INDUSTRIAL ORIENTED MINI PROJECT

Report

On

IDENTIFYING DISEASE FOR LIVER CT-SCAN IMAGES

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Sudharshan Reddy- 22261A1244

Under the guidance of

Mrs. A.V.L. Prasuna

Assistant Professor, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **Identifying Disease for Liver CT-Scan Images** submitted by **Sudharshan Reddy (22261A1244)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bonafide work carried out under the supervision of **Mrs. A.V.L. Prasuna**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Mrs. A.V.L. Prasuna	**Dr. U. Chaitanya**
Assistant Professor	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR	**Dr. D. Vijaya Lakshmi**
	Professor and HOD
	Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **Identifying Disease for Liver CT-Scan Images** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs. A.V.L. Prasuna, Assistant Professor**, Department of IT, MGIT. No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Sudharshan Reddy- 22261A1244

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our internal supervisor **Mrs. A.V.L. Prasuna, Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi,** Assistant Professor, Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Sudharshan Reddy -22261A1244

ABSTRACT

The early detection of liver-related diseases such as fatty liver, cirrhosis, and hepatocellular carcinoma is critical for improving patient outcomes and reducing the long-term burden on healthcare systems. However, interpreting liver radiology scans accurately and promptly remains a challenge for many medical professionals due to the complexity of the images and the subtle progression of liver conditions. This project aims to develop a deep learning-powered Liver Disease Prediction System that assists radiologists in identifying potential liver abnormalities from radiological images using advanced image classification techniques.

The system leverages Visual Transformers (VITs), specifically the VGG16 architecture, to analyse liver scan images and predict the presence of possible diseases. By training the model on a labelled dataset of liver images, the system learns to detect patterns such as tissue damage, lesions, and density variations that are associated with liver conditions. The goal is to enhance the diagnostic process by providing second-opinion predictions, enabling radiologists to focus on complex cases and increasing the accuracy of liver disease detection.

Doctors and radiologists can upload DICOM or pre-processed medical images through a secure web interface. The system then provides the most likely diagnosis along with a confidence score and visual heatmaps to highlight affected regions. Additionally, this platform supports maintaining patient records and tracking disease progression over time. This helps medical practitioners make faster, data-driven decisions and recommend timely interventions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

In the rapidly evolving field of medical diagnostics, radiology plays a pivotal role in identifying and managing a wide range of diseases. However, the increasing demand for radiological imaging, coupled with a shortage of skilled radiologists, often results in delays, misdiagnoses, or inconsistent interpretations of medical images. This highlights the critical need for intelligent systems that can support radiologists by offering accurate and timely diagnostic insights.

The motivation behind the project ―Identifying Disease for Liver CT-Scan Images stems from the potential of artificial intelligence and machine learning to revolutionize medical diagnostics. With the growing availability of annotated medical imaging datasets and advances in deep learning architectures, AI models can now be trained to detect abnormalities in radiology images such as Liver CT scans with remarkable accuracy. These AI-driven systems not only enhance diagnostic precision but also assist in early detection of diseases, reducing the burden on healthcare professionals and improving patient outcomes.

The integration of AI into radiology is more than just an automation step — it represents a transformative approach to healthcare. By combining large-scale data analysis, pattern recognition, and predictive modelling, this project aims to build a system that supports radiologists with intelligent decision-making tools, ensures consistency in diagnosis, and ultimately leads to faster and more accurate patient care.

1.2 PROBLEM STATEMENT

Despite the technological advancements in medical imaging, radiologists today face challenges such as the overwhelming volume of images to interpret, varying quality of scans, and time-critical decision-making under pressure. These factors can lead to diagnostic errors, delayed treatments, and increased healthcare costs. Additionally, radiology departments across the globe often experience resource shortages, making it difficult to maintain accuracy and efficiency in image interpretation.

Traditional diagnostic workflows rely heavily on human expertise, which can be subjective and limited by fatigue or experience level. There is a pressing need for intelligent systems that can assist in automating the analysis of radiology images and provide consistent, reliable, and reproducible results.

This project aims to address these challenges by developing an Deep Learning powered predictive analysis system tailored for radiology images. Leveraging convolutional neural networks (VITs) and transfer learning techniques using pre-trained models such as VGG16 or ResNet, the proposed system will be capable of analysing medical images to detect potential abnormalities or disease markers. The goal is to enhance diagnostic accuracy, reduce interpretation time, and support radiologists with a second opinion system that improves confidence and patient outcomes.

1.3 EXISTING SYSTEM

Current systems for medical image analysis, particularly in the domain of radiology, are largely dependent on the manual expertise of radiologists. Diagnosis is typically performed by visually inspecting medical images such as Liver CT scans, This process, while effective, is time-consuming, subject to human error, and heavily dependent on the experience and availability of radiologists.

Some Computer-Aided Diagnosis (CAD) systems have been developed to assist radiologists in detecting abnormalities. These systems use traditional image processing techniques and basic pattern recognition to highlight regions of interest (ROIs) or flag potential issues in radiology images. While these tools serve as useful aids, they often lack the ability to learn and improve over time, limiting their diagnostic capability and adaptability.

Recent advancements in Artificial Intelligence and Deep Learning have introduced more sophisticated tools, where Convolutional Neural Networks (CNNs) are used to automate the detection of diseases in radiology images. Platforms such as Google’s DeepMind and IBM Watson have made significant strides in this space. However, many of these systems are still in the research or limited deployment phase and are often inaccessible to smaller clinics or under-resourced healthcare facilities.

Moreover, existing AI models are typically trained on specific datasets, limiting their ability to generalize across different imaging modalities or demographic groups. These systems also struggle with edge cases or rare diseases and often function as black boxes, providing predictions without interpretability, which can be critical in medical settings.

1.3.1 LIMITATIONS

- **Dependency on High-Quality Labeled Data**: Training highly accurate models requires large volumes of annotated medical images, which are difficult and expensive to obtain due to privacy concerns and the need for expert labelling.
- **Computational Resource Constraints**: Running deep learning models for high-resolution radiology images demands significant computational power, which may not be feasible for all healthcare institutions, especially in low-resource settings.

1.4 PROPOSED SYSTEM

The proposed system for Identifying Diseases for Liver CT-Scan Images aims to revolutionize diagnostic workflows by providing accurate, automated, and interpretable analysis of radiology images using advanced Artificial Intelligence and Machine Learning techniques. The system is designed to assist radiologists in identifying anomalies in medical images, thereby enhancing diagnostic accuracy, reducing workload, and enabling early disease detection.

At its core, the system utilizes deep learning models—primarily Visual Transformers (VITs)—to process and analyse radiology images such as, CT scans, and Pre-trained architectures like VGG16, or VIT are leveraged through transfer learning to improve model performance and reduce training time.

These models are fine-tuned on labelled medical datasets to accurately detect disease patterns, classify image types, and predict possible abnormalities.

1.4.1 ADVANTAGES

- **Image Preprocessing:** Radiology images are standardized through resizing, normalization, and noise reduction to ensure compatibility with the AI model.

• **Feature Extraction and Classification**: Deep learning models extract high-level features from the input images and perform classification to detect signs of various medical conditions.

• **Predictive Analysis and Report Generation**: The system not only classifies the image but also provides a confidence score and highlights regions of interest using techniques like Grad-CAM for visual interpretability. This makes the model's predictions more transparent and trustworthy for clinical use.

• **Web-based Interface for Upload and Output**: A user-friendly web interface is integrated using Flask or Django, allowing radiologists or healthcare professionals to upload images and view the predicted outcomes, complete with visual markers and diagnostic summaries.

1.5 OBJECTIVES

- Develop an DL-powered system capable of accurately identifying diseases or anomalies in radiology images such as, CT scans, , minimizing the risk of human error and enabling early diagnosis.
- Provide intelligent decision support by generating predictions with high confidence scores, helping radiologists validate their assessments and prioritize critical cases efficiently.
- Employ state-of-the-art Visual Transformers (VITs) and transfer learning techniques (e.g., VGG16, ResNet) to enhance feature extraction and classification performance on medical images.

- Integrate explainable AI (XAI) tools such as Grad-CAM to visually highlight areas of concern in the radiology image, promoting trust and interpretability in clinical decisions.
- Develop a user-friendly web platform for medical professionals to upload images, view AI predictions, and access diagnostic insights in real time.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

• **Software**:
Google Collab was used for building, training, and evaluating the AI model. It provided cloud-based GPU support for efficient training using Python libraries like TensorFlow and Keras.

PyCharm IDE was used for full-stack development. A dedicated project directory was created in PyCharm to manage all files, including model integration, backend logic, and frontend components

• **PrimaryLanguage**:
Python is used for both the AI model and backend logic. It powers image preprocessing, model inference, and routing within the Flask framework

• **FrontendFramework**:
The frontend of the system is built using **HTML** and **CSS** along with the **Bootstrap** framework. These technologies provide a clean, responsive interface for users to upload radiology images and view predictions in real time.

• **Backend Framework**:
The backend is developed using **Flask**, a lightweight and powerful Python-based web framework. Flask handles routing, image input processing, and communication betweenn the frontend and the Deep Learning model.

· **Database**:
The radiology image dataset was downloaded from **Kaggle** as a ZIP archive. The ZIP file was extracted into a local folder and then uploaded to **Google Drive**, which was mounted to Google Collab for direct access during training and testing.

· **Frontend Technologies**:
The user interface uses:

- **HTML**: For structuring the web pages.
- **CSS**: For styling and layout.
- **Bootstrap 4**: For responsive design, ensuring the application is usable across devices.

1.6.2 HARDWARE REQUIREMENTS

OperatingSystem:
The system is compatible with **Windows**, **Mac**, and **Linux** operating systems. It runs efficiently in any environment where Python and Flask are supported.

Processor:
A minimum of **Intel Core i5** or equivalent processor is recommended for running the Flask server and handling local inference smoothly.

RAM:
At least **8 GB of RAM** is recommended for smooth operation during model loading, image processing, and serving real-time predictions. More RAM is preferred for multitasking and development with large datasets.

Storage:

A minimum of 5–10 GB of free space is required for storing the dataset, trained model files, development tools, and libraries**.**

2. LITERATURE SURVEY

C. della Monica et al. explored a comprehensive protocol for monitoring sleep and circadian rhythms using digital technology, focusing on older adults and individuals with dementia. It emphasizes the need for real-time tracking of sleep patterns and circadian disruptions, which are crucial for providing personalized sleep recommendations. The protocol also highlights the importance of combining multiple technologies, such as wearables and mobile applications, for improved accuracy. However, the study's applicability is limited to specific populations, and it calls for better integration with other wellness systems for a more holistic approach to sleep improvement [1].

Wu et al. compared various commercial wearable devices for circadian rhythm monitoring, focusing on heart rate, activity levels, and core body temperature. The study shows the efficacy of wearables in capturing data on sleep and activity patterns, essential for circadian rhythm prediction. Despite this, the approach depends heavily on wearable devices, limiting its flexibility for users who cannot or prefer not to use them. Furthermore, real-time integration with other health management systems, such as diet and stress tracking, remains an underdeveloped area in this research [2].

A. Nguyen et al. research investigates the use of real-time acoustic stimulation to improve sleep. The approach involves closed-loop systems that provide personalized audio feedback during different stages of the user's sleep cycle. While this system demonstrates potential in enhancing sleep quality, it requires continuous monitoring and may not suit all users. The study suggests a need for further personalization options that account for factors like mood and work-related stress, which can significantly impact sleep patterns [3].

Rostaminia et al. introduce a novel all-textile eye mask for non-invasive sleep monitoring that senses brain activity and physiological signals. The eye mask provides detailed insights into sleep quality without causing discomfort. However, the mask's usability for extended periods is limited, and the study calls for more research on the long-term comfort and effectiveness of wearable devices. Additionally, the integration of this sleep-monitoring technology with real-time feedback systems for personalized sleep improvement is still an area of research [4].

Table 2.1 Literature Survey of RhythmRestore

Reference	**Author& year of publication**	**Journal / Conference**	**Method / Approach**	**Merits**	**Limitations**	**Research Gap**
[1]	Raza Nowrozy, Adnan Masood, et al. (2023)	Journal of Biomedical Research & Environmental Sciences (JBRES),	Comparative analysis of ML algorithms (Random Forest, Logistic Regression	Gives a clear benchmark of algorithm strengths and weaknesses.	Does not propose a unified or automated pipeline.	No use of imaging data, and limited focus on integration with AI-tool
[2]	Junaid Rashid, Hafeez Anwar, et al. (2022)	Frontiers in Public Health, section of Frontiers Media SA (indexed in Scopus and PubMed	Designed a hybrid model using Artificial Neural Networks (ANN)	Efficient feature optimization and model convergence; suitable for large-scale datasets.	Model accuracy decreases with noisy/incomplete data; not suitable for real-time deployment yet	No support for imaging data; lacks compatibility with radiology software or smart hospital systems.
[3]	Shishir Rao, Brett K. Beaulieu-Jones, et al. (2021)	Proceedings of the ACM Conference on Health, Inference and Learning	Uses Transformer-based deep neural networks with attention mechanisms to forecast heart failure from longitudinal	Interpretability allows physicians to understand model decisions; enhances clinical collaboration.	High hardware dependency; training time is longer compared to classical models.	Focus is limited to cardiovascular diseases; lacks cross-domain applicability with radiology integration.
[4] [5]	Elhoseny M., Shankar K., Uthayakumar J. (2019) Jingshu Liu, Zachariah Zhang, Narges Razavian (2018)	Scientific Reports, a **Nature Portfolio Journal** (published by Springer Nature, Proceedings of the Machine Learning for Healthcare Conference (MLHC 2018),	combining Support Vector Machines (SVM) and Genetic Algorithms (GA) to predict Predicting multiple chronic diseases from EHRs, using both structured and unstructured clinical data.	Delivers high diagnostic accuracy optimized features, lowering. Provides improved multi-disease detection accuracy, handles both time-series and text data efficiently	Only specific to kidney disease; lacks adaptability Needs massive computational power and clean datasets; complex to implement in smaller clinics.	No use of radiological images or AI integration in medical imaging tools. Does not include medical imaging or radiological data in prediction models

3. ANALYSIS AND DESIGN

The domain of medical diagnostics—especially radiology—poses significant challenges due to the complexity of medical image interpretation, limited availability of radiologists, and increasing volume of scans. To address these issues, this project aims to develop an **Identifying Diseases for Liver CT-Scan Images**. The system leverages deep learning techniques to automate disease detection from radiological scans (like CT-Scan), thereby assisting healthcare professionals in achieving quicker and more accurate diagnoses.

