

**A MAJOR PROJECT**

**On**

**MACHINE LEARNING BASED CLASSIFICATION OF SHOULDER  
IMPLANT X RAYS FOR MANUFACTURER IDENTIFICATION**

*Submitted*

*In partial fulfillment of the requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**Computer Science and Engineering (Data Science)**

**By**

**BILAVATH SHIVA**

**21641A67E8**

**BASHABOYINA SAI GAGAN**

**21641A67E1**

**VUGGE PALLAVI**

**21641A78C8**

**MACHARLA SAI KIRAN**

**21641A67H5**

*Under the Guidance of*

**Mr. SAYYED HASANODDIN**

Assistant Professor, Department of CSE (Data Science).



**Department of Computer Science & Engineering (Data science)**

**Vaagdevi College of Engineering**

(UGC Autonomous, Accredited by NAAC with “A”)

Bollikunta, Khila Warangal (Mandal), Warangal Urban – 506005(T.S)

**(2021-2025)**

**VAAGDEVI COLLEGE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**(DATA SCIENCE)**

(UGC Autonomous, Accredited by NBA, Accredited by NAAC with “A”)  
Bollikunta, Khila Warangal (Mandal), Warangal Urban –506005(T.S)



**CERTIFICATE**

This is to certify that the major project entitled “***MACHINE LEARNING BASED CLASSIFICATION OF SHOULDER IMPLANT X RAYS FOR MANUFACTURER IDENTIFICATION***” is submitted by **B. SHIVA (21641A67E8), B. SAI GAGAN (21641A67E1), V. PALLAVI (21641A67C8), M. SAI KIRAN (21641A67H5)** in partial fulfillment of the requirements for the award of the Degree in Bachelor of Technology in Computer Science and Engineering (Data Science) during the academic year 2024-2025.

**Project Guide:**

Mr. Sayyed Hasanoddin

**Head of the Department:**

Dr. Ayesha Banu

**External Examiner**

## DECLARATION

We declare that the work reported in the project entitled “MACHINE LEARNING BASED CLASSIFICATION OF SHOULDER IMPLANT X RAYS FOR MANUFACTURER IDENTIFICATION” is a record of work done by us in the partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering (Data Science), VAAGDEVI COLLEGE OF ENGINEERING (Autonomous), Affiliated to JNTUH, Accredited By NBA, under the guidance of **Mr. Sayyed Hasanoddin**, Assistant Professor of CSE (DS). We hereby declare that this project work bears no resemblance to any other project submitted at Vaagdevi College of Engineering or any other university/college for the award of the degree.

Bilavath Shiva (21641A67E8)

Bashaboyina Sai Gagan (21641A67E1)

Vugge Pallavi (21641A67C8)

Macharla SaiKiran (21641A67H5)

## ACKNOWLEDGEMENT

The development of the project though it was an arduous task, it has been made by the help of many people. We are pleased to express our thanks to the people whose suggestions, comments, criticisms greatly encouraged us in betterment of the project.

We would like to express our sincere gratitude and indebtedness to my project Guide **Mr. Sayyed Hasanoddin**, Assistant Professor, CSE (DS) for his valuable suggestions and interest throughout the completion of this project.

We are also thankful to Head of the Department **Dr. Ayesha Banu**, Associate Professor, CSE (DS) for providing excellent support in completing the project successfully.

We would like to express our sincere thanks and profound gratitude to **Dr. K. Prakash**, Principal of Vaagdevi College of Engineering, for his support, guidance and encouragement in the course of our project.

We are also thankful to Project Coordinators, for their valuable suggestions, encouragement and motivations for completing this project successfully.

We are thankful to all other faculty members for their encouragement.

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

Bilavath Shiva (21641A67E8)

Bashaboyina Sai Gagan (21641A67E1)

Vugge Pallavi (21641A67C8)

Macharla SaiKiran (21641A67H5)

## **ABSTRACT**

Recent advancements in medical imaging and machine learning have facilitated automated classification of orthopaedic implants, improving accuracy and efficiency. Shoulder implants are used in orthopaedic surgeries, and their identification is crucial for revision procedures and post-operative assessments. Studies show that over 250,000 shoulder replacements are performed annually in the U.S., with implant misidentification contributing to 30% of revision complications. Traditional manual classification by radiologists is time-consuming and prone to errors, with accuracy rates varying between 65% and 80%, depending on expertise. This study proposes a machine learning-based approach for the automated classification of shoulder implant X-rays to identify their manufacturers, reducing dependency on manual efforts. The dataset consists of labeled X-ray images of shoulder implants categorized by manufacturer, with labels representing different implant brands and models. The preprocessing pipeline involves contrast enhancement, noise reduction, and resizing to ensure consistency. To address data imbalance, Synthetic Minority Over-sampling Technique (SMOTE) is employed, ensuring equal representation across classes. The dataset is then split into 80% training and 20% testing for model evaluation. We compare an existing Support Vector Machine (SVM) classifier, which has limitations in scalability and performance on imbalanced data, with a proposed Random Forest Classifier (RFC) model. The RFC outperforms SVM by leveraging multiple decision trees, reducing overfitting, and improving classification accuracy. Experimental results demonstrate that the RFC model achieves a significantly higher accuracy, precision, and recall, proving its effectiveness in real-world applications. The proposed method offers a robust, automated solution for implant identification, aiding radiologists and orthopaedic surgeons in streamlining patient care and implant tracking.

## **TABLE OF CONTENTS**

	Page No
<b>ABSTRACT</b>	<b>ii</b>
<b>TABLE OF CONTENTS</b>	<b>iii</b>
<b>CHAPTER 1 INTRODUCTION</b>	
<b>1.1 OVERVIEW</b>	<b>01</b>
<b>1.2 MOTIVATION</b>	<b>01</b>
<b>1.3 PROBLEM STATEMENT</b>	<b>02</b>
<b>1.4 RESEARCH OBJECTIVES</b>	<b>02</b>
<b>1.5 SIGNIFICANCE</b>	<b>03</b>
<b>1.6 APPLICATIONS</b>	<b>04</b>
<b>CHAPTER 2 LITERATURE SURVEY</b>	<b>05</b>
<b>CHAPTER 3 TRADITIONAL SYSTEM</b>	
<b>3.1 RADIOLOGIST-BASED VISUAL INSPECTION</b>	<b>08</b>
<b>3.2 IMPLANT MANUFACTURER REFERENCE</b>	
<b>CATALOGUES AND MANUAL MATCHING</b>	<b>08</b>
<b>3.3 PATIENT RECORDS AND SURGICAL</b>	
<b>DOCUMENTATION REVIEW</b>	<b>09</b>
<b>3.4 LIMITATIONS</b>	<b>10</b>
<b>CHAPTER 4 PROPOSED SYSTEM</b>	
<b>4.1 OVERVIEW</b>	<b>11</b>
<b>4.2 DATASET PREPROCESSING</b>	<b>13</b>
<b>4.3 SMOTE</b>	<b>15</b>
<b>4.4 TRAIN-TEST SPLITTING</b>	<b>16</b>
<b>4.5 ML MODEL BUILDING</b>	<b>17</b>

