatile and powerful language known for its simplicity and readability. It is widely used across various domains due to its extensive libraries, ease of use, and support for multiple programming paradigms.

The framework used is Flask, a versatile and easy-to-use web framework for Python that allows for quick development of web applications, designed for building web applications quickly and with minimal setup. Its minimalist approach provides flexibility, while its support for extensions ensures that additional features can be easily integrated when needed.

- **Database (MySQL)**

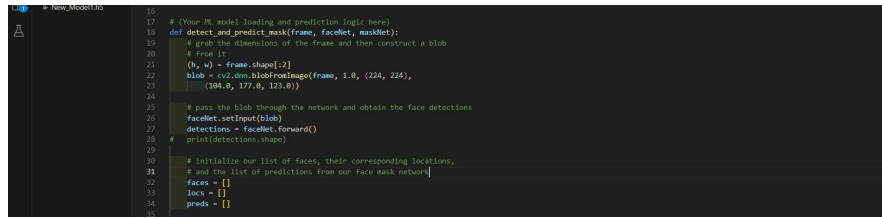
MySQL is a popular, open-source relational database management system known for its high performance and reliability. It is widely used in web development due to its ease of integration with various programming languages and its strong support for handling large datasets efficiently.

Code explanation: We used several Python libraries to load and run the trained models, shown in 7.21, 7.22, 7.23, 7.24, 7.25, 7.26 and 7.27.

Which contains:

tkinter (Image): Although Image is not typically imported directly from tkinter, this library is generally used for creating graphical user interfaces (GUIs). You might be referring to 'PIL.Image' from the Pillow library for image processing.

NumPy (np): A fundamental library for scientific computing in Python is used here for handling arrays and matrices.



```
16
17 # (Your ML model loading and prediction logic here)
18 def detect_and_predict_mask(frame, faceNet, maskNet):
19     # grab the dimensions of the frame and then construct a blob
20     # from it
21     (h, w) = frame.shape[:2]
22     blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
23                                 (104.0, 177.0, 123.0))
24
25     # pass the blob through the network and obtain the face detections
26     faceNet.setInput(blob)
27     detections = faceNet.forward()
28     # print(detections.shape)
29
30     # initialize our list of faces, their corresponding locations,
31     # and the list of predictions from our face mask network
32     faces = []
33     locs = []
34     preds = []
```

Figure 7.22: Detect and Predict Mask Function

cv2 (OpenCV): An open-source computer vision library. It provides tools for image processing and computer vision tasks.

urllib.request: A module for opening and reading URLs. Often used to fetch data from the web.

‘tensorflow.keras.applications.mobilenet’ v2 (preprocess input): Part of TensorFlow’s Keras module, this provides functions to preprocess input data for the MobileNetV2 model.

‘tensorflow.keras.preprocessing.image’ (img to array): Utility for converting a PIL Image instance to a Numpy array.

tensorflow.keras.models (load model): Provides functions to load pre-trained Keras models.

Flask: A lightweight WSGI web application framework in Python, used for creating web applications and APIs. **os:** A module providing a way of using operating system-dependent functionality like reading or writing to the file system.

werkzeug.utils (secure filename): A utility for securing a filename before storing it directly on the filesystem.

- **frame:** The input image frame where face masks need to be detected.
- **faceNet:** The pre-trained deep learning model used for face detection.
- **maskNet:** The pre-trained deep learning model used for face mask classification.