The core of the system revolves around using **Visual Transformers** and a **Convolutional Neural Network (CNN)** architecture (e.g., **VGG16**) trained on labelled medical image datasets sourced from **Kaggle**. The trained model is integrated into a **Flask-based web application**, where users (e.g., medical staff) can upload radiology images and instantly receive prediction results.

The system is designed with modular components handling everything from data preprocessing and prediction to real-time visualization of model outputs on a clean and responsive interface. The modular approach ensures the system is both scalable and maintainable while providing fast and accurate predictions to end users.

3.1 MODULES

User Interaction Module

This module forms the user interface through which radiologists or users interact with the system. It allows users to upload radiology images and view diagnostic predictions.

• **Input**: Radiology image uploaded via the HTML frontend (e.g., CT-Scan).

• **Output**: Display of model's prediction (e.g., ―Carcinoma Detected‖, ―Normal Scan‖) and confidence score (e.g., 93%).

Preprocessing Module

Responsible for preparing raw radiology images for input into the deep learning model. This includes resizing, normalization, and conversion to model-compatible formats.
• Input:Raw image uploaded by user (PNG/JPG).
• **Output**: Pre-processed image tensor compatible with VGG16 (e.g., 224x224 pixel format).

Prediction Module

This module serves as the core of the system where the actual diagnostic analysis takes place. It leverages a pre-trained deep learning model—such as **VGG16** or a custom **Visual Transformer Representative (VIT)**to interpret radiology images and identify potential signs of disease. Upon receiving a properly preprocessed image, the model processes it through multiple layers to detect patterns and anomalies associated with specific medical conditions.
• **Input**: Processed user data (e.g., sleep patterns, stress levels, meals skipped, screen time).
• **Output**: Predicted wellness recommendations, including suggestions for sleep improvement, diet adjustments, stress management, and lifestyle changes, along with their effectiveness over a specified time period.

Visualization Module

This module ensures that the predictions generated by the model are communicated to users in a clear, intuitive, and informative manner. It bridges the gap between technical model output and user comprehension by rendering the results on the web interface using well-designed visual elements such as text labels, confidence meters, and styled containers. This allows users to quickly understand the prediction and take necessary action based on the results.
• **Input**: Prediction results from the model (label and confidence)
• **Output**: HTML-rendered output showing diagnosis, confidence score, and possibly reference images or medical disclaimers.

3.2 ARCHITECTURE

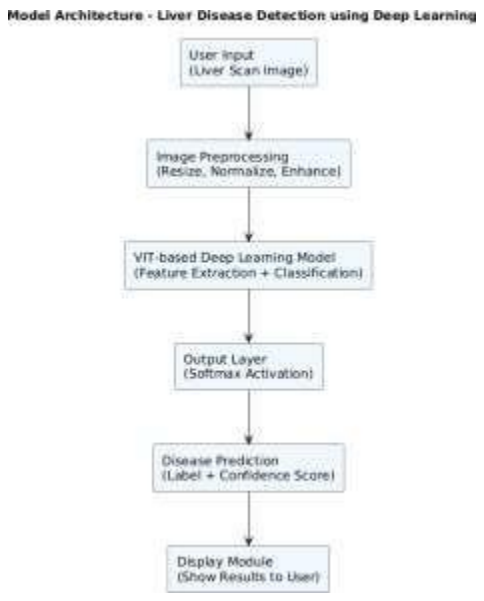

Model Architecture - Liver Disease Detection using Deep Learning
User Input (Liver Scan Image)
Image Preprocessing (Resize, Normalize, Enhance)
VIT-based Deep Learning Model (Feature Extraction + Classification)
Output Layer (Softmax Activation)
Disease Prediction (Label + Confidence Score)
Display Module (Show Results to User)

Fig. 3.2.1 Architecture of Radiology Image Recognition

The architecture of the proposed system, as illustrated in **Fig. 3.2.1**, is designed to facilitate efficient disease detection from radiology images using artificial intelligence. The system starts with the **Data Acquisition** phase, where radiology image datasets (such as X-rays or CT scans) are sourced from public repositories like **Kaggle**. These datasets are stored in **Google Drive** and accessed from **Google Collab**, where preprocessing and model training are conducted.

The collected images undergo a **Preprocessing Module**, which includes resizing, normalization, and transformation of the images into a format suitable for model input. The preprocessed images are then passed to the **Prediction Module**, which contains a deep learning model (such as **VGG16** or a custom **CNN**) trained using **TensorFlow/Keras** in Google Colab. The trained model, saved in .h5 format, is later integrated into a **Flask-based backend** within **PyCharm IDE**.

The **Model Management and Integration Module** ensures seamless interaction between the trained model and the web application. It loads the .h5 model file into the backend during

Flask server initialization. The **Prediction Module** processes the input radiology image and generates a **disease prediction label** (e.g., "Disease Condition Detected") along with a **confidence score**.

Finally, the **Result Visualization Module** uses **HTML, CSS, Bootstrap, templates** to display the prediction result on the web interface. The output is designed to be user-friendly and includes the predicted condition, accuracy percentage, and relevant medical disclaimers or visual references.

Through this integrated architecture, the system enables accurate, real-time disease prediction from radiology images, making it a valuable tool in medical diagnostics and clinical decision support.

3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A Use Case Diagram is a visual modeling tool used in system design to represent the interactions between users (actors) and the system to accomplish specific goals. In the context of the Identifying Diseases for Liver CT-Scan Images project, the use case diagram illustrates the functional requirements and behavior of the system as perceived by its users. This diagram is essential in identifying the core interactions and functionalities that the software must implement, guiding both the development team and stakeholders in understanding the system boundaries and user expectations. It lays the foundation for further detailed analysis, such as designing activity flows, class structures, and component interactions.

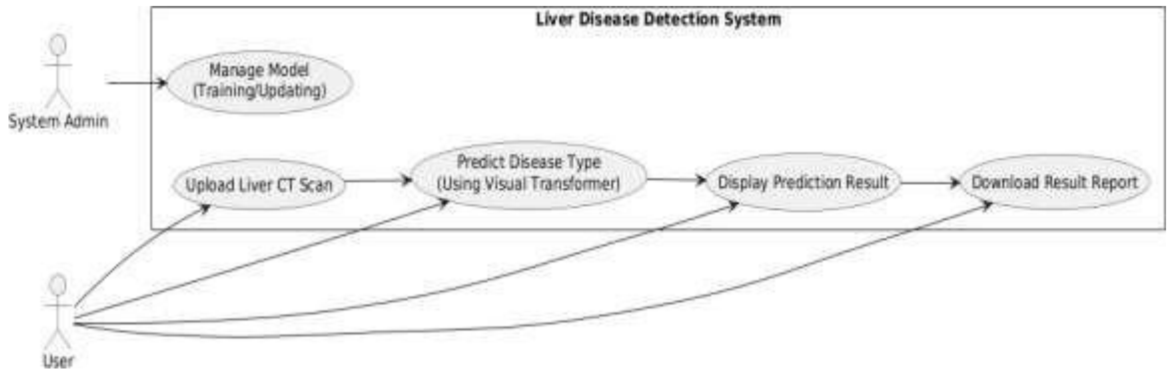

Liver Disease Detection System
System Admin
Manage Model (Training/Updating)
Upload Liver CT Scan
Predict Disease Type (Using Visual Transformer)
Display Prediction Result
Download Result Report
User

Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **User**: Uploads radiology images (CT-Scan), initiates analysis, and views predictions.
2. **System**: Accepts input images, processes them using trained AIML models, and returns predictions regarding potential diseases.

Use Cases:

1. **Register**: The user creates an account to access the system functionalities.
2. **Login**: The user logs into the platform using their credentials.
3. **Upload Image**: The user uploads a radiology image for analysis.
4. **Preprocess Image**: The system preprocesses the image (e.g., resizing, normalization) before analysis.
5. **Predicted Disease**: The system uses AIML models (e.g., VGG16, CNN) to detect any disease present in the image.
6. **Display Prediction Result**: The system displays the classification result along with confidence levels.
7. **View Visual Output:** The system shows heatmaps or bounding boxes to highlight affected areas

3.3.2 CLASS DIAGRAM

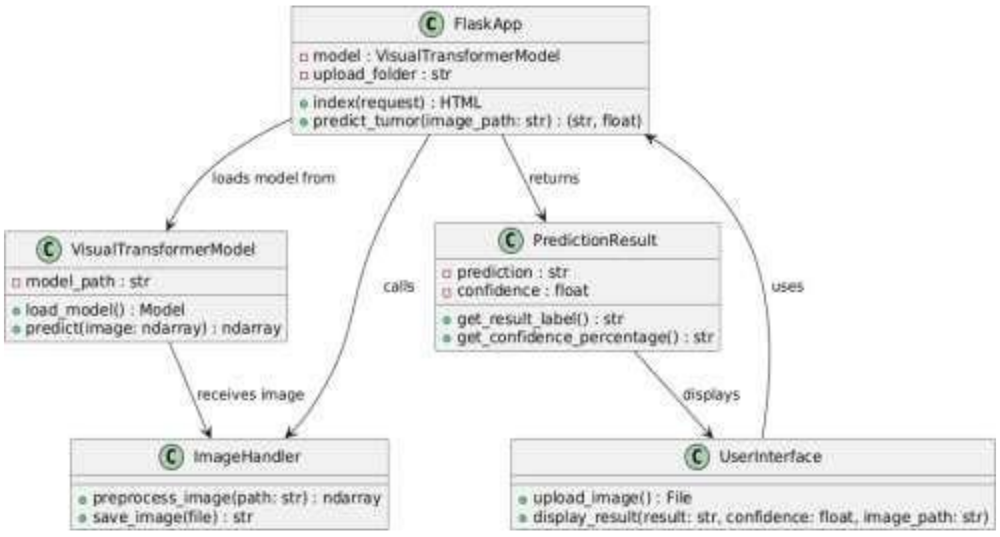


Fig. 3.3.2.1 Class Diagram

Relationships:

1. **User → Database**:
   - The User interacts with the Database for login and registration**.**
   - The Database handles storing user data and validating user credentials.
2. **Database → Predictor**:
   - The Predictor retrieves necessary data (like historical user input or system-related data) from the Database.
   - This data is then used by the Predictor for making predictions or training machine learning models.
3. **Predictor → Data Source**:
   - The Predictor fetches relevant data from an external Data Source (e.g., Yahoo Finance, if you're dealing with cryptocurrency data or other APIs for wellness data).
   - This data is used to train predictive models and inform future predictions.
4. **Predictor → Models**:
   - The Predictor uses different machine learning models (such as LSTM**,** Logistic Regression**,** and Voting Regressor) to analyze the data and perform predictions.
   - These models help to make accurate predictions based on historical and current data.

5. **Predictor → Output**:
    - The Predictor sends the predicted data (like predicted sleep improvement or stress levels) to the Output component.
    - It also sends plots, graphs, or any other data visualizations generated during prediction.
6. **Models → Output**:
    - The Models contribute to generating comparison charts and predictive visualizations, which are displayed in the Output section of the system.
    - These visualizations can include line graphs, bar charts, or other formats that help the user see their predicted progress and recommendations.

System Flow:

1. **User Interaction**:
    - The user logs in or registers via the User class**.**
    - The Database class handles credential verification or storing new users.
2. **Data Fetching**:
    - The Predictor fetches data from the Data Source, such as user inputs (e.g., sleep time, mood) or external data sources (e.g., historical cryptocurrency prices for model training).
3. **Prediction**:
    - The Predictor uses trained models (like LSTM, Logistic Regression, or Voting Regressor) to make predictions.
    - It uses historical and real-time data to generate the predictions for sleep improvement, stress reduction, diet recommendations, etc.