<b>CHAPTER 5</b>	<b>UML DIAGRAMS</b>	
5.1	CLASS DIAGRAM	23
5.2	SEQUENCE DIAGRAM	24
5.3	ACTIVITY DIAGRAM	25
5.4	DATA FLOW DIAGRAM	26
5.5	COMPONENT DIAGRAM	27
5.6	USE CASE DIAGRAM	28
5.7	DEPLOYMENT DIAGRAM	29
<b>CHAPTER 6</b>	<b>SOFTWARE AND HARDWARE REQUIREMENTS</b>	<b>30</b>
<b>CHAPTER 7</b>	<b>FUNCTIONAL REQUIREMENTS</b>	<b>35</b>
<b>CHAPTER 8</b>	<b>SOURCE CODE</b>	<b>37</b>
<b>CHAPTER 9</b>	<b>TESTING</b>	<b>52</b>
<b>CHAPTER 10</b>	<b>RESULT AND DISCUSSION</b>	
10.1	IMPLEMENTATION DESCRIPTION	56
10.2	RESULTS	59
<b>CHAPTER 11</b>	<b>CONCLUSION AND FUTURE SCOPE</b>	<b>68</b>
<b>REFERENCES</b>		<b>69</b>
<b>PUBLISHED PAPER</b>		<b>71</b>

# CHAPTER 1

## INTRODUCTION

### 1.1 Overview

Health system efficiency has been a growing concern over the years, with global healthcare expenditures increasing at an alarming rate. In 2018, global healthcare spending reached \$8.3 trillion, and by 2023, this number soared to over \$12 trillion. This massive rise in spending is attributed to aging populations, the prevalence of chronic diseases, and advancements in medical technologies, which often demand more resources and higher costs. Yet, despite the increased investments, many health systems across the world continue to face inefficiencies in service delivery, patient care, and resource management.

For instance, the World Health Organization (WHO) estimates that up to 20-40% of healthcare resources are wasted through inefficiencies, which could range from delays in care delivery to redundant or unnecessary procedures. The complexity of managing health data, particularly with the rise of electronic health records (EHRs), has added to the burden, as outdated systems and fragmented processes prevent seamless data integration and real-time decision-making. These inefficiencies not only drive-up costs but also impede the ability to provide timely and high-quality care, highlighting the need for more innovative and adaptive solutions, such as the integration of machine learning in healthcare systems.

### 1.2 Motivation

The motivation for this research arises from the critical need to enhance diagnostic accuracy and operational efficiency in the medical field, particularly in orthopedic implant management. Shoulder implant surgeries have become increasingly common, and with them comes the challenge of accurately identifying implant manufacturers from X-ray images—a task that is both time-consuming and prone to human error when performed manually. In clinical settings, misidentification of implant components can lead to improper post-operative care, delayed intervention in case of complications, and challenges in managing implant recalls.

Advancements in machine learning and computer vision offer a promising avenue to automate this process, thereby reducing reliance on subjective human judgment and streamlining diagnostic workflows. By leveraging algorithms such as the Support Vector Classifier (SVC) and the Random Forest Classifier (RFC), this research aims to develop a system capable of accurately classifying shoulder implant X-rays based on pixel-level features. The project not



only explores the efficacy of these models in handling high-dimensional image data but also compares their performance to determine the most effective approach.

Furthermore, the integration of a user-friendly graphical interface ensures that the solution is accessible to both hospital administrators and patients, facilitating smoother adoption in real-world scenarios. Ultimately, this research is driven by the goal of improving patient outcomes through more accurate, rapid, and automated implant identification, thereby setting the stage for enhanced diagnostic procedures and safer clinical practices.

### **1.3 Problem Statement**

Manual approaches in healthcare management, such as paper-based systems and disjointed digital platforms, are fraught with inefficiencies that hinder the overall performance of health systems. Traditional methods for patient record-keeping, appointment scheduling, and resource allocation are prone to human error, data loss, and significant delays. For example, when medical records are kept manually or in disparate systems, it becomes challenging to access accurate patient histories or update critical data in real-time. This can result in duplicated tests, delayed diagnoses, and compromised patient safety. Moreover, the manual entry and processing of data consume a large portion of healthcare professionals' time, diverting attention away from direct patient care.

The need for automation is clear in addressing these inefficiencies. Automated systems, driven by advanced software engineering techniques, can streamline data entry, ensure real-time updates across departments, and facilitate seamless data sharing. Machine learning models, for instance, can predict patient no-shows, optimize appointment schedules, and recommend resource allocation based on historical data. By eliminating the need for manual processes, automation not only improves operational efficiency but also enhances patient outcomes by allowing healthcare professionals to focus more on care delivery rather than administrative burdens.

### **1.4 Research Objectives**

The primary research objective of this project is to develop and evaluate a machine learning-based system that accurately classifies shoulder implant X-ray images for manufacturer identification. This involves designing a robust image preprocessing pipeline, implementing and comparing two different classification algorithms (SVC and RFC), and integrating the best-

performing model into an intuitive, role-based GUI application for both hospital administrators and patients.

## **1.5 Significance**

This project holds significant value in the intersection of medical imaging and machine learning by addressing a critical need for accurate and efficient classification of shoulder implant X-ray images. The ability to reliably identify the manufacturer of an implant not only assists in ensuring appropriate post-operative care and monitoring but also aids in the rapid diagnosis of implant-related complications. By leveraging machine learning algorithms, the system reduces the dependence on manual image analysis, which is both time-consuming and prone to human error. The integration of advanced preprocessing techniques—such as image resizing, flattening, and class balancing using SMOTE—ensures that the input data is optimized for robust model performance. Moreover, the dual-model approach, comparing SVC and RFC, highlights the importance of selecting the appropriate algorithm for high-dimensional image data. The superior performance of the Random Forest Classifier in this context underscores the potential for ensemble methods in medical diagnostics. Overall, the project not only improves diagnostic accuracy but also streamlines the workflow in clinical environments, ultimately contributing to better patient outcomes and more efficient hospital operations.

## **1.6 Applications**

### **1. Patient Scheduling and Appointment Management:**

- Machine learning algorithms can predict patient behavior, such as the likelihood of appointment no-shows, allowing health systems to optimize scheduling and reduce idle time. This results in better utilization of medical professionals' time and shorter wait times for patients.

### **2. Resource Allocation and Optimization:**

- Advanced software systems, integrated with machine learning, can analyze hospital resource usage (such as bed availability, staff workload, and medical supplies) in real-time, ensuring optimal allocation of resources. This minimizes wastage and ensures that resources are directed to where they are needed most.

### **3. Predictive Analytics for Disease Management:**

- By analyzing patient data, machine learning models can help in predicting the onset or progression of diseases. This enables early intervention for high-risk patients, potentially reducing the severity of illnesses and the associated healthcare costs.

### **4. Natural Language Processing (NLP) for Medical Records:**

- NLP tools can automatically extract relevant information from medical records, reducing the time doctors spend on documentation. This improves the accuracy of medical data, enhances communication between healthcare professionals, and supports more personalized care plans for patients.