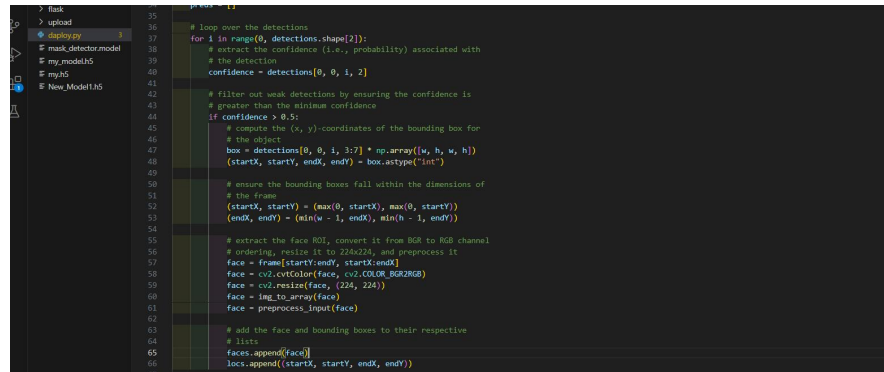


Figure 7.23: Loop over detections

- It extracts the height and width of the input frame.
- **blobFromImage** function of OpenCV's CNN module is used to pre-process the frame:
 - It resizes the image to (224, 224) pixels.
 - Normalizes the pixel intensities.
 - Converts the image to a blob format suitable for input to a neural network.
- The pre-trained faceNet model is used to detect faces in the image.
- The preprocessed blob is set as the input to the faceNet model.
- The forward method is called to perform forward pass inference, obtaining face detections.
- Iterates over the detections found by the face detection model.
- Extracts the confidence (probability) associated with the detection.
- Filters out weak detections by ensuring the confidence is greater than a minimum threshold (here, 0.5).
- Computes the (x, y)-coordinates of the bounding box for the detected face. The bounding box coordinates are relative to the dimensions of the input frame.
- Ensures that the bounding boxes fall within the dimensions of the frame to prevent errors.
- Extracts the face region from the frame.

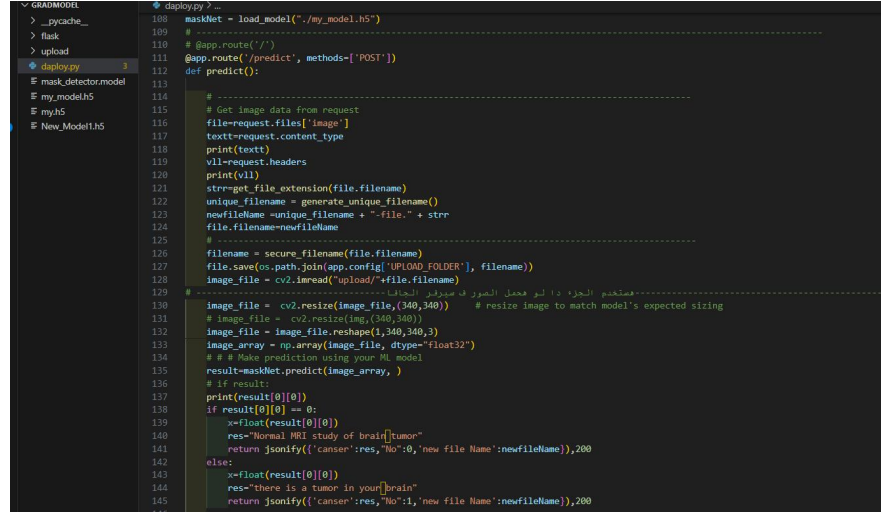
```

69 if len(faces) > 0:
70     # for faster inference we'll make batch predictions on all
71     # faces at the same time rather than one-by-one predictions
72     # in the above for loop
73     faces = np.array(faces, dtype="float32")
74     preds = maskNet.predict(faces, batch_size=32)
75
76     # return a 2-tuple of the face locations and their corresponding
77     # locations
78     return (locs, preds)
79
80
81
82 UPLOAD_FOLDER = './upload'
83 ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])
84
85 app = Flask(__name__)
86 # -----
87 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
88 # -----
89
90 def get_file_extension(file_name):
91     if file_name is None:
92         return None
93     file_name_parts = file_name.split('.')
94     return file_name_parts[-1]
95
96 # -----
97 import uuid
98
99 def generate_unique_filename():
100     return str(uuid.uuid4().hex)
101 # -----
102
103 # load our serialized face detector model from disk
104 prototxtPath = r"face_detector\deploy.prototxt"
105 weightsPath = r"face_detector\res10_384_384_ssd_iter_140000.caffemodel"
106 # facenet = cv2.dnn.readNet(prototxtPath, weightsPath)
107 # -----
108 # load the face mask detector model from disk
109 maskNet = load_model("./my_model.h5")
110