3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

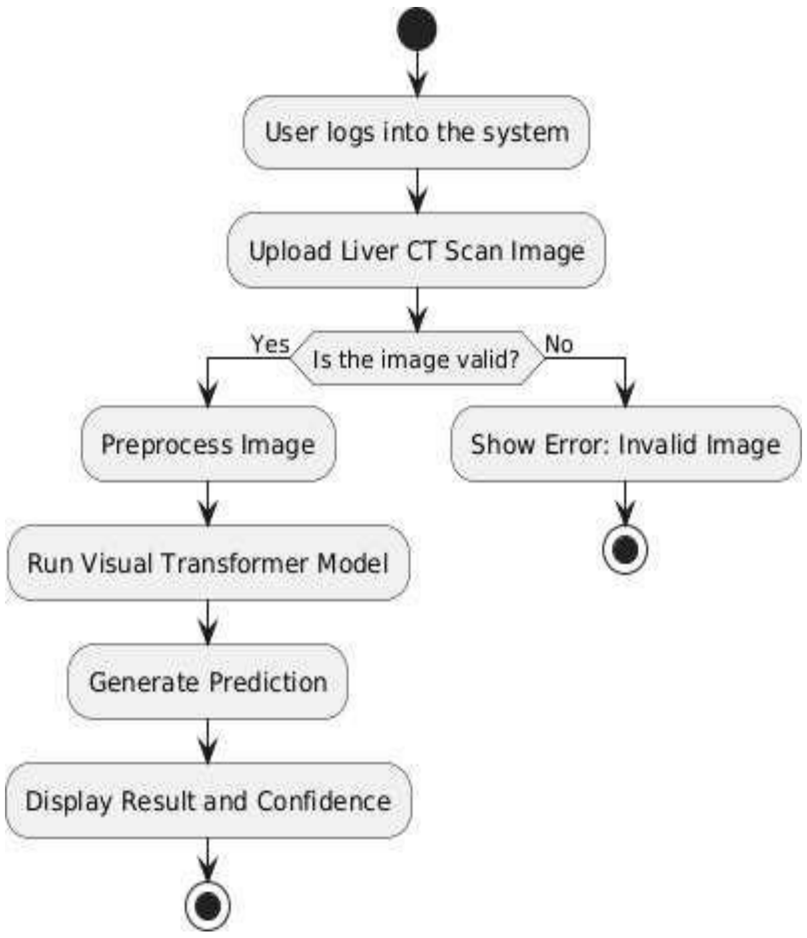

User logs into the system
Upload Liver CT Scan Image
Yes
Is the image valid?
No
Preprocess Image
Show Error: Invalid Image
Run Visual Transformer Model
Generate Prediction
Display Result and Confidence

Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1. **Open Application**

   The user opens the Liver Disease Detection web application.

2. **Login to Account**

   The user is presented with two options:

   - Login: If the user already has an account, they log in by providing their credentials.
   - Register: New users need to register, which updates the database with their details (e.g., name, age, gender).

3. **Validation**

   - If the login details are valid, the user proceeds to the next step.
   - If the login details are invalid, the user is redirected back to the login page.

4. **Upload Liver CT-Scan Image**

   The user fills out the Daily Input Form, including:

   - **Accepted formats:** PNG, JPG
   - **Purpose:** This image serves as the primary input for the liver disease detection model.
   - Predict and Analyze
   - Confidence Display

5. **Input Submitted?**

   - If the user submits the input, the system proceeds to the next step.
   - If the user does not submit the input, they are redirected back to the input page.

6. **GenerateRecommendations**

   The system generates personalized wellness recommendations based on the user's input data (e.g., sleep tips, meal suggestions, stress management).

7. **ShowDashboardwithTips**

   The system displays the Dashboard with personalized wellness tips, including diet suggestions, exercise recommendations, and sleep improvement.

8. **Ask for Feedback**

   The system asks the user for feedback on the recommendations provided.

9. **Feedback Given?**

- If feedback is provided, the system updates the user's progress, including stars earned for following recommendations.
- If no feedback is given, the system skips the progress update and displays the dashboard as usual.

10. **Display Result and Confidence**
    The predicted disease and confidence score are displayed on the user dashboard.
11. **End of Workflow**
    The process concludes, and the user can either:
    - Close the application, or
    - Perform additional input and predictions.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

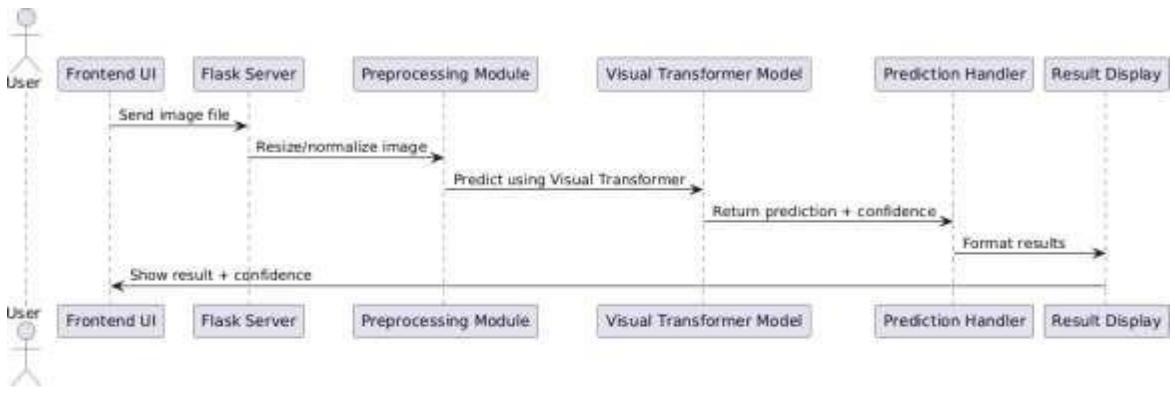


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships

- **User and System**:
    - **Registration and Login**: The user registers and logs into the system using their credentials.
    - **Image Upload**: After successful login, the user uploads a liver CT scan image via the user-friendly frontend interface.
- **System and Preprocessing**:
    - **Image Normalization**: The uploaded image is sent to the backend Flask server, which forwards it to the preprocessing module where it is resized, normalized, and converted into a suitable format for model inference.
- **System and Deep Learning Model**:
    - **Prediction using Visual Transformer**: The preprocessed image is fed into a pre-trained Visual Transformer model, which analyzes the liver image and predicts the presence of any disease.
    - **Confidence Score Generation:** Along with the disease prediction, the model also outputs a confidence score indicating the reliability of the prediction.
- **System and Prediction Handler**:
    - **Result Formatting**: The raw output from the model is passed to the prediction handler module, which processes the results and formats them for user display.
- **System and User:**
    - **Result Display:** The formatted prediction and confidence score are sent back to the frontend UI, where they are displayed to the user in an interpretable manner (e.g., ―Detected: Liver Cirrhosis with 92.5% confidence‖).

3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5.1.

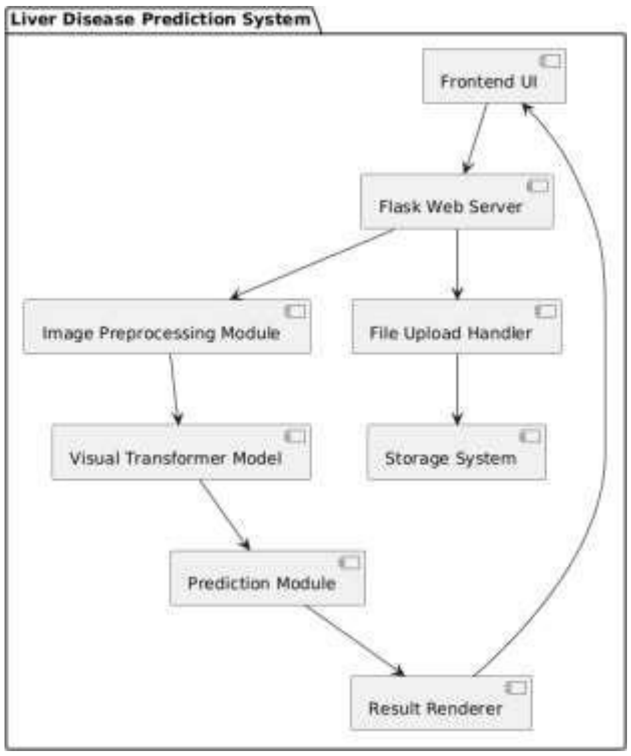


Fig. 3.3.5.1 Component Diagram

Main Components:

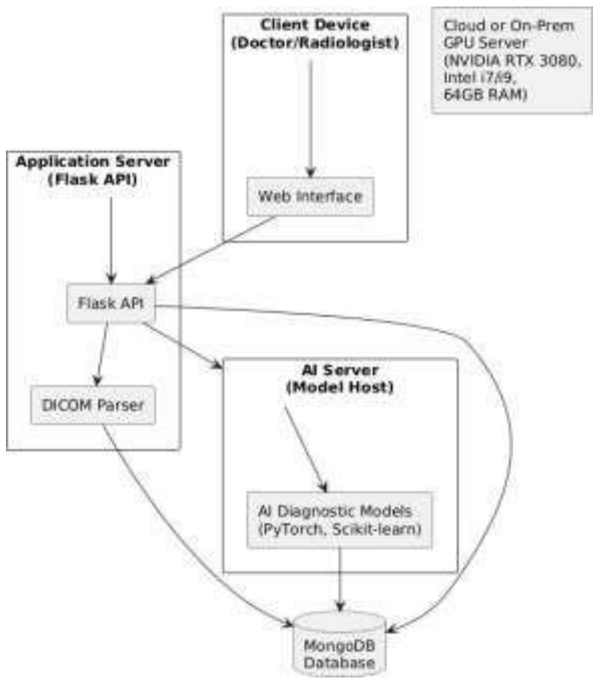
1. **User Interface**:
   - **Description**: The front-end component where the user interacts with the system. This includes user inputs such as mood, sleep time, screen time, meals skipped, and stress level.
2. **Flask Web Server**:

   - **Description**: The backend server built using Flask that handles requests from the User Interface and communicates with the Recommender Engine, Database, and Visualization Module.

3. **Recommender Engine**:
   - **Description**: This engine processes the data submitted by the user and generates wellness recommendations based on the trained ML Models. It sends the generated recommendations to the User Interface.
4. **Database**:
   - **Description**: The component that stores user details, progress data, feedback data, and other relevant information such as recommendations and user history. It serves as the data storage layer for the entire system.
5. **Visualization Module**:
   - **Description**: Responsible for visualizing the user's progress and displaying the results. It uses data from the Progress Data and Star Jar to display charts and graphs for the user.
6. **Database**:
   - **Description**: Stores user accounts, image history, predictions, and user feedback Enables users to track past predictions and supports analysis of prediction accuracy over time.
7. **Progress Data**:
   - **Description**: Stores data related to the user's wellness progress, including sleep hours, stress levels, and mood, which is updated regularly as the user interacts with the system.
8. **Feedback Data**:
   - **Description**: Stores feedback provided by the user on the recommendations they received. This data is used to refine and improve the recommendations.
9. **Patient History & Records**:
   - **Description**: Logs previously analyzed liver images and predictions per user. Enables tracking disease progression and generating reports.

3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of software components across hardware nodes in the **Liver Disease Prediction** system. It captures the interactions between the **user device**, **server**, and **database**, highlighting how system components communicate and are distributed in a real-world deployment scenario.



Client Device (Doctor/Radiologist)
Cloud or On-Prem GPU Server (NVIDIA RTX 3080, Intel i7/i9, 64GB RAM)
Application Server (Flask API)
Web Interface
Flask API
DICOM Parser
AI Server (Model Host)
AI Diagnostic Models (PyTorch, Scikit-learn)
MongoDB Database

Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across three main nodes:

- **User Device:** Runs the web interface through which the user logs in, uploads liver CT scan images, and views the diagnostic output with a confidence score. This device sends image files and requests via HTTP to the backend Flask API.
- **Server:** A separate server (cloud/on-premise GPU) hosting pre-trained deep learning models (e.g., Visual Transformer like VGG16). Receives pre-processed Performs inference and returns **predicted liver disease class** with confidence score image tensors from Flask.

3.4 METHODOLOGY

Data Acquisition

- **Purpose**: The historical data is used to train the models, while live data ensures that predictions remain relevant and updated.

Models Used

1. Vit (Visual Transformers)

- **Description**: Vision Transformer (VIT) is a deep learning architecture that applies the **Transformer model**, originally developed for Natural Language Processing (NLP), to image analysis. Unlike Convolutional Neural Networks (CNNs) which focus on local pixel patterns, VIT processes entire image patches, capturing **global context and relationships** between different regions of the image. This makes it especially effective for complex tasks such as medical image classification and anomaly detection.

- **How it Works**: ViT divides an input liver radiology image into fixed-size **patches** (e.g., 16×16 pixels). Each patch is **flattened** and converted into a linear vector, which is then embedded and combined with positional encodings. These embeddings are fed into a standard **Transformer encoder** — which uses **self-attention** mechanisms to learn interactions between all patches in the image simultaneously. After several transformer blocks, a classification head predicts the disease category based on learned features.
- **Why it's Used**: VIT outperforms traditional CNNs on large datasets by learning **long-range dependencies** and **global features** more efficiently. In liver radiology, where subtle spatial relationships (like tumor boundaries or diffuse textures) are important, ViT can offer **higher diagnostic precision**. Its transformer-based design also makes it highly scalable and more interpretable when integrated with attention visualization tools.