### **5. Telemedicine and Remote Monitoring:**

- With the rise of telehealth, software systems can integrate remote patient monitoring devices with machine learning algorithms to analyze health data in real-time. This enables doctors to make timely interventions for patients with chronic conditions or those recovering from surgery, improving patient outcomes while reducing the need for hospital visits.

## **CHAPTER 2**

### **LITERATURE SURVEY**

Lyon et al. (2021) [1] explored the potential of AI-driven optimization in the healthcare diagnostic process. Their study emphasized the significant role of AI in enhancing diagnostic accuracy and reducing human error, which traditionally slows down the process. By employing machine learning algorithms, the study highlighted improvements in decision-making, leading to more timely and accurate diagnoses. However, challenges such as data integration and system interoperability remain areas for improvement.

Tripathi et al. (2021) [2] examined the evolving role of big data and AI in drug discovery. The authors discussed how AI-powered tools have accelerated the drug discovery process by analyzing vast datasets and identifying potential drug candidates more efficiently than traditional methods. While AI presents numerous advantages, the study also pointed out limitations in data handling and the complexity of integrating AI systems into existing pharmaceutical frameworks.

Khan et al. (2023) [3] discussed the drawbacks of AI in the healthcare sector, emphasizing challenges such as data privacy, bias in AI algorithms, and the high cost of implementing AI solutions. The paper suggested potential solutions, including more robust data governance frameworks and continuous algorithm validation to minimize bias and improve trust in AI-driven healthcare solutions.

Dileep and Gianchandani (2022) [4] focused on the use of AI in breast cancer screening and diagnosis. The study demonstrated how AI could improve early detection rates and reduce false positives in mammograms, leading to better patient outcomes. The authors also highlighted the need for ongoing training of AI models to account for new medical data and maintain high levels of diagnostic accuracy.

Chandrashekar et al. (2020) [5] introduced a deep learning approach to generate contrast-enhanced CT angiograms without the need for intravenous contrast agents. The study underscored the potential of AI to reduce patient risk by minimizing exposure to contrast agents, while still maintaining high-quality imaging for diagnostic purposes. This innovation marked a significant step forward in non-invasive imaging techniques.

William et al. (2018) [6] assessed the accuracy of an AI-driven algorithm for detecting atrial fibrillation using smartphone technology. The iREAD study showed promising results, with

the AI model achieving high accuracy rates in identifying atrial fibrillation, thus offering a convenient and accessible method for early detection of heart conditions, especially in remote or underserved populations.

Li et al. (2020) [7] evaluated the use of AI to detect COVID-19 and community-acquired pneumonia using pulmonary CT scans. The study found that AI models could effectively distinguish between COVID-19 and other respiratory conditions, providing a valuable diagnostic tool during the pandemic. The authors also highlighted the importance of rapid AI adaptation to emerging diseases and updated data.

Olive-Gadea et al. (2020) [8] developed a deep learning-based software capable of identifying large vessel occlusions on noncontrast CT scans. The study demonstrated that AI could significantly enhance the speed and accuracy of stroke diagnosis, enabling faster intervention and potentially improving patient outcomes in critical care settings.

Lin et al. (2022) [9] discussed the role of AI-driven decision-making in the auxiliary diagnosis of epidemic diseases. The study focused on how AI models could analyze vast datasets in real-time to predict outbreaks and assist in the early diagnosis of diseases. By integrating machine learning techniques, healthcare systems could become more adaptive and responsive to emerging public health threats.

Iqbal et al. (2022) [10] conducted a narrative review on the future of AI in neurosurgery. The authors concluded that AI holds great promise in areas such as surgical planning, real-time decision support, and postoperative care. However, the review also pointed out the need for more extensive clinical validation of AI systems before they can be widely adopted in neurosurgical practices.

Nguyen et al. (2018) [11] explored deep learning for sudden cardiac arrest detection in automated external defibrillators (AEDs). The study showed that AI models could significantly enhance AED functionality by improving the accuracy of sudden cardiac arrest detection, potentially saving lives in critical situations where rapid response is essential.

Mostafa et al. (2022) [12] conducted a survey on AI techniques used for thoracic disease diagnosis via medical images. The study highlighted the advances in deep learning models that have improved diagnostic accuracy for conditions such as lung cancer and pneumonia. However, the survey also identified challenges related to the interpretability of AI models and the need for further research in this area.

Comito et al. (2022) [13] discussed the application of AI-driven clinical decision support systems in enhancing disease diagnosis by exploiting patient similarity. Their research demonstrated how machine learning could identify patterns in patient data to offer personalized diagnostic and treatment options, improving overall patient care and operational efficiency in hospitals.

Brinker et al. (2019) [14] conducted a study comparing the diagnostic capabilities of deep neural networks to those of dermatologists in melanoma image classification. The results showed that AI outperformed human experts, marking a significant advancement in the use of AI for dermatological diagnostics and opening up new possibilities for remote or automated skin cancer screening.

Santosh and Gaur (2021) [15] provided a comprehensive overview of AI and machine learning applications in public healthcare. They discussed how AI can address various public health issues, such as disease surveillance, outbreak prediction, and resource optimization, thereby improving health system efficiency on a large scale. The book emphasized the need for ethical AI practices and robust frameworks to ensure responsible implementation of these technologies in public health.

## **CHAPTER 3**

### **TRADITIONAL SYSTEM**

#### **3.1 Radiologist-Based Visual Inspection**

In traditional clinical practice, radiologists visually inspect X-ray images of shoulder implants to identify the manufacturer and model. This process relies heavily on the radiologist's experience, knowledge, and ability to recognize implant-specific design features. Typically, radiologists compare the X-ray images with reference images from implant catalogs or previous case studies. This approach is time-intensive, as it requires expertise in distinguishing fine details such as screw placement, stem shape, and material composition.

Despite extensive training, radiologists can make errors due to the complexity and variability of implant designs across different manufacturers. Many implants have subtle differences that are not easily distinguishable on standard X-rays, leading to misclassification. Moreover, overlapping structures in the shoulder region, such as bones and soft tissues, can obscure key implant features, further complicating the identification process. This lack of standardization in manual classification results in inconsistent accuracy rates among radiologists, ranging from 65% to 80% depending on experience levels.

Another major limitation of radiologist-based identification is the increasing workload in hospitals and imaging centers. As patient volumes grow, radiologists are required to analyze a large number of scans daily, leading to fatigue and potential diagnostic errors. The reliance on subjective interpretation also means that different radiologists may provide different identifications for the same implant, affecting clinical decisions and surgical planning. In cases requiring urgent intervention, delays in implant identification can impact patient outcomes, making this method inefficient for modern healthcare needs.

#### **3.2 Implant Manufacturer Reference Catalogues and Manual Matching**

Another manual approach involves using printed or digital implant manufacturer catalogues, which contain detailed images and specifications of various shoulder implants. In this method, radiologists, orthopaedic surgeons, or medical staff manually compare X-ray images with reference catalogue images to identify the correct implant. These catalogues are typically provided by manufacturers and include implant dimensions, material compositions, and structural designs.