```

Figure 7.24: Load the trained model

- Converts the color space of the face region from BGR to RGB.
- Resizes the face region to the required input size (224x224) for the mask classification model.
- Converts the face region to a NumPy array and preprocesses it for the model.
- Adds the preprocessed face and its bounding box coordinates to their respective lists for further processing.
- If there are detected faces, it converts the list of faces into a NumPy array.
- It then uses the prediction method of the mask detection model (maskNet) to predict whether each face is wearing a mask or not. This is done in batch mode for faster processing.
- Returns a tuple containing the locations of the detected faces and their corresponding predictions regarding whether they are wearing masks or not.
- Set up configurations for handling uploaded files, including the allowed file extensions and the upload folder.
- Defines a utility function to extract the file extension from a given file name.



```

108 maskNet = load_model("./my_model.h5")
109
110 # -----
111 # @app.route('/')
112 @app.route('/predict', methods=['POST'])
113 def predict():
114     # -----
115     # Get image data from request
116     file=request.files['image']
117     text=request.content_type
118     print(text)
119     vll=request.headers
120     print(vll)
121     str=get_file_extension(file.filename)
122     unique_filename = generate_unique_filename()
123     newfileName =unique_filename + "-file." + str
124     file.filename=newfileName
125     # -----
126     filename = secure_filename(file.filename)
127     file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
128     image_file = cv2.imread("upload"+file.filename)
129     # -----
130     image_file = cv2.resize(image_file,(340,340)) # resize image to match model's expected sizing
131     image_file = cv2.cvtColor(image_file, cv2.COLOR_BGR2RGB)
132     image_file = image_file.reshape(1,340,340,3)
133     image_array = np.array(image_file, dtype="float32")
134     # # Make prediction using your ML model
135     result=maskNet.predict(image_array, )
136     # if result:
137     print(result[0][0])
138     if result[0][0] == 0:
139         res="Normal MRI study of brain"
140         return jsonify({'cancer':res,"No":0,"new file Name":newfileName}),200
141     else:
142         res="there is a tumor in your brain"
143         return jsonify({'cancer':res,"No":1,"new file Name":newfileName}),200
144
145

```

Figure 7.25: The main API entry point

- Defines a utility function to generate a unique filename using the UUID module.
- Loads the face detection model (prototxtPath and weightsPath) and the face mask detection model (maskNet) from disk.
- The face detection model seems to be based on the Caffe framework, and the face mask detection model is loaded using Keras's load model function.
- Retrieves the image file from the request.
- Collects information about the content type and headers of the request.
- Generates a unique filename for the uploaded image.
- Saves the uploaded image with the generated filename in the upload folder specified in the application's configuration.
- Reads the uploaded image using OpenCV (cv2).
- Resizes the image to match the expected input size of the model.
- Prepares the image array for prediction.
- Uses the trained machine learning model (maskNet) to predict the class of the input image.


```
164 # model2
165 import tensorflow as tf
166 import numpy as np
167 from PIL import Image
168 # from google.colab import files
169 import cv2
170 model = tf.keras.models.load_model('./New_Model1.h5')
171
172
173 #Function to preprocess the image
174 def preprocess_image(image_path):
175     img = Image.open(image_path).resize((140, 140))
176     |
177     # Convert PIL Image to NumPy array
178     img = np.array(img)
179
180     # Convert the image to grayscale using OpenCV
181     img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
182
183     # Normalize pixel values
184     img_gray = img_gray / 255.0
185
186     # Add batch dimension
187     img_gray = np.expand_dims(img_gray, axis=0)
188
189     return img_gray
190
191 def predict_stroke(image_path, newfilename):
192     img = preprocess_image(image_path)
193     prediction = model.predict(img)
194     if prediction[0] > 0.5:
195         # x=float(result[0][0])
196         res="There is brain stroke in you brain"
197         return {'cancer':res,"No":1,"new file Name":newfilename}
198         # return "The image contains a stroke."
199     else:
200         res="Normal CT Study of brain"
201         return {'cancer':res,"No":0,"new file Name":newfilename}
202         # return "The image does not contain a stroke."
203
204
```