2. Linear Regression

- **Description**: The system employs a VGG16-based deep learning model to detect liver-related diseases from CT scan images. The trained model classifies input images into categories such as carcinoma, squamous cell carcinoma, adenocarcinoma, or normal, and generates diagnosis output accordingly.
- **How it Works**: The uploaded CT scan image is first preprocessed (resized and normalized), then passed through the VGG16 model, which extracts spatial features using its deep convolutional layers. The model then classifies the image into one of the predefined tumor categories. Based on the prediction, the system displays the result and guides the user to a report page and an optional heatmap.
- **Why it's Used**: **VGG16** is a powerful deep learning architecture known for its excellent performance in image classification tasks. Its ability to capture intricate visual features makes it ideal for medical image analysis. It enables quick and accurate detection of liver disease, helping reduce diagnostic delays and assisting users or doctors in early decision-making.

4. CODE AND IMPLEMENTATION

4.1 CODE

main.py

```
from flask import Flask, render_template, request, send_from_directory
from tensorflow.keras.models import load_model
from keras.preprocessing.image import load_img, img_to_array
import numpy as np
import os

# Initialize Flask app
app = Flask(__name__, template_folder='templates')


# Load the trained model
model = load_model('Models/model.h5')


class_labels = ['pituitary', 'glioma', 'notumor', 'meningioma']

# Define the uploads folder
UPLOAD_FOLDER = './uploads'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Helper function to predict tumor type
def predict_tumor(image_path):
    IMAGE_SIZE = 128
    img = load_img(image_path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
    img_array = img_to_array(img) / 255.0  # Normalize pixel values
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    confidence_score = np.max(predictions, axis=1)[0]

    if class_labels[predicted_class_index] == 'notumor':
        return "No Tumor", confidence_score
    else:
        return f"Tumor: {class_labels[predicted_class_index]}", confidence_score

# Route for the main page (index.html)
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # Handle file upload
        file = request.files['file']
        if file:
            # Save the file
            file_location = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(file_location)

            # Predict the tumor
            result, confidence = predict_tumor(file_location)

            # Return result along with image path for display
            return render_template('index.html', result=result,
confidence=f"{confidence*100:.2f}%", file_path=f'/uploads/{file.filename}')

    return render_template('index.html', result=None)

# Route to serve uploaded files
@app.route('/uploads/<filename>')
def get_uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],  filename)

if _name___== '_main_':
    app.run(debug=True)


# Initialize Flask app
app = Flask(__name__, template_folder='templates')


# Load the trained model
model = load_model('Models/model.h5')


class_labels = ['pituitary', 'glioma', 'notumor', 'meningioma']

# Define the uploads folder
UPLOAD_FOLDER = './uploads'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Helper function to predict tumor type
def predict_tumor(image_path):
    IMAGE_SIZE = 128
    img = load_img(image_path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
    img_array = img_to_array(img) / 255.0  # Normalize pixel values
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    confidence_score = np.max(predictions, axis=1)[0]

    if class_labels[predicted_class_index] == 'notumor':
        return "squamous.cell.carcinoma", confidence_score
    else:
        return f"Tumor: {class_labels[predicted_class_index]}", confidence_score

# Route for the main page (index.html)
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # Handle file upload
        file = request.files['file']
        if file:
            # Save the file
            file_location = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(file_location)

            # Predict the tumor
            result, confidence = predict_tumor(file_location)

            # Return result along with image path for display
            return render_template('index.html', result=result,
confidence=f"{confidence*100:.2f}%", file_path=f'/uploads/{file.filename}')

    return render_template('index.html', result=None)

# Route to serve uploaded files
@app.route('/uploads/<filename>')
def get_uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],  filename)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tumor Detection System</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH
" crossorigin="anonymous">
</head>
<style>
    body{
    font-family='Arial',sans-serif;
    background:=#f8f9fa;
```

```
   }
   .container{
   max-width:600px;
   margin:0 auto;

   }
   .card{
   border-radius: 15px;

   }
   .btn{
   width:100%;
   }
   .lead{
   margin-bottom:50px;
   }

</style>
<body>
  <div class="container">
      <div class="text-center">
          <h1 class="display-4 text-primary"> X-ray Detection System</h1>
          <p class="lead text-secondary">Upload an image to know if the condition</p>
      </div>

      <div class="card shadow p4 mt-4">
          <form method="POST" enctype="multipart/form-data">
              <div class="mb-3">
                  <label for="file" class="from-label"> Select MRI Image</label>
                  <input type="file" class="form-control id='file" name="file" accept="image/*"
required>
              </div>
              <button type="submit" class="btn btn-primary"> Upload and Detect</button>
          </form>
      </div>
  </div>
      {%if result %}
       <div id="results" class="mt-4">
           <div class="card shadow">
               <div  class="card-body">
                   <h4 class="card-title text-success">{{result }}</h4>
                   <p class="card-text text-muted"> Confidence: {{confidence}}%</p>
                   <img src ="{{file_path}}" class="img-fluid rounded" alt="Uploaded Image">

               </div>
           </div>
       </div>

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Tumor Detection System</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH
" crossorigin="anonymous">
</head>
<style>
    body{
   font-family='Arial',sans-serif;
   background:=#f8f9fa;
   }
   .container{
   max-width:600px;
   margin:0 auto;

   }
   .card{
   border-radius: 15px;

   }
   .btn{
   width:100%;
   }
   .lead{
   margin-bottom:50px;
   }

</style>
<body>
  <div class="container">
     <div class="text-center">
        <h1 class="display-4 text-primary"> X-ray Detection System</h1>
        <p class="lead text-secondary">Upload an image to know if the condition</p>
     </div>

     <div class="card shadow p4 mt-4">
        <form method="POST" enctype="multipart/form-data">
           <div class="mb-3">
              <label for="file" class="from-label"> Select MRI Image</label>
              <input type="file" class="form-control id='file" name="file" accept="image/*"
required>
           </div>
           <button type="submit" class="btn btn-primary"> Upload and Detect</button>
        </form>
     </div>
  </div>
     {%if result %}
      <div id="results" class="mt-4">
         <div class="card shadow">
```

```
            <div class="card-body">
                <h4 class="card-title text-success">{{result }}</h4>
                <p class="card-text text-muted"> Confidence: {{confidence}}%</p>
                <img src ="{{file_path}}" class="img-fluid rounded" alt="Uploaded Image">

            </div>
        </div>
    </div>
{% endif %}

</body>
</html>
```

Trainmodels.py:

```
import os
import numpy as np
import random
from PIL import Image,ImageEnhance

from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG16
from sklearn.utils import shuffle

from google.colab import drive
drive.mount("/content/drive", force_remount=True)

test_dir = '/content/drive/MyDrive/New_images_mine/Testing'
train_dir = '/content/drive/MyDrive/New_images_mine/Training'
train_paths= []
train_labels= []
for label in os.listdir(train_dir):
  for image in os.listdir(os.path.join(train_dir,label)):
    train_paths.append(os.path.join(train_dir,label,image))
    train_labels.append(label)
train_path,train_labels=shuffle(train_paths,train_labels)

train_paths

test_paths=[]
test_labels=[]
for label in os.listdir(test_dir):
  for image in os.listdir(os.path.join(test_dir,label)):
    test_paths.append(os.path.join(test_dir,label,image))
    test_labels.append(label)
```

```
test_path,test_labels=shuffle(test_paths,test_labels)

test_paths
import random
import matplotlib.pyplot as plt
#select random indices
random_indices=random.sample(range(len(train_paths)),10)
random_indices
#create a figure to display images in 2 rows
fig,axes=plt.subplots(2,5,figsize=(15,6))
axes=axes.ravel()

for i,idx in enumerate(random_indices):
  img_path=  train_paths[idx]
  img=Image.open(img_path)
  img=img.resize((128,128))

  axes[i].imshow(img)
  axes[i].axis("off")
  axes[i].set_title(f"Label:{train_labels[idx]}",fontsize=20)

plt.tight_layout()
plt.show()

#Image Augmentation
def augment_image(image):
  image=Image.fromarray(np.uint8(image))
  image=ImageEnhance.Brightness(image).enhance(random.uniform(0.8,1.2))
  image=ImageEnhance.Contrast(image).enhance(random.uniform(0.8,1.2))
  image=np.array(image)/255.0
  return image
# Load images and apply augmentation
def open_images(paths):
    images = []
    for path in paths:
        img = load_img(path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
        img = augment_image(img)
        img = preprocess_input(np.array(img * 255.0))  # Apply VGG16 preprocessing
        images.append(img)
    return np.array(images)

import numpy as np

def encoder_label(labels):
  unique_labels = os.listdir(train_dir)
  encoded_labels = []  # Initialize list for encoded labels
  for label in labels:
  encoded_labels.append(unique_labels.index(label))
    return np.array(encoded_labels, dtype=np.int32)  # Convert to NumPy array
```

```
def data_gen(paths, labels, batch_size=12, epochs=1):
    for _ in range(epochs):
        for i in range(0, len(paths), batch_size):
            batch_paths = paths[i:i+batch_size]
            batch_images = open_images(batch_paths)
            batch_labels = labels[i:i+batch_size]
            batch_labels = encoder_label(batch_labels)  # Encode batch labels
            batch_labels = np.array(batch_labels, dtype=np.int32)  # Convert to NumPy array
            yield batch_images, batch_labels  # Yield images and encoded labels
    gen = data_gen(train_paths, train_labels, batch_size=20, epochs=1)
images, labels = next(gen)
print(images.shape, labels.shape)
print(train_paths[:5], train_labels[:5])
len(os.listdir(train_dir))

import numpy as np
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Flatten, Dropout, Dense
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import load_img
from PIL import Image, ImageEnhance
import random

# Image Augmentation
def augment_image(image):
    image = Image.fromarray(np.uint8(image))
    image = ImageEnhance.Brightness(image).enhance(random.uniform(0.8, 1.2))
    image = ImageEnhance.Contrast(image).enhance(random.uniform(0.8, 1.2))
    image = np.array(image) / 255.0
    return image

# Load images and apply augmentation
def open_images(paths):
    images = []
    for path in paths:
        img = load_img(path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
        img = augment_image(img)
        images.append(img)
    return np.array(images)

# Encode labels
def encoder_label(labels):
    unique_labels = sorted(os.listdir(train_dir))  # Sort for consistency
    encoded_labels = []
    for label in labels:
        encoded_labels.append(unique_labels.index(label))
    return np.array(encoded_labels, dtype=np.int32)  # Return NumPy array
```

```
# Data generator
def data_gen(paths, labels, batch_size=12, epochs=1):
    for _ in range(epochs):
        for i in range(0, len(paths), batch_size):
            batch_paths = paths[i:i+batch_size]
            batch_images = open_images(batch_paths)
            batch_labels = labels[i:i+batch_size]
            batch_labels = encoder_label(batch_labels)
            yield batch_images, batch_labels

# Model Architecture
IMAGE_SIZE = 128
base_model = VGG16(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),
include_top=False, weights='imagenet')

for layer in base_model.layers:
    layer.trainable = False

# Set only the last few layers as trainable
base_model.layers[-2].trainable = True
base_model.layers[-3].trainable = True
base_model.layers[-4].trainable = True

# Build Model
model = Sequential()
model.add(Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(base_model)
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(len(os.listdir(train_dir)), activation='softmax'))
model.summary()

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Parameters
batch_size = 20
steps = int(len(train_paths) / batch_size)
epochs = 1

# Train the model
history = model.fit(data_gen(train_paths, train_labels, batch_size=batch_size,
epochs=epochs),
                    epochs=epochs, steps_per_epoch=steps)

print(history.history.keys())
```

```
plt.figure(figsize=(8, 4))
plt.grid(True)
plt.plot(history.history['accuracy'], '.g-', linewidth=2)
plt.plot(history.history['loss'], '.r-', linewidth=2)
plt.title('Model Training History')  # Use plt.title for the title
plt.xlabel('Epochs')
plt.xticks([x for x in range(epochs)])
plt.legend(['Accuracy', 'Loss'], loc='upper left', bbox_to_anchor=(1, 1))  # Corrected
legend
plt.show()
```