While reference catalogues serve as a valuable resource, they come with significant challenges. First, the growing number of implant manufacturers and models makes it difficult to maintain an exhaustive and up-to-date catalogue. New designs are frequently introduced, while older models may be discontinued, leading to gaps in reference materials. Medical professionals may struggle to find an exact match, especially if an implant's X-ray does not clearly show all identifying features.

Additionally, this method is highly time-consuming, as it requires manual cross-referencing across multiple pages or digital databases. In emergency or revision surgeries, where rapid identification is essential, this process can introduce unnecessary delays. Furthermore, slight variations in implant positioning, rotation, or degradation over time can make it challenging to match the X-ray to the correct reference image, increasing the risk of misidentification and improper surgical planning.

### **3.3 Patient Records and Surgical Documentation Review**

A third manual system involves retrieving patient records and surgical documentation to determine the implant manufacturer. In many healthcare settings, hospitals maintain detailed surgical logs that record the type of implant used, the manufacturer, and model information. These records can help identify an implant without relying solely on visual examination.

However, this system is far from perfect. Record-keeping inconsistencies across hospitals and clinics mean that implant details may not always be documented accurately. In some cases, missing or incomplete patient records can render this method ineffective. Additionally, patients who have undergone surgery at different hospitals may not have their implant information easily accessible, especially if data-sharing protocols between institutions are not in place.

Another issue arises when dealing with older implants or international patients. If a patient received an implant years ago, the manufacturer may have discontinued the model, or the hospital may no longer have access to historical records. Similarly, if a patient was treated in another country, language barriers and differences in documentation systems may make it difficult to retrieve implant details. Lastly, relying on patient records assumes that no clerical errors occurred during data entry, yet medical record errors are a well-documented issue in healthcare systems worldwide.



### 3.4 Limitations

1. **Time-Consuming and Inefficient** – Manual identification methods, whether through radiologist inspection, catalogue comparison, or patient records, are slow and impractical, especially in urgent medical situations.
2. **Prone to Human Error** – Radiologists and medical professionals may misidentify implants due to fatigue, experience level, or inconsistencies in interpretation, leading to incorrect diagnoses.
3. **Limited Availability of Reference Materials** – Implant catalogues and databases are not always up-to-date, and older implants may lack sufficient reference data, making accurate classification difficult.
4. **Inconsistent and Non-Standardized Approach** – Different hospitals, radiologists, and medical staff may use varying methods for identification, leading to inconsistent and unreliable results.
5. **Data and Record-Keeping Issues** – Missing, incomplete, or inaccurate patient records can hinder implant identification, particularly for older implants or patients who received treatment at multiple facilities.

## **CHAPTER 4**

### **PROPOSED SYSTEM**

#### **4.1 Overview**

The project presents a machine learning framework for shoulder implant manufacturer identification, combining advanced image preprocessing, synthetic data balancing (SMOTE), and an optimized classifier selection approach—a methodology not previously explored in existing surveys. Unlike conventional SVM-based approaches, which suffer from scalability and imbalanced datasets, this method integrates Random Forest Classifier (RFC) for improved generalization and decision-making. The pipeline begins with image preprocessing techniques such as contrast enhancement, noise filtering, and morphological operations to refine implant structures. SMOTE is applied after feature extraction to address class imbalance, ensuring fair representation of all implant categories. The dataset is then split into training (80%) and testing (20%), where SVM serves as a baseline model while RFC is proposed as a superior alternative. The experimental results demonstrate that RFC outperforms SVM in accuracy, recall, and robustness against imbalanced data, making it an effective automated solution for implant classification.

#### **Step-1: Data Acquisition and Preprocessing**

The dataset consists of shoulder implant X-ray images with labels representing different implant manufacturers. To ensure consistency and improve classification performance, data preprocessing is applied. The key steps include:

- Resizing the images to a uniform dimension to maintain consistency across the dataset.
- Standardization of pixel intensity values to ensure uniformity in brightness and contrast.

This preprocessing step prepares the dataset for effective feature extraction and classification.

#### **Step-2: Handling Class Imbalance Using SMOTE**

Since some implant categories have fewer samples than others, the dataset is imbalanced. To address this, Synthetic Minority Over-sampling Technique (SMOTE) is used to generate synthetic samples for underrepresented classes. This ensures that all implant categories have sufficient representation, preventing bias during model training.

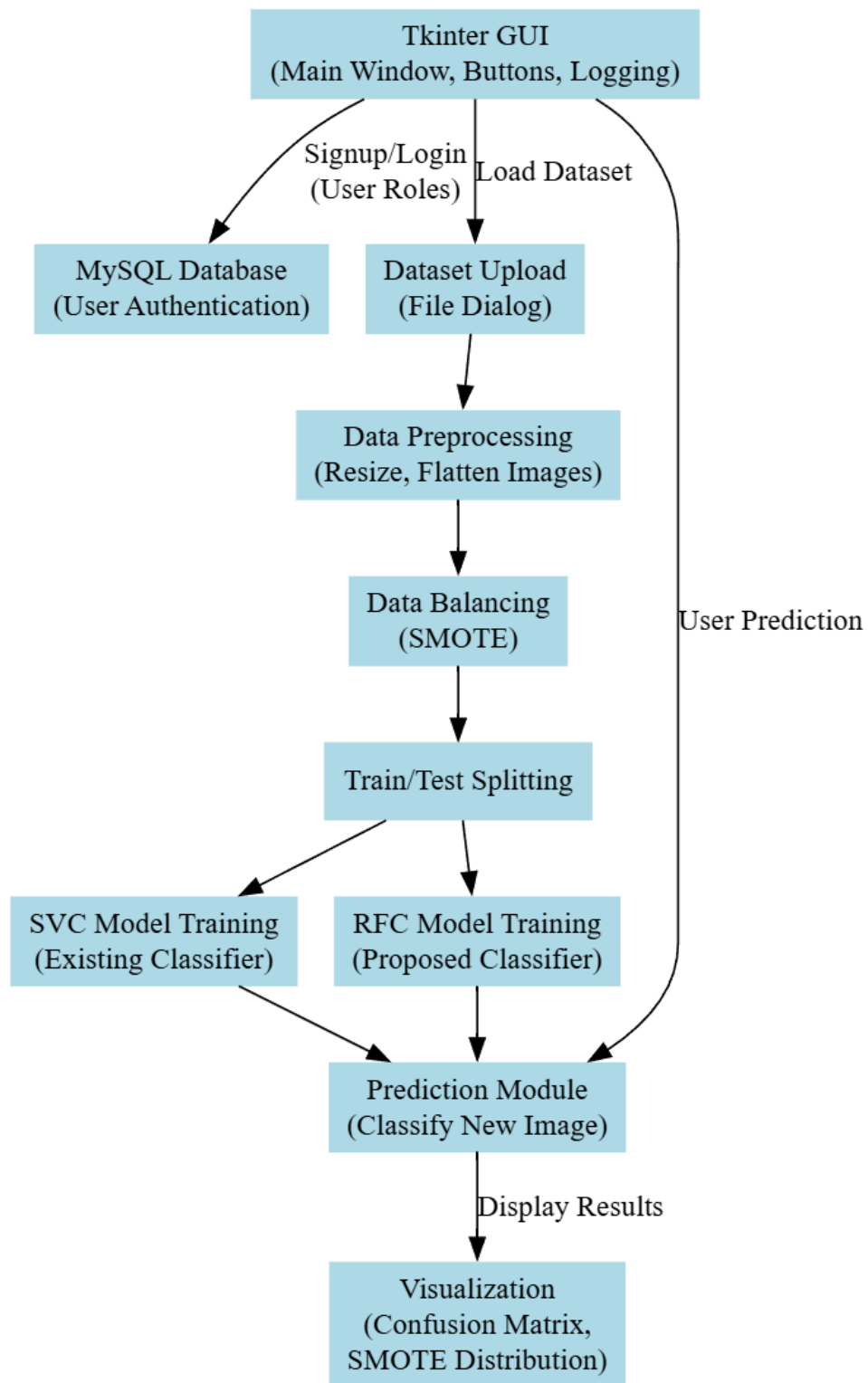


Fig. 4.1: Proposed system architecture of ML-based classification of shoulder implant X-rays for manufacturer identification.