Figure 7.26: Pre-process the user input

- Based on the prediction result, it returns a JSON response indicating whether the image depicts a normal MRI study or if there's a tumor in the brain. It also includes the new filename generated for the uploaded image.
- TensorFlow (tf) for loading the machine learning model.
- NumPy (np) for numerical operations.
- PIL (Image) for image manipulation.
- OpenCV (cv2) for image processing.
- Loads a trained Keras model from the specified file (New Model1.h5).
- Resizes the image to (140, 140) pixels.
- Converts the image to grayscale using OpenCV.
- Normalizes pixel values to the range [0, 1].
- Adds a batch dimension to the image.
- Preprocesses the input image using the preprocess image function.
- Uses the loaded model to predict whether the image contains a stroke.
- Returns a dictionary indicating the prediction result along with the new filename.

```
204 @app.route('/predict2', methods=['POST'])
205 def predict2():
206     # -----
207     # Get image data from request
208     file=request.files['image']
209     text=request.content_type
210     print(text)
211     vll=request.headers
212     print(vll)
213     str=get_file_extension(file.filename)
214     unique_filename = generate_unique_filename()
215     newfilename =unique_filename + "-file." + str
216     file.filename=newfilename
217     # -----
218     filename = secure_filename(file.filename)
219     file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
220     # -----
221     result = predict_stroke("upload/"+file.filename,newfilename)
222     return jsonify(result),200
223     # -----
224     # if result:
225
226
227
228 @app.route('/upload/<name>')
229 def display_file(name):
230     return send_from_directory(app.config["UPLOAD_FOLDER"], name)
231
232
233 if __name__ == '__main__':
234     app.debug=True
235     app.run(host='0.0.0.0', port=5000) # Listen on all interfaces, port 5000
236
```

Figure 7.27: The prediction API entry point

- Checks if the model prediction is greater than 0.5.
- If the prediction is greater, it suggests that the image contains a stroke; otherwise, it suggests the image is normal.
- Returns a dictionary containing the prediction result (res), a flag (No) indicating the presence of stroke (1) or absence (0), and the new filename for the uploaded image.
- Defines a new route /predict2 that listens for POST requests.
- Retrieves the image file from the request.
- Collects information about the content type and headers of the request.
- Generates a unique filename for the uploaded image.
- Saves the uploaded image with the generated filename in the upload folder specified in the application's configuration.
- Calls the predict stroke function to predict whether the image contains a brain stroke.
- Returns the prediction result as a JSON response along with an HTTP status code indicating success.
- Defines a route /upload/<name> to display uploaded files.
- The function display file retrieves the file with the given filename from the upload folder and sends it to the client.
- Runs the Flask application on the specified host and port (0.0.0.0:5000).

Bibliography

- E. Abdel-Maksoud, M. Elmogy, and R. Al-Awadi. Brain tumor segmentation based on a hybrid clustering technique. *Egyptian Informatics Journal*, 16(1):71–81, 2015. ISSN 1110-8665. doi: <https://doi.org/10.1016/j.eij.2015.01.003>. URL <https://www.sciencedirect.com/science/article/pii/S1110866515000043>.
- B. Beddad, K. Hachemi, and S. Vaidyanathan. Design and implementation of a new cooperative approach to brain tumour identification from mri images. *International Journal of Computer Applications in Technology*, 59(1):1–10, 2019. doi: 10.1504/IJCAT.2019.097113. URL <https://www.inderscienceonline.com/doi/abs/10.1504/IJCAT.2019.097113>.
- M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access*, 5:8869–8879, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2694446.
- A. Hamada. Br35h :: Brain tumor detection 2020. Kaggle, 2020. URL <https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection>.
- C. Ma, G. Luo, and K. Wang. Concatenated and connected random forests with multiscale patch driven active contour model for automated brain tumor segmentation of mr images. *IEEE Transactions on Medical Imaging*, 37(8):1943–1954, 2018. doi: 10.1109/TMI.2018.2805821.
- A. Masood, B. Sheng, P. Yang, P. Li, H. Li, J. Kim, and D. D. Feng. Automated decision support system for lung cancer detection and classification via enhanced rfcn with multilayer fusion rnn. *IEEE Transactions on Industrial Informatics*, 16(12):7791–7801, Dec 2020. ISSN 1941-0050. doi: 10.1109/TII.2020.2972918.

- H. Mohapatra and A. Rath. *Fundamentals of Software Engineering: Designed to provide an insight into the software engineering concepts*. 01 2020. ISBN 9388511778, 9789388511773.
- A. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne. Data management challenges for deep learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 140–147, Aug 2019. doi: 10.1109/SEAA.2019.00030.
- Y. Xia, C. Liu, Y. Li, and N. Liu. A boosted decision tree approach using bayesian hyper-parameter optimization for credit scoring. *Expert Systems with Applications*, 78:225–241, 2017. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2017.02.017>. URL <https://www.sciencedirect.com/science/article/pii/S0957417417301008>.