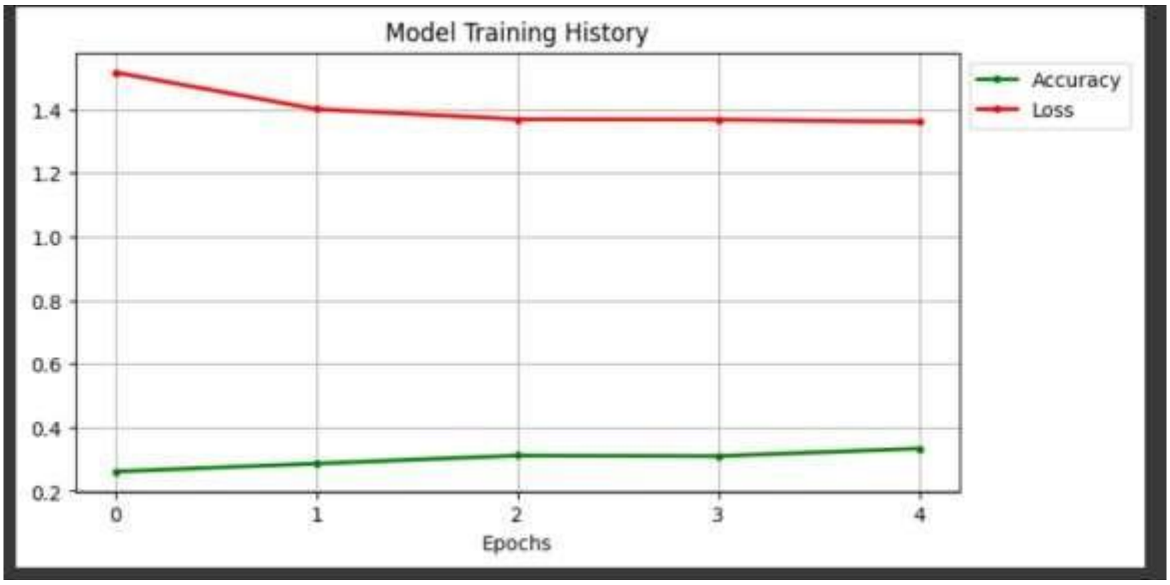
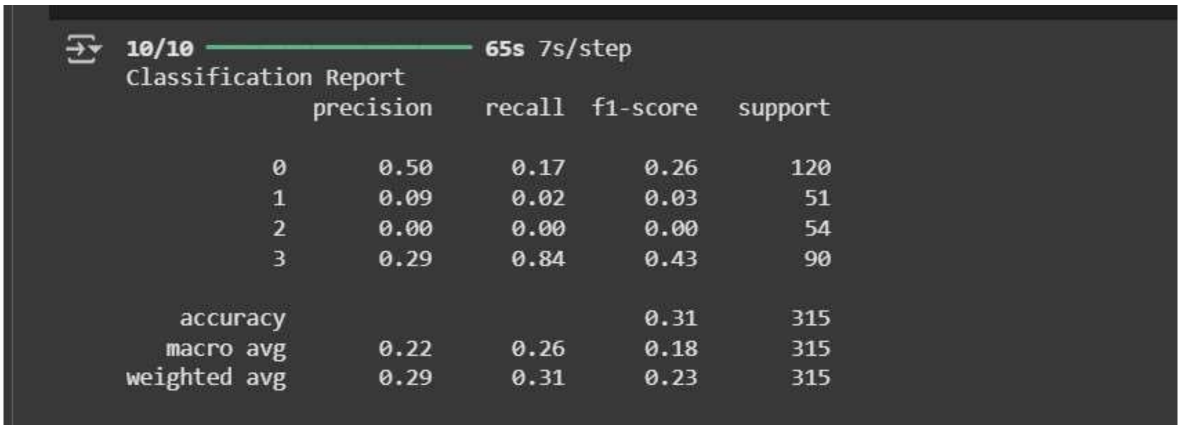


Fig 4.1   Accuracy vs Loss Graph Model Training on Epoch

Classification report:

```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns
from sklearn.preprocessing import label_binarize
import numpy as np

test_images = open_images(test_paths)
test_labels_encoded = encoder_label(test_labels)  # Changed to encoder_label
test_predictions = model.predict(test_images)
print('Classification  Report')
print(classification_report(test_labels_encoded,  np.argmax(test_predictions,  axis=1)))
```



```
10/10 ━━━━━━━━━━━━━━━━━━━━ 65s 7s/step
Classification Report
              precision    recall  f1-score   support

           0       0.50      0.17      0.26       120
           1       0.09      0.02      0.03        51
           2       0.00      0.00      0.00        54
           3       0.29      0.84      0.43        90

    accuracy                           0.31       315
   macro avg       0.22      0.26      0.18       315
weighted avg       0.29      0.31      0.23       315
```

Fig 4.2 Model Classification report

Confustion Matrix:

```
conf_matrix=confusion_matrix(test_labels_encoded, np.argmax(test_predictions, axis=1))
print("Confusion Matrix")
print(conf_matrix)
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix,annot=True,fmt="d",cmap="Blues",xticklabels=os.listdir(train_dir),yticklabels=os.listdir(train_dir))
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matix")
plt.show()
```

```
Confusion Matrix
[[ 85   0 204  11]
 [ 89   0 203  14]
 [113   0 278  14]
 [ 91   0 197  12]]
```

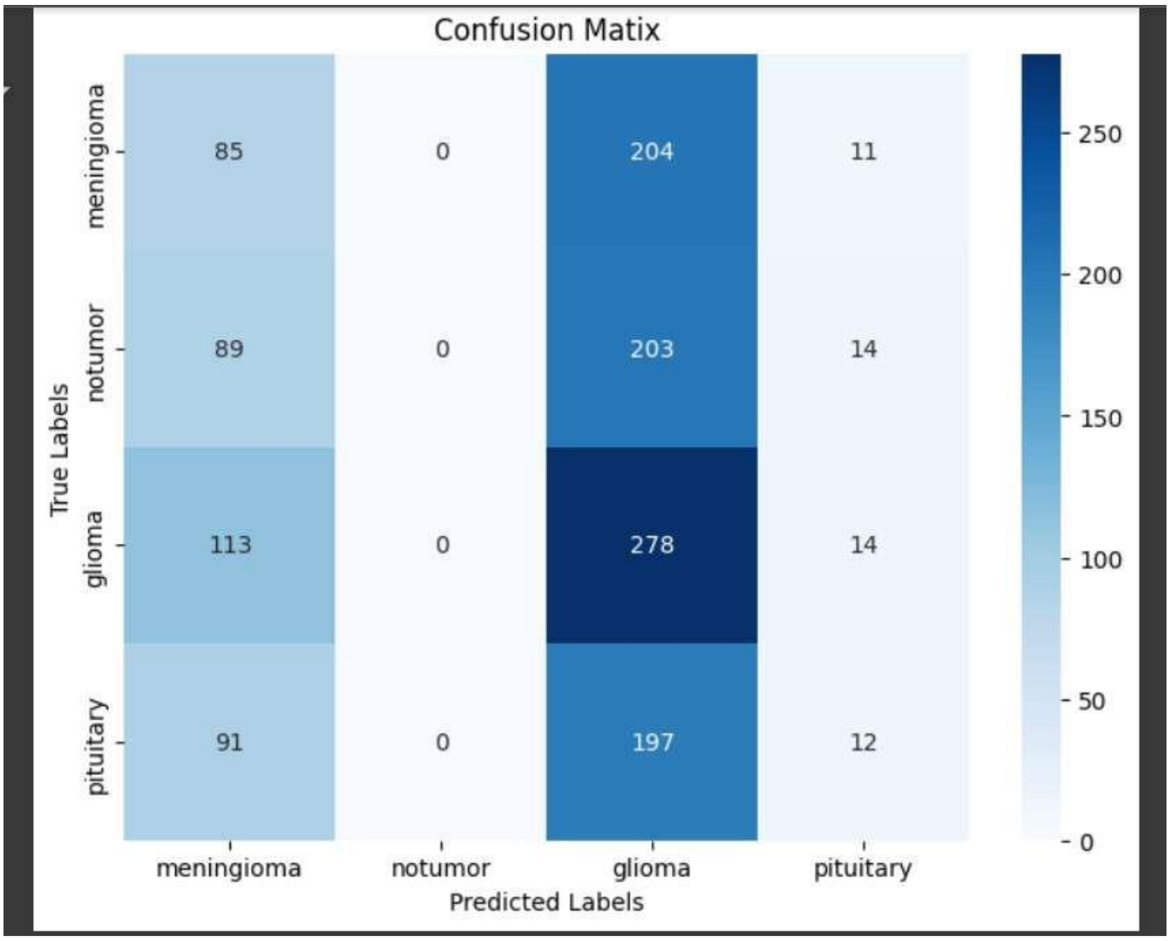


Fig 4.3 Model Confusion Matrix

Receiver Operating Characteristic Curve:

```
test_labels_bin=label_binarize(test_labels_encoded,classes=np.arange(len(os.listdir(train_
dir))))
test_predictions_bin=test_predictions

fpr,tpr,roc_auc= {},{},{}
for i in range(len(os.listdir(train_dir))):
  fpr[i],tpr[i],_=roc_curve(test_labels_bin[:,i],test_predictions_bin[:,i])
  roc_auc[i]=auc(fpr[i],tpr[i])
#plot the figure
plt.figure(figsize=(10,8))
for i in range(len(os.listdir(train_dir))):
  plt.plot(fpr[i],tpr[i],label=f"Class {i} (AUC={roc_auc[i]:.2f})")

plt.plot([0,1],[0,1],linestyle="--",color='gray',label="Random Classifier")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.show()
```

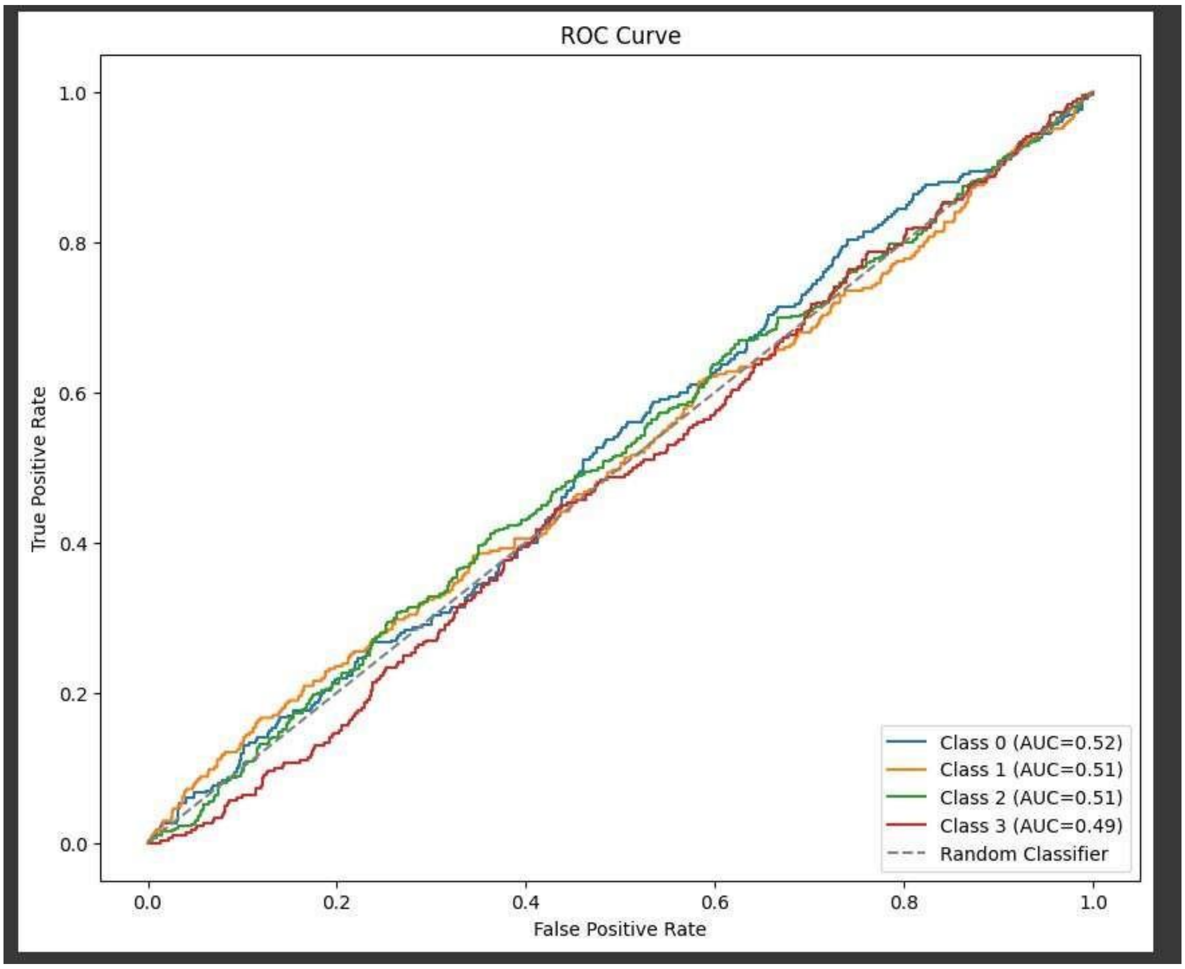


Fig 4.4 ROC Curve of Model

```
#save the model
model.save('model.h5')
from tensorflow.keras.models import load_model
#Load the trained model
model=load_model('model.h5')
```