### **Step-3: Train-Test Splitting and Baseline Model (SVM)**

The dataset is then split into 80% training and 20% testing to evaluate model performance. As a baseline, an SVM (Support Vector Machine) classifier is implemented. SVM is a widely used classification algorithm, but it has limitations in handling large datasets and imbalanced distributions, which can affect classification accuracy.

### **Step-4: Proposed Model: Random Forest Classifier (RFC)**

To improve classification performance, this study proposes the Random Forest Classifier (RFC) as an alternative to SVM. RFC is an ensemble learning algorithm that constructs multiple decision trees and aggregates their predictions. Compared to SVM, RFC provides better generalization, improved robustness to imbalanced data, and higher accuracy in identifying implant manufacturers.

### **Step-5: Model Evaluation**

The models (SVM and RFC) are evaluated using classification metrics, including:

- **Accuracy** – Measures the overall correctness of predictions.
- **Precision & Recall** – Evaluates the reliability of classification results.
- **F1-Score** – Ensures balanced evaluation for imbalanced datasets.

Experimental results demonstrate that RFC outperforms SVM, proving its effectiveness in shoulder implant manufacturer identification.

## **4.2 Dataset Preprocessing**

The dataset preprocessing function is responsible for loading, normalizing, and structuring the data to ensure it is ready for machine learning classification. This process is crucial for improving model performance and ensuring consistency in input data.

### **Step-1: Loading and Checking for Existing Processed Data**

The first step in the function is to check if preprocessed data is already available. This is done by verifying the existence of X.txt.npy and Y.txt.npy files, which store the preprocessed input features (X) and corresponding labels (Y). If these files are found, the function loads them using NumPy, avoiding redundant computation and speeding up the process. This method ensures that once preprocessing is completed, the data can be reused without repeating the entire processing pipeline.

## **Step-2: Iterating Through Dataset Folders and Extracting Images**

If the preprocessed data files do not exist, the function starts from scratch by traversing the dataset directory structure. It does this by iterating through folders that contain images, where each folder represents a different implant manufacturer category. The function extracts the folder name (which corresponds to a label category) and loads all images within it. This step is necessary to organize the dataset in a structured format where each image is mapped to its respective category label.

## **Step-3: Image Preprocessing and Feature Extraction**

Each image in the dataset undergoes preprocessing to standardize its format before being used in the machine learning model. The images are first read using OpenCV, and then resized to a fixed dimension of 64x64 pixels with 3 color channels (RGB format). Resizing ensures that all images have a uniform shape, allowing the model to process them efficiently. After resizing, the image data is flattened into a one-dimensional feature vector, which is necessary for feeding the data into machine learning algorithms that require structured input.

## **Step-4: Label Encoding for Classification**

To facilitate supervised learning, each image must be assigned a numerical label corresponding to its category. The function retrieves the name of the category from the folder structure and converts it into a numerical value using an indexing method. This step transforms textual class labels (manufacturer names) into a format suitable for machine learning models, ensuring that categorical information can be effectively processed.

## **Step-5: Converting Data to NumPy Arrays and Saving for Future Use**

After processing all images, the input feature matrix (X) and output labels (Y) are converted into NumPy arrays, which provide efficient storage and computation benefits. The processed data is then saved as NumPy binary files (.npy) for quick access in future training or testing sessions. This ensures that preprocessing is performed only once, improving efficiency and avoiding redundant computations when reloading the dataset.

## **Step-6: Final Output and Confirmation Message**

Upon completion, the function provides a confirmation message indicating that dataset preprocessing and normalization have been successfully completed. This message serves as a log for users, ensuring transparency in the data preparation workflow.

### 4.3 SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a powerful data balancing method used to address class imbalance in machine learning datasets. In this project, SMOTE is applied to ensure that all implant manufacturer categories have an equal representation, preventing the classifier from being biased toward majority classes. Instead of simply duplicating existing samples, SMOTE generates synthetic data points by interpolating between real samples in the feature space, creating more diverse and realistic training data. This enhances model generalization, improves classification performance, and ensures that minority categories are well-represented during training. By visualizing the class distribution before and after SMOTE, we can observe how this technique effectively balances the dataset, making the classification model more reliable and fair across all implant categories.

#### **Step-1: Identifying Class Imbalance**

The first step in this method is to analyze the distribution of class labels. The function extracts unique class labels (labels) from the dataset along with their corresponding sample counts (label\_count). This allows the model to determine the extent of imbalance present in the dataset. A visualization of the class distribution before SMOTE is created using a bar chart, which highlights the disproportion between different classes.

#### **Step-2: Applying SMOTE for Data Balancing**

Once the imbalance is identified, SMOTE is applied to resample the dataset. Unlike traditional oversampling methods that duplicate existing samples, SMOTE generates new synthetic samples by interpolating between existing minority class instances. It does this by selecting a random sample from the minority class, finding its nearest neighbors, and creating a new synthetic point along the line connecting them. This technique ensures that the newly generated data points are more diverse and realistic, helping the model generalize better.

#### **Step-3: Evaluating the Effect of SMOTE**

After applying SMOTE, the function retrieves the new distribution of class labels (labels\_resampled) and their respective counts (label\_count\_resampled). The results are then visualized through a side-by-side comparison of class distributions before and after SMOTE using bar charts. The first chart displays the original dataset's imbalance, while the second chart shows how SMOTE has successfully equalized the class representation.

## 4.4 Train -Test Splitting

In machine learning data pre-processing, we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model. Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So, we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

**Training Set:** A subset of dataset to train the machine learning model, and we already know the output.

**Test set:** A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

### Step-1: Declaring Global Variables

The function uses global variables (X, Y, X\_train, X\_test, y\_train, y\_test, scaler) to ensure that the dataset and its splits are accessible throughout the program. The variables X and Y contain the feature matrix and corresponding labels, respectively, which will be used to create training and testing subsets.

### Step-2: Splitting the Dataset into Training and Testing Sets

The `train_test_split` function is used to divide the dataset into two parts:

- X\_train, y\_train: Contains 70% of the data for training the model.
- X\_test, y\_test: Contains 30% of the data for testing the model.