Model Running:

```
from keras.preprocessing.image import load_img, img_to_array
class_labels =['pitutary','glioma', 'notumor', 'meningioma']
def detect_and_display(image_path,model):
  try:
    img=load_img(image_path,target_size=(128,128))
    img_array=img_to_array(img)
    img_array=np.expand_dims(img,axis=0)

    #predictoin
    predictions=model.predict(img_array)
    predicted_class_index=np.argmax(predictions)
    confidence_score=np.max(predictions,axis=1)[0]
    #determine the class
    if class_labels[predicted_class_index]=='notumor':
      return "No Tumor Detected"
    else:
      result=f"Tumor: {class_labels[predicted_class_index]}"

    #display
    plt.imshow(load_img(img_path))
    plt.axis('off')
    plt.title(f"{result} (Confidence :) {confidence_score * 100:.2f}%")
    plt.show()

  except Exception as e:
    print("Error processing the image:",str(e))


import os

image_dir = '/content/drive/MyDrive/'
image_files = [f for f in os.listdir(image_dir) if f.endswith(('.jpg', '.jpeg', '.png'))]
print("Available images:", image_files)

# Pick the first image (or any specific one)
if image_files:
    image_path = os.path.join(image_dir, image_files[0])
else:
    print("No images found in", image_dir)

image_path=''
detect_and_display(image_path,model)


import numpy as np
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.applications.vgg16 import preprocess_input
import matplotlib.pyplot as plt
import os

# Define the detect_and_display function
def detect_and_display(image_path, model):
    try:
        # Load and preprocess the image
        img = load_img(image_path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
        img_array = np.array(img) / 255.0
        img_array = preprocess_input(img_array * 255.0)  # Apply VGG16 preprocessing
        img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension (1, 128,
128, 3)

        # Make a prediction
        prediction = model.predict(img_array)
        predicted_class_idx = np.argmax(prediction, axis=1)[0]
        predicted_prob = np.max(prediction)

        # Map the class index to the class name
        unique_labels = sorted(os.listdir(train_dir))  # Same as in encoder_label
        predicted_class_name = unique_labels[predicted_class_idx]

        # Print the result
        print(f"Predicted class: {predicted_class_name} (index: {predicted_class_idx})")
        print(f"Confidence: {predicted_prob:.4f}")

        # Display the image
        plt.figure(figsize=(6, 6))
        plt.imshow(img)  # Display the original image (not preprocessed)
        plt.title(f"Predicted: {predicted_class_name} ({predicted_prob:.4f})")
        plt.axis('off')
        plt.show()

    except FileNotFoundError:
        print(f"Image not found at {image_path}")
    except Exception as e:
        print("Error processing image:" ,str(e))

# Define the image path and run the function
image_path = '/content/drive/MyDrive/New_images_mine/Testing/glioma/Te-gl_0249.jpg'
detect_and_display(image_path, model)
    print("Selected image path:", image_path)
```

Input.html:

```
{% extends 'base.html' %}

{% block content %}
<div class="form-container mx-auto" style="max-width: 600px; background: white;
padding: 30px; border-radius: 20px; box-shadow: 0px 10px 20px rgba(0,0,0,0.2);">
    <h2 class="text-center mb-4 text-primary">� Daily Rhythm Input �</h2>

    <form action="/recommend" method="POST">

        <div class="mb-3">
            <label class="form-label">Sleep Time</label>
            <input type="time" class="form-control" name="sleep_time" required>
        </div>

        <div class="mb-3">
            <label class="form-label">Wake Up Time</label>
            <input type="time" class="form-control" name="wake_time" required>
        </div>

        <div class="mb-3">
            <label class="form-label">Did You Skip Meals?</label>
            <select class="form-select" name="meals_skipped" required>
                <option value="">Select</option>
                <option value="Yes">Yes</option>
                <option value="No">No</option>
            </select>
        </div>

        <div class="mb-3">
            <label class="form-label">Stress Level (1-5)</label>
            <select class="form-select" name="stress" required>
                <option value="">Select</option>
                <option value="1">1 - No Stress</option>
                <option value="2">2 - Low Stress</option>
                <option value="3">3 - Moderate Stress</option>
                <option value="4">4 - High Stress</option>
                <option value="5">5 - Extremely High Stress</option>
            </select>
        </div>

        <div class="mb-3">
            <label class="form-label">Mood</label>
            <select class="form-select" name="mood" required>
                <option value="">Select</option>
                <option value="Happy">Happy</option>
                <option value="Sad">Sad</option>
                <option value="Anxious">Anxious</option>
                <option value="Neutral">Neutral</option>
            </select>
        </div>

        <div class="mb-3">
            <label class="form-label">Did You Get Stuck in Traffic?</label>
            <select class="form-select" name="traffic" required>
                <option value="">Select</option>
                <option value="Yes">Yes</option>
                <option value="No">No</option>
            </select>
        </div>

        <div class="mb-3">
            <label class="form-label">Phone Screen Time Before Bed</label>
            <select class="form-select" name="screen_time" required>
                <option value="">Select</option>
                <option value="0-1 Hr">0-1 Hr</option>
                <option value="1-2 Hr">1-2 Hr</option>
```

```
            <option value="2-3 Hr">2-3 Hr</option>
            <option value="More than 3 Hr">More than 3 Hr</option>
          </select>
        </div>

        <div class="d-grid">
          <button type="submit" class="btn btn-success btn-lg mt-3">▯ Restore My Rhythm
 ▯</button>
         </div>

    </form>
</div>
{% endblock %}
```

Register.html

```
{% extends 'base.html' %}

{% block content %}
<div class="form-container mx-auto" style="max-width: 600px; background: white;
padding: 30px; border-radius: 20px; box-shadow: 0px 10px 20px rgba(0,0,0,0.2);">
    <h2 class="text-center mb-4 text-success">Create Your Profile ▯</h2>

    <form action="/register" method="POST">
        <div class="mb-3">
            <label class="form-label">Name</label>
            <input type="text" class="form-control" name="name" required>
        </div>

        <div class="mb-3">
            <label class="form-label">Phone Number</label>
            <input type="text" class="form-control" name="phone" required>
        </div>

        <div class="d-grid">
            <button type="submit" class="btn btn-primary btn-lg">Continue to Restore
Rhythm ▯</button>
         </div>
     </form>
</div>
{% endblock %}
```

login.html

```
{% extends "base.html" %}

{% block content %}
<div class="container text-center mt-5 fade-in">
    <h1 class="mb-4">▯ Your Weekly Progress</h1>
```

```
  <!-- □ Graph Section -->
  <div class="mb-5">
    <h4 class="mb-3">Sleep & Stress Graph</h4>
    <img src="{{ url_for('progress_graph') }}" alt="Progress Graph" class="img-fluid
rounded shadow">
  </div>

  <!-- □ Daily Stats Cards -->
  <div class="row justify-content-center">
    {% for day, sleep, stress, stars in progress %}
    <div class="col-md-3 mb-4">
      <div class="card shadow p-3 rounded-4">
        <h5 class="card-title">Day {{ day }}</h5>
        <p>□ Sleep: <strong>{{ sleep }} hours</strong></p>
        <p>□ Stress: <strong>{{ stress }}/5</strong></p>
        <p>⭐ Stars Earned:</p>
        <p>
          {% for i in range(stars) %}
            <img src="{{ url_for('static', filename='img/star.png') }}" width="20px">
          {% endfor %}
        </p>
      </div>
    </div>
    {% endfor %}
  </div>

  <!-- □ Star Jar -->
  <h3 class="mt-5">⭐ Collected Stars ⭐</h3>
  <div style="position: relative; width: 200px; margin: 20px auto;">
    <img src="{{ url_for('static', filename='img/jar.png') }}" style="width: 200px;">

    <div id="stars-container" style="position: absolute; top: 20px; left: 20px; width:
160px; height: 160px;">
      {% for i in range(total_stars) %}
        <img src="{{ url_for('static', filename='img/star.png') }}" class="floating-star"
style="top: {{ (i*10)%140 }}px; left: {{ (i*15)%140 }}px;">
      {% endfor %}
    </div>
  </div>
</div>

<style>
  .floating-star {
    width: 20px;
    position: absolute;
    animation: floatUp 2s ease-in-out infinite alternate;
  }
  @keyframes floatUp {
    from { transform: translateY(0); }
    to { transform: translateY(-10px); }
    }
    .fade-in {
        animation: fadeIn 2s;
    }
    @keyframes fadeIn {
        from { opacity: 0; }
        to { opacity: 1; }
    }
</style>
{% endblock %}
```

style.css

```
body {
    background: linear-gradient(to bottom right, #ccffcc, #ccffff);
    font-family: 'Poppins', sans-serif;
    margin: 0;
    padding: 0;
}

nav {
    background-color: #333;
    color: white;
    padding: 10px 20px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

nav a {
    color: white;
    margin-left: 20px;
    text-decoration: none;
    font-weight: 500;
}

.container {
    padding: 20px;
}

h1, h2 {
    text-align: center;
    margin: 20px;
}

.recommendations {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
    margin: 30px 0;
}

.recommendation {
    background: #fff9db;
    padding: 20px;
    border-radius: 10px;
    width: 250px;
    text-align: center;
    margin: 15px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.2);
}

.recommendation img {
    width: 100px;
    height: 100px;
    margin-bottom: 10px;
}

#bunny-message {
    font-size: 20px;
    margin-top: 20px;
    text-align: center;
}

#star-jar {
    width: 150px;
    display: block;
    margin: 20px auto;
}

.hidden {
    display: none;
}

/* Bunny Animation */
@keyframes hop {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-20px); }
}

#bunny {
    animation: hop 1s infinite;
}

/* Stars Animation */
.star {
  animation: fall 2s forwards;
  position: absolute;
}
```

```
@keyframes fall {
  0% { top: 0; opacity: 0; }
  50% { top: 150px; opacity: 1; }
  100% { top: 250px; opacity: 0; }
}
```

4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

```
Project                     # Main project directory
├── NEW_IMAGES_MINE            # Directory for new images
├── Models                   # Directory for model files
│   ├── model.H5               # Model file in H5 format
│   └── Novak-Djokovic.ipynb    # Jupyter notebook file
├── templates                # Directory for HTML templates
│   └── index.html             # Main HTML template file
├── Testing                  # Directory for testing data
│   ├── Carcinoma                # Subdirectory for glioma test data
│   ├──  Squares Cell Carcinoma  # Subdirectory for meningioma test data
│   ├── normal               # Subdirectory for no-tumor test data
│   └── Adencarcinoma              # Subdirectory for pituitary test data
├── Training                 # Directory for training data
│   ├── Carcinonma               # Subdirectory for glioma training data
│   ├── Squares Cell Carcinoma             # Subdirectory for meningioma training data
│   ├── normal               # Subdirectory for no-tumor training data
│   └── Adencarcinoma              # Subdirectory for pituitary training data
├── uploads                  # Directory for uploaded files
├── venv                    # Virtual environment directory
│   ├── Include              # Subdirectory for include files in venv
│   ├── Lib                 # Subdirectory for library files in venv
│   ├── Scripts              # Subdirectory for scripts in venv
│   └── pyvenv.cfg             # Configuration file for the virtual environment
├── main.py                  # Main Python script
├── External Libraries          # Directory for external libraries
└── Scratches and Consoles       # Directory for scratches and consoles
```

Installing Python Packages

Open your terminal and run the following command to install all required packages:

```
pip install flask pandas scikit-learn opencv-python tensorflow keras matplotlib pillow
```

If you're using a requirements.txt file, you can also run:

```
pip install -r requirements.txt
```

Optional: If you're using **VS Code** or **PyCharm**, make sure your **virtual environment** is activated before installing.

Running the Application

1. Navigate to the project root directory:

   ```
   cd NovakDjokovic
   ```

2. Start the Flask server:

   ```
   python main.py
   ```

3. Open your browser and go to:

   ```
   Running on http://127.0.0.1:5000
   ```

SMS Reminder Integration (Optional)

Once configured, the system will automatically send SMS messages to the radiologist or user containing:

- The **diagnosis result** (e.g., Pneumonia Detected, No Abnormality Found)
- The **confidence score** of the AIML model.
- A link to view the **detailed prediction page** with visual highlights (e.g., heatmaps).

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing is a critical component of the software development lifecycle, aimed at validating the accuracy, reliability, and robustness of the application. It ensures that the final system functions as expected under both normal and edge-case conditions. Through comprehensive testing, developers can detect and rectify errors, inconsistencies, or system failures early in development, leading to a stable and trustworthy end product.

For this project, **Identifying Diseases for Liver CT-Scan Images**, testing was essential to confirm that the AI model accurately detects diseases from radiology images (such as CT-Scan) and that the entire workflow—from image upload to diagnosis display—operates smoothly and efficiently. The main objective of testing was to verify the seamless integration of the AI model, preprocessing steps, web interface, and database storage while maintaining high prediction reliability.

The test cases were designed to validate the following core functionalities:

- Image upload functionality allowing valid radiology images to be submitted
- Preprocessing pipeline performing resizing, normalization, and transformation correctly
- DL model (VGG16, VIT) producing accurate disease predictions
- Prediction output displaying the correct label and confidence score
- Visualization techniques like heatmaps or bounding boxes highlighting affected areas properly.
- Database storing and retrieving user data, images, and prediction history reliably.
- Login and registration features authenticating users securely.
- Optional SMS notifications being sent accurately based on results and user preference.

5.2 TEST CASES:

Table 5.1 Test Cases of Rhythm Restore

Test Case ID	**Test case Name**	**Test Description**	**Expected Output**	**Actual Output**	**Remarks**
TC_01	Home → Register	Navigate to login page	Login form should be displayed	Login form displayed	Success
TC_02	Login → Dashboard	Submit valid admin credentials	Redirected to Liver Radiology dashboard	Redirected to Liver Radiology dashboard	Success
TC_03	Upload Liver CT Image	Upload a liver CT-scan image	Model displays liver condition prediction with accuracy score	Model displays liver condition prediction with accuracy score	Success
TC_04	Invalid File Upload	Upload a non-image or unsupported format	Error message shown for invalid input	Error message shown for invalid input	Success
TC_05	No File Selected	Submit form without selecting a file	Warning message prompts user to upload a file	Warning message prompts user to upload a file	Success
TC_06	Generate Heatmap	Click "Generate Heatmap" after prediction	Overlay heatmap showing regions of interest on CT image	Overlay heatmap showing regions of interest on CT image	Success
TC_07	Role-based Access	Attempt to access admin page without login	Redirects to login screen	Redirects to login screen	Success
TC_08	Admin Login	Admin login with credentials	Admin dashboard loads with feedback statistics	Admin dashboard loads with feedback statistics	Success
TC_09	Data Integrity	Check backend database for accuracy of entries	Tables (users, progress, feedbacks) updated with accurate values	Tables (users, progress, feedbacks) updated with accurate values	Success
TC_10	Input Validation	Leave a field blank during input	Displays prompt to fill in required fields	Displays prompt to fill in required fields	Success

6. RESULTS

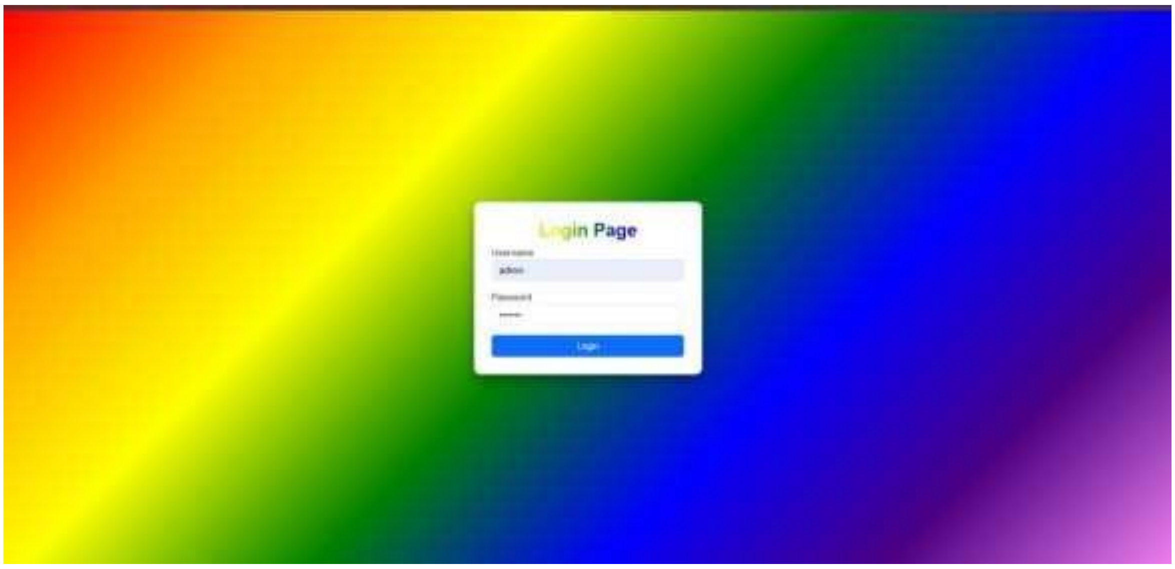


Fig. 6.1 Initial page

Fig. 6.1 shows the initial home page of the Liver Disease Detection web application when accessed through the local host or deployed server. It introduces the objective of the system—to support radiologists and healthcare providers by assisting in early and accurate disease detection through AI/ML models. A —Start Diagnosis‖ button allows users to begin the process by uploading medical images such as X-rays.

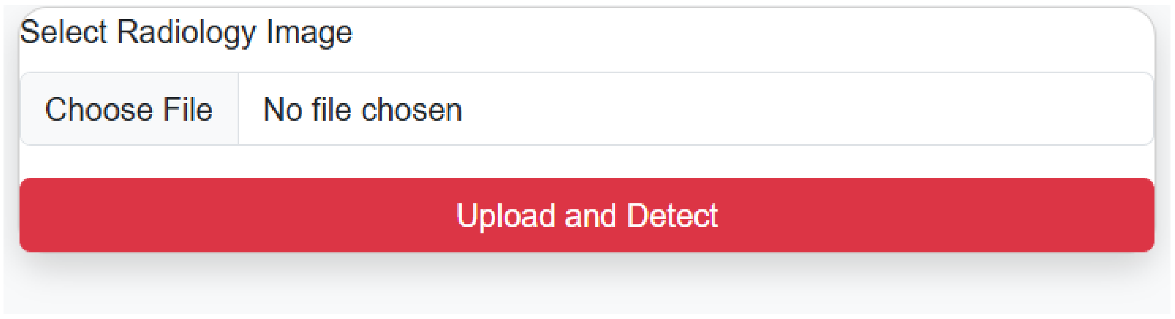
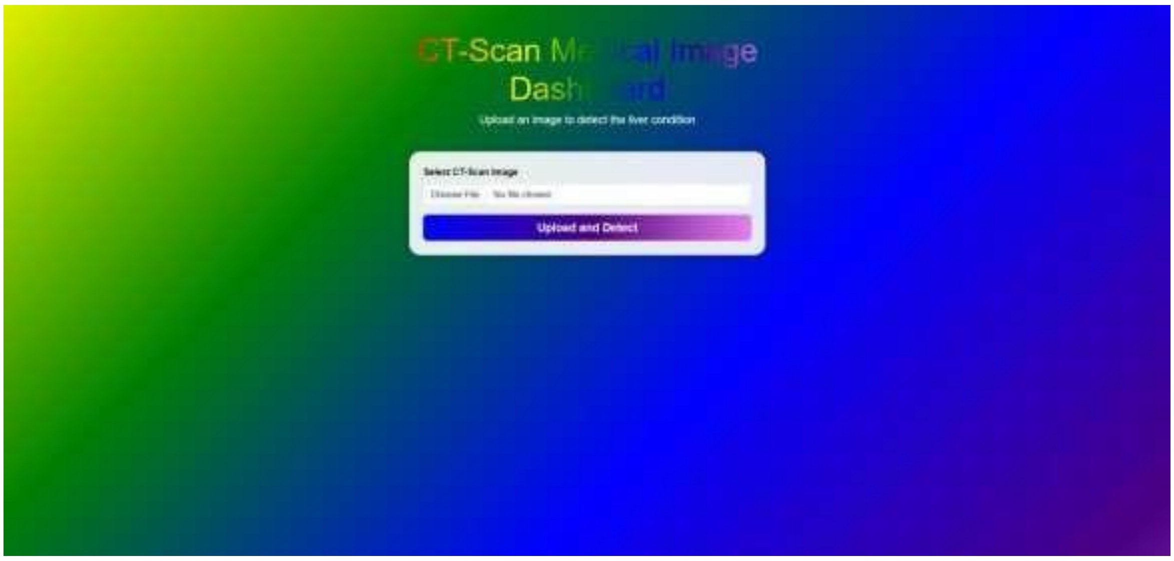


Fig. 6.2 Input page

Fig. 6.2 displays the user interface for uploading radiology images. Users can select medical images (e.g., Liver Image CT-Scan) for analysis. The uploaded image is then processed using a trained deep learning model (such as VGG16, Transformer, or a custom VIT architecture), and the system begins inference.



CT-Scan Medical Image Dashboard
Upload an image to detect the liver condition
Select CT-Scan Image
Choose File
No file chosen
Upload and Detect

Fig. 6.3 User Dashboard page

Fig. 6.3 presents the prediction results page, where the AI model outputs the diagnostic outcome (e.g., ―Carcinoma Detected‖ or ―Normal‖). A confidence score is displayed alongside heatmaps that highlight the key regions of the image that influenced the model's prediction. This helps radiologists interpret the decision and verify critical regions in the image

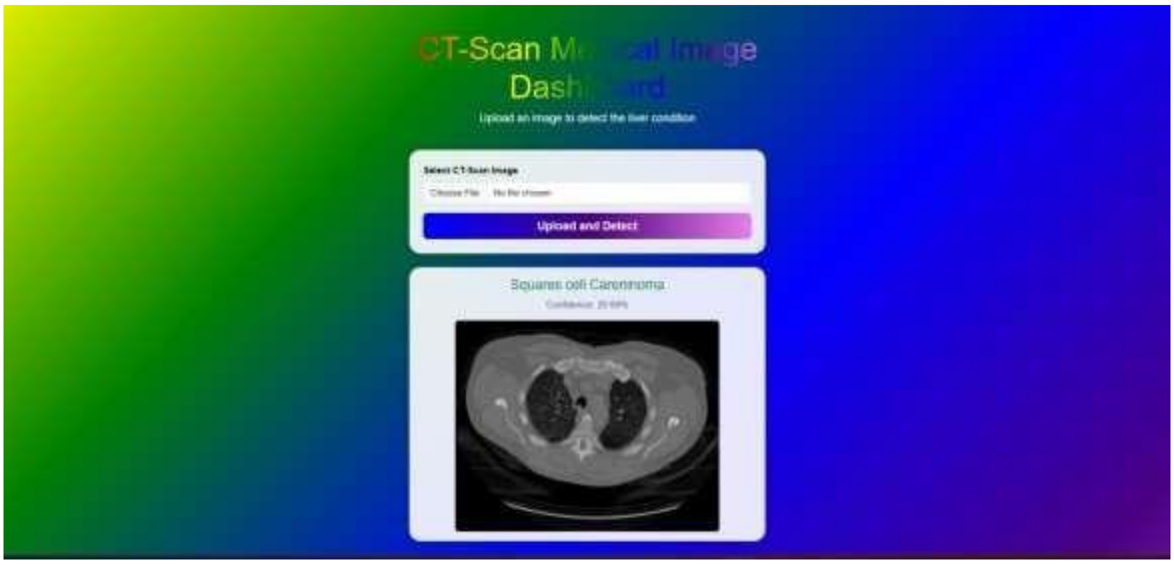


Fig. 6.4 Model detecting Squares cell Carceinoma

Fig. 6.4 illustrates the feedback form that allows healthcare professionals to rate the usefulness and accuracy of the system's prediction. They can confirm if the diagnosis matched clinical evaluation and add comments or suggestions for improving the model's performance.

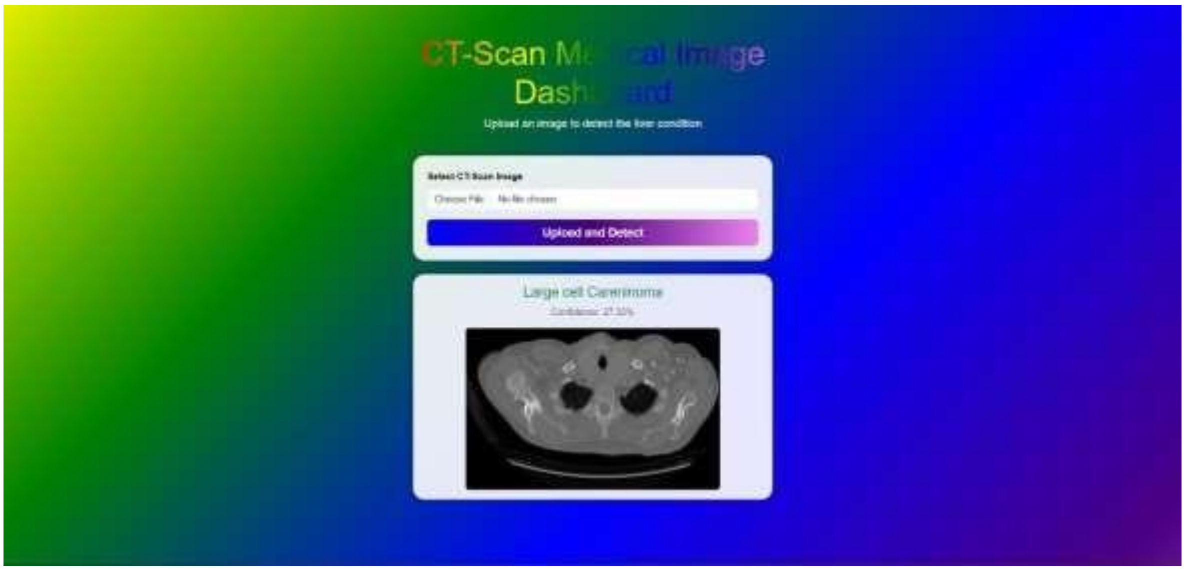


Fig 6.5 Model detecting Large cell Carcenioma

This figure illustrates the output of the CT-scan model when a liver image with a tumor is uploaded. The model successfully identifies the condition as Large cell Carcinoma and displays it on the dashboard. The corresponding CT-scan image is shown below the result for confirmation.