The function specifies `test_size=0.3`, which means 30% of the dataset is allocated for testing, while the remaining 70% is used for training. This ratio helps balance the need for a sufficient amount of data in both subsets.

### Step-3: Displaying the Dataset Split Information

After the train-test split is performed, the function displays key details about the dataset:

- Total number of images in the dataset (`X.shape[0]`).
- Number of images assigned to training (`X_train.shape[0]`).
- Number of images assigned to testing (`X_test.shape[0]`).

## 4.5 ML Model Building

### 4.5.1 SVM (Support Vector Machine)

Support Vector Machine (SVM) is a supervised learning algorithm used for classification tasks. It works by finding the optimal decision boundary that maximizes the margin between different classes. The SVM classifier is implemented efficiently to classify shoulder implant manufacturers based on X-ray images. An SVM classifier is implemented to classify shoulder implant X-rays based on manufacturer labels. The process begins by checking if a pre-trained SVM model (`SVM_model.pkl`) exists; if found, it is loaded using `joblib` to avoid retraining. If not, a new Support Vector Classifier (SVC) is initialized with a polynomial kernel (`poly`), which transforms input data into a higher-dimensional space to handle complex decision boundaries. The regularization parameter (`C=1.0`) balances misclassification tolerance, while `gamma='scale'` determines the impact of each training sample on the decision boundary. The model is trained using `X_train` (input features) and `y_train` (manufacturer labels), learning to classify X-ray images accurately. Once trained, the model is saved for future use, ensuring efficiency. During testing, the trained SVM model predicts manufacturer labels for unseen `X_test` data, and the predictions (`y_pred_bnb`) are compared to actual labels (`y_test`) as shown in Fig.4.1. The function then evaluates classification performance using metrics like accuracy, precision, recall, and F1-score to measure how well the model differentiates between manufacturers. Although SVM is effective in high-dimensional spaces and for complex classification tasks, it has drawbacks such as being computationally expensive, sensitive to noise, and requiring careful kernel selection, which may limit its scalability for large datasets.



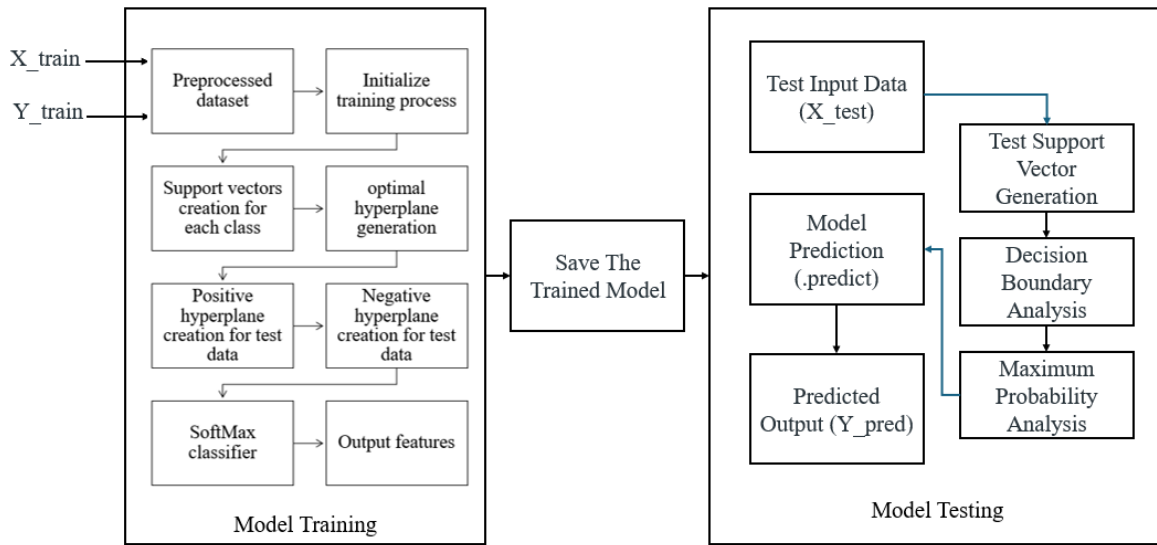


Fig. 4.1: Internal workflow of SVC model.

### Step 1: Initializing Variables for Performance Metrics

The function first initializes four global lists: accuracy, precision, recall, and fscore. These lists will store the evaluation metrics of the Support Vector Machine (SVM) classifier after testing. These metrics help assess how well the model performs in classifying the shoulder implant X-ray images based on manufacturer labels.

### Step 2: Checking for an Existing SVM Model

The function then checks if a pre-trained SVM model (SVM\_model.pkl) exists in the model directory. If the file is found, it is loaded using joblib, allowing the model to be reused without retraining. This step is efficient as it saves time by avoiding unnecessary retraining.

### Step 3: Training the SVM Classifier with Training Data

If the model is not found, the function initializes a new Support Vector Classifier (SVC) with the following parameters:

- `kernel='poly'`: A polynomial kernel is used to map input features into higher-dimensional space, allowing for better separation of complex data.
- `C=1.0`: This parameter controls the trade-off between achieving a low error and maximizing the margin.

- `gamma='scale'`: Determines how much influence a single training example has, affecting the decision boundary's flexibility.
- `random_state=42`: Ensures reproducibility by setting a fixed random seed.

The classifier is then trained using `X_train` and `y_train`, where:

- `X_train`: Represents the input features of shoulder implant X-rays.
- `y_train`: Represents the manufacturer labels corresponding to each X-ray.

The trained SVM model is then saved as a file (`SVM_model.pkl`) for future use, preventing the need for retraining unless the dataset changes.

#### **Step 4: Predicting Labels for Test Data**

Once trained, the model is used to predict the labels for test data (`X_test`). The `predict` function takes `X_test` (new shoulder implant X-ray images) as input and assigns each image to a predicted manufacturer category (`y_pred_bnb`). This output is then compared to the actual labels (`y_test`) to assess the model's performance.

#### **Step 5: Evaluating the SVM Model**

After prediction, the function calculates performance metrics such as accuracy, precision, recall, and F1-score. These metrics help measure how well the model correctly classifies the shoulder implant manufacturers. The results are stored in the previously initialized lists for analysis.

#### **4.5.2 Limitations of SVM**

- **Computationally Expensive:** SVM requires high computational power, especially with large datasets, making training slower.
- **Sensitive to Noisy Data:** The classifier is highly affected by noise and outliers, leading to incorrect decision boundaries.
- **Hard to Tune Kernel Parameters:** Choosing the right kernel (linear, poly, rbf) is challenging, and an incorrect choice reduces model performance.
- **Not Ideal for Large Datasets:** With a large number of features or samples, SVM becomes memory-intensive and inefficient.

- **Poor Performance with Overlapping Classes:** If different categories have overlapping regions, SVM may struggle to find an optimal decision boundary, reducing accuracy.