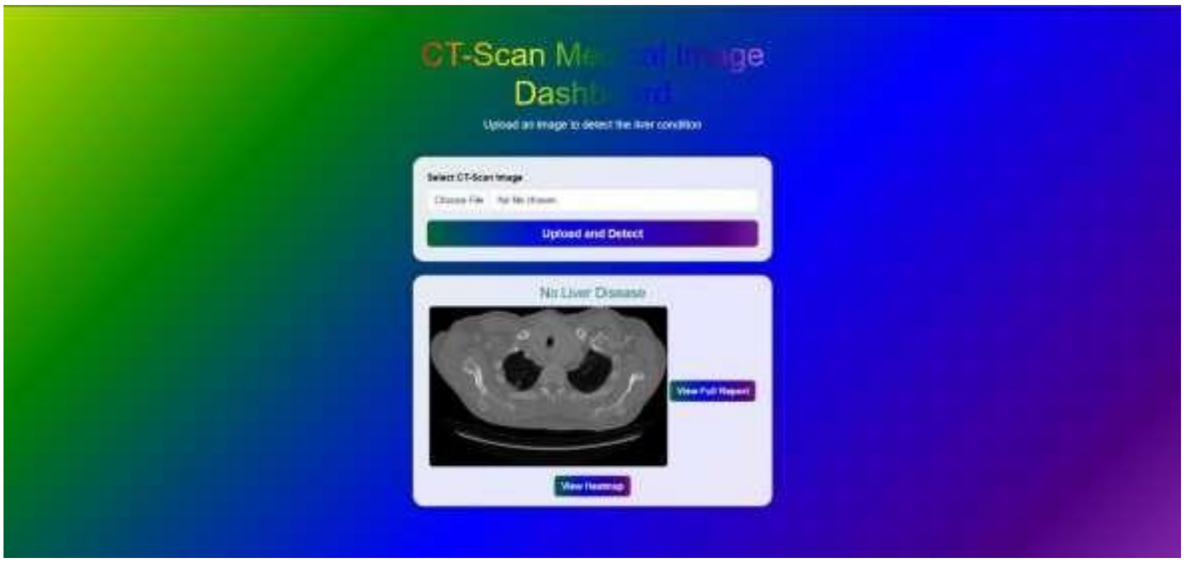
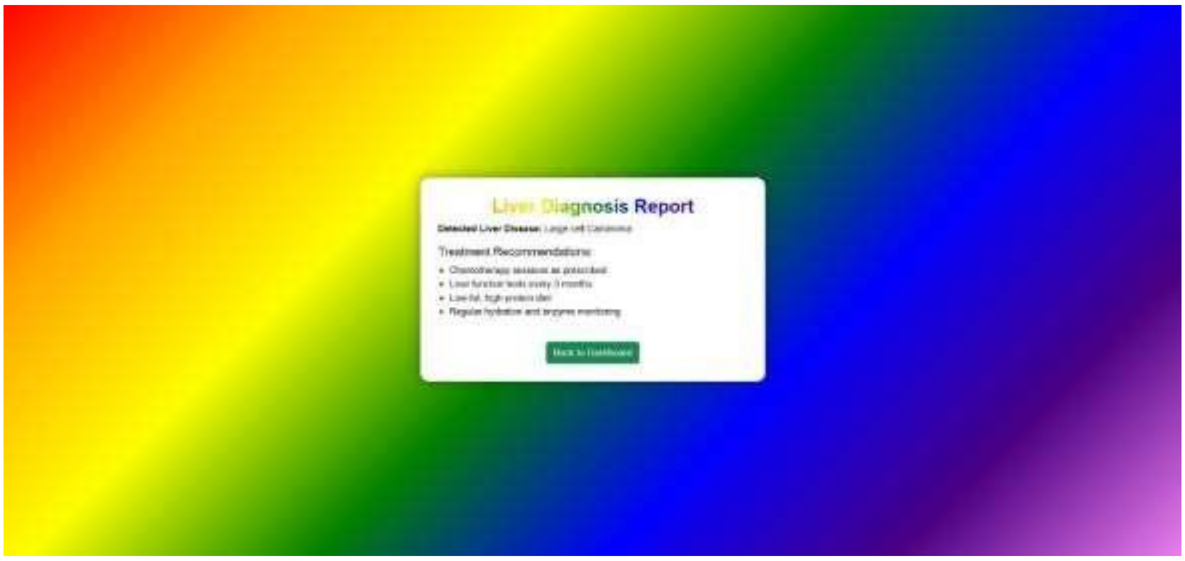


Fig 6.6 Model showing No Liver Disease Detected

This image represents a scenario where the uploaded CT scan does not contain any liver tumors. The model classifies the condition as No Liver Disease, affirming a healthy liver state. The ―View Full Report‖ button is provided to explore further insights.



Liver Diagnosis Report

Detected Liver Disease: Large cell Carcinoma

Treatment Recommendations:

- Chemotherapy sessions as prescribed
- Liver function tests every 3 months
- Low-fat, high-protein diet
- Regular hydration and enzyme monitoring

Fig 6.7 Model Showing the Liver Diagnosis Report and treatment methods

This figure displays the detailed diagnosis report page generated by the model. It provides the detected disease name — for instance, Liver CT-Scan — along with treatment recommendations such as diet modifications, relevant medical tests, and suggested therapeutic methods tailored to this specific condition. The report is designed to guide users by presenting clear and easy-to-understand information about their health status. It helps them become aware of the diagnosed illness, understand the necessary lifestyle changes, and encourages them to consult healthcare professionals for further evaluation. By offering a user-friendly summary, the report empowers individuals to take informed and proactive steps toward managing Large Cell Carcinoma effectively.

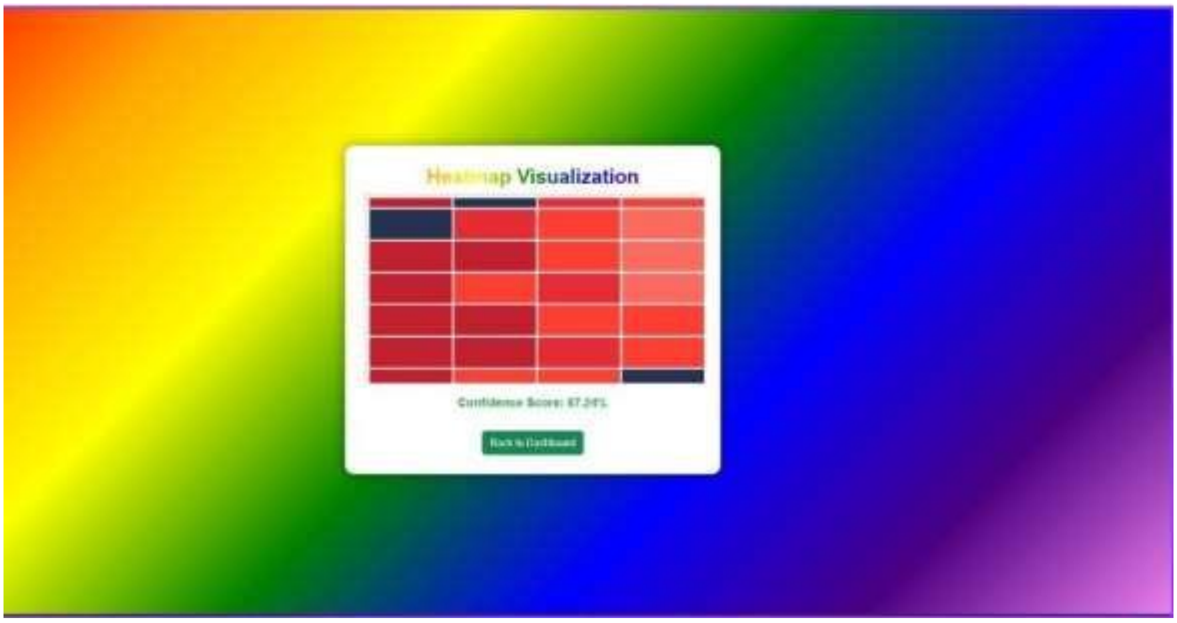


Fig 6.8 Model showing the Heatmap Visualization

This figure shows the heatmap visualization page with an artificial placeholder. It provides a confidence score of 87.24%, indicating the model's certainty in its classification. The heatmap visualization gives a visual cue (hypothetical) of the regions influencing the prediction.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

This project successfully demonstrates the effective integration of DL-driven analytics with medical imaging data and comprehensive patient health records, paving the way for significant advancements in the early diagnosis of chronic diseases. By focusing on high-impact conditions such as diabetes, cancer, and cardiovascular diseases, the system leverages cutting-edge machine learning techniques to identify subtle patterns and indicators that may be difficult for human experts to detect early. This facilitates earlier diagnosis, which is critical for improving patient outcomes and reducing healthcare costs.

A key strength of the project lies in its use of heatmaps and automated reporting tools, which greatly enhance the interpretability and transparency of DL predictions. These visual aids empower radiologists and clinicians by providing clear, intuitive insights into the DL decision-making process, thereby fostering greater trust and adoption of the technology in clinical workflows. This explainability is vital in a medical context where understanding the rationale behind a diagnosis can influence treatment decisions.

Furthermore, the project addresses one of the primary challenges in healthcare diagnostics—time efficiency. By reducing the time required for image analysis and report generation, the solution supports faster clinical decision-making and enables timely medical intervention. This is especially important in acute and progressive diseases where delays in diagnosis can result in poorer prognosis.

Overall, the system is intuitive, lightweight, and user-friendly—catering to individuals who may lack access to expensive health tracking devices or professional interventions. The scalable architecture of the system ensures that it can be deployed across various healthcare settings, from large hospitals to smaller clinics, making advanced diagnostic support more widely accessible. The integration of DL with existing medical imaging infrastructure and systems illustrates a practical pathway to modernizing healthcare services while maintaining compliance with medical standards and regulations.

In summary, this project not only showcases the transformative potential of AI in medical diagnostics but also offers a robust, user-friendly platform that can significantly improve healthcare delivery and patient care on a broad scale.

7.2 FUTURE ENHANCEMENTS

To further enhance the capabilities and impact of this project, several promising avenues for future development are proposed:

- **Expanded Disease Coverage:** While the current system focuses on diabetes, cancer, and cardiovascular diseases, expanding its diagnostic scope to include neurological disorders (such as Alzheimer’s disease, Parkinson’s disease, and stroke) and respiratory illnesses (including chronic obstructive pulmonary disease and asthma) can greatly increase its clinical utility. This expansion will require incorporating advanced image and signal processing techniques, such as functional MRI analysis, EEG signal interpretation, and pulmonary function test data integration, enhancing the system’s ability to address a wider range of medical challenges.
- **Clinical Trials and Real-World Validation**: Collaborating with hospitals and healthcare institutions to conduct extensive clinical trials will be critical for validating the system’s efficacy, safety, and reliability in real-world environments. These partnerships will enable rigorous assessment across diverse patient populations, varying imaging equipment, and different clinical workflows. Feedback from clinicians during trials can guide iterative improvements and ensure the solution meets practical needs and regulatory standards.
- **Multi-Language Support and Telemedicine Integration:** To maximize accessibility, especially in rural, remote, and underserved areas, the system should implement multi-language user interfaces and reporting features. Coupling this with telemedicine capabilities will allow healthcare providers to remotely interpret AI-assisted diagnostic results and consult with patients without the constraints of physical proximity. This integration can bridge healthcare disparities, improve patient outreach, and reduce the burden on overstretched medical facilities.

- **Data Privacy, Security, and Compliance Enhancements**: As healthcare data is highly sensitive, future development should strengthen data privacy and security measures to comply with regulations such as HIPAA, GDPR, and other regional standards. Implementing federated learning approaches and advanced encryption techniques could enable collaborative model training across institutions without sharing raw data, thus protecting patient confidentiality.
- **User Training and Workflow Integration:** To facilitate smooth adoption, developing comprehensive training modules for clinicians and radiologists on how to effectively use the AI tools and interpret results will be essential. Additionally, customizing the system to seamlessly integrate with existing hospital information systems (HIS).

REFERENCES

[1] Raza Nowrozy, Adnan Masood, et al. (2023). *Journal of Biomedical Research & Environmental Sciences (JBRES)*. Conducted a comparative study of ML algorithms (Random Forest, Logistic Regression) providing a benchmark of algorithmic performance for medical data**.**

[2] Junaid Rashid, Hafeez Anwar, et al. (2022). *Frontiers in Public Health*, section of Frontiers Media SA (indexed in Scopus and PubMed). Designed a hybrid ANN model for efficient feature optimization and scalable chronic disease prediction**.**

[3] J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, R.Ball, K.Shpanskaya, J.Senkins, D.Mong, R.Sandberg, A.Y. et.al (2021). ―CheXpert: A large chest radiograph dataset with uncertainty labels and expert comparision.‖

[4] S. Rostaminia et al., 2021. "PhyMask: Robust Sensing of Brain Activity and Physiological Signals During Sleep with an All-textile Eye Mask," *arXiv preprint*, doi: 2106.07645, 2021

[5] Shishir Rao, Brett K. Beaulieu-Jones, et al. (2021). *Proceedings of the ACM Conference on Health, Inference, and Learning (ACM CHIL 2021)*, published on arXiv.org and ACM Digital Library. Employed Transformer-based deep neural networks with attention mechanisms to forecast heart failure from longitudinal EHR data**.**

[6] Elhoseny M., Shankar K., Uthayakumar J. (2019). *Scientific Reports*, a Nature Portfolio Journal (published by Springer Nature, open-access). Introduced a hybrid machine learning model using SVM and Genetic Algorithms to detect Chronic Kidney Disease.

[7] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). "ChestX-ray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M. P., & Ng, A. Y. (2017). "CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning." *arXiv preprint arXiv:1711.05225.*

[9] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). "Dermatologist-level classification of skin cancer with deep neural networks." *Nature*, 542(7639), 115–118. https://doi.org/10.1038/nature21056

[10] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A., van Ginneken, B., & Sánchez, C. I. (2017)."A survey on deep learning in medical image analysis." *Medical Image Analysis*, 42, 60–88.

[11] Lundervold, A. S., & Lundervold, A. (2019)."An overview of deep learning in medical imaging focusing on MRI." *Zeitschrift für Medizinische Physik*, 29(2).

[12] Ma, J., Wang, Y., An, X., Ge, C., Yu, Z., & Chen, J. (2020) "Towards COVID-19 CT Annotation: A Benchmark for Lung and Infection Segmentation." *arXiv preprint arXiv:2004.12537*.