#### 4.5.3 RFC (Random Forest Classifier)

Random Forest Classifier (RFC) is an ensemble learning algorithm that builds multiple decision trees and combines their outputs to make accurate predictions. Each decision tree in the forest is trained on a different random subset of the dataset, allowing RFC to capture various patterns in the data as shown in Fig.1. During classification, each tree independently predicts the class label, and the final decision is determined by majority voting. This ensemble approach makes RFC more robust to overfitting, noise, and variations in data compared to single decision trees. It is particularly useful for handling high-dimensional datasets, complex feature interactions, and imbalanced data, making it an effective choice for classifying shoulder implant X-ray images based on manufacturer labels.

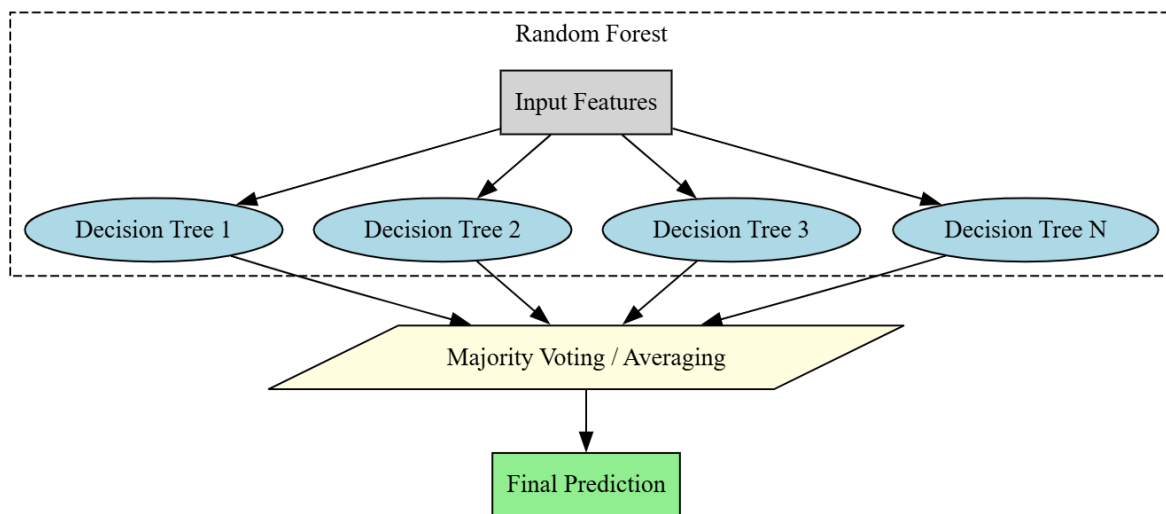


Fig. 4.3: RFC classifier workflow.

#### Step 1: Checking for an Existing Model

The function first checks if a pre-trained Random Forest Classifier (RFC) model (RFC\_Model.pkl) exists in the model directory. If the file is found, it is loaded using joblib, allowing the system to reuse the trained model without retraining. This approach saves computational time and ensures efficiency when handling large datasets.

## **Step 2: Initializing and Training the RFC Model**

If no pre-trained model is found, a new RandomForestClassifier is instantiated and trained using the dataset. The training process involves:

- `X_train` as input data → This contains the preprocessed X-ray images of shoulder implants in numerical format.
- `y_train` as output labels → These labels indicate the manufacturer category corresponding to each X-ray image.

The Random Forest algorithm works by creating multiple decision trees, each trained on random subsets of the dataset. During training, each tree learns different decision boundaries to classify the X-ray images. By aggregating predictions from multiple trees, RFC improves accuracy and reduces overfitting.

## **Step 3: Saving the Trained Model**

Once the model is trained, it is saved using `joblib.dump()`, ensuring that the classifier can be used later without needing to retrain it. This step enhances efficiency, especially when working with large datasets that require significant training time.

## **Step 4: Making Predictions on Test Data**

The trained RFC model is then tested using `X_test` (unseen X-ray images). The `predict` function is applied to classify each test sample into a manufacturer category, producing a set of predicted labels (`y_pred`). These predictions are compared with the actual manufacturer labels (`y_test`) to evaluate the model's performance.

## **Step 5: Evaluating the Model**

The predicted labels (`y_pred`) are passed to the `calculateMetrics` function, which computes performance metrics such as accuracy, precision, recall, and F1-score. These metrics help assess the classifier's effectiveness in identifying shoulder implant manufacturers based on X-ray images. By using an ensemble of decision trees, RFC ensures robust classification performance with improved generalization compared to single classifiers like SVM.

#### 4.5.4 Advantages of RFC

- **High Accuracy and Stability:** By combining multiple decision trees, RFC produces more reliable and accurate predictions than single classifiers like SVM.
- **Handles Large and High-Dimensional Data Well:** RFC can efficiently process large datasets with many features, making it suitable for medical imaging tasks.
- **Robust to Noise and Missing Data:** RFC is less sensitive to noise and missing values, making it more adaptable to real-world datasets.
- **Fast Prediction Speed:** While training may take longer, RFC provides quick predictions, making it ideal for real-time classification tasks.

## **CHAPTER 5**

### **UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

#### **5.1 Class Diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram was capable of providing certain

functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

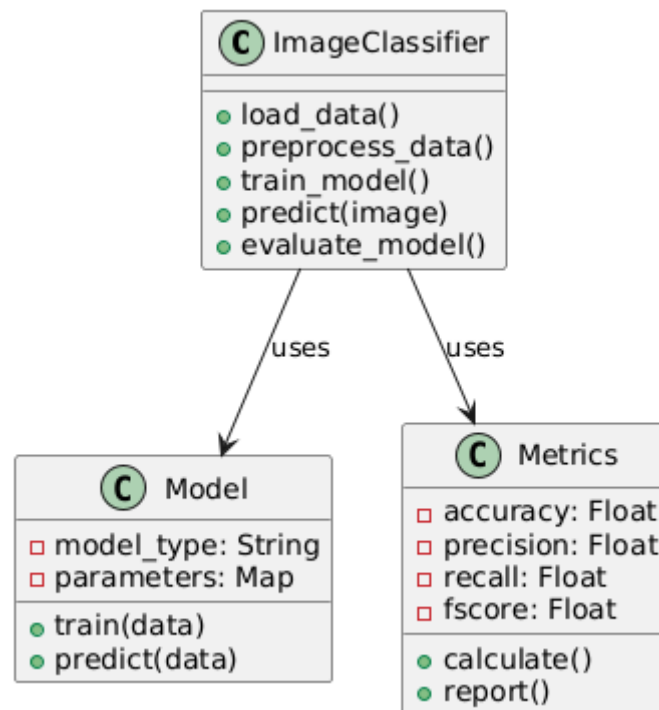


Fig. 5.1: Class diagram.

## 5.2 Sequence Diagram

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

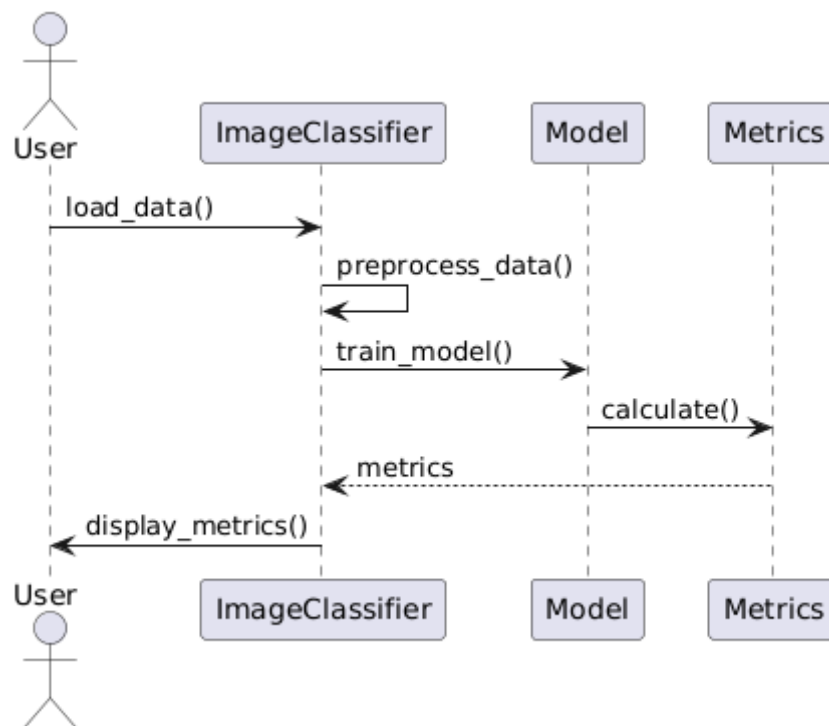


Fig. 5.2: Sequence diagram.

### 5.3 Activity Diagram

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. An activity diagram in Software Engineering represents the flow of activities in a system using UML. It includes start/end points, actions, decisions, and flow arrows to show process logic. It helps visualize workflows, use case behaviour, and method logic clearly. Supports sequential, conditional, and parallel execution paths. Useful for analysing, designing, and communicating system behaviour.



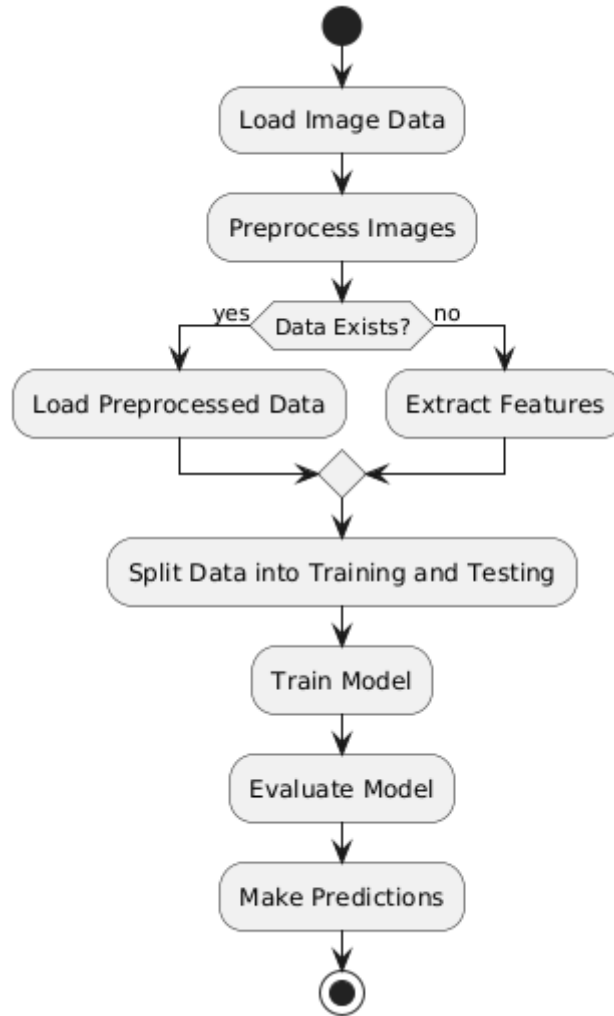


Fig. 5.3: Activity diagram.

#### 5.4 Data flow Diagram

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data between various processes, data stores, data sources, and data destinations within a system or between systems. Data flow diagrams are often used to depict the structure and behavior of a system, emphasizing the flow of data and the transformations it undergoes as it moves through the system.

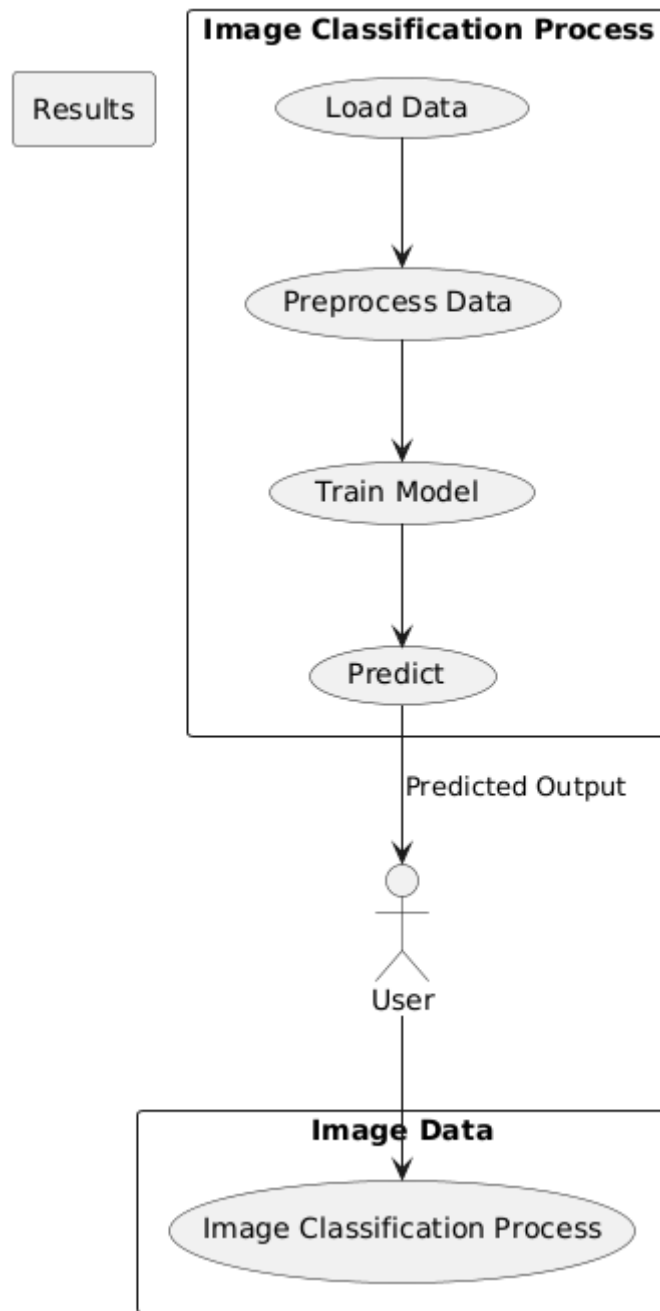


Fig. 5.4: Dataflow diagram.

### 5.5 Component Diagram

Component diagram describes the organization and wiring of the physical components in a system. A component diagram shows the organization and dependencies among software components. It models the physical aspects of a system like files, libraries, or executables. Useful for visualizing system architecture and how components interact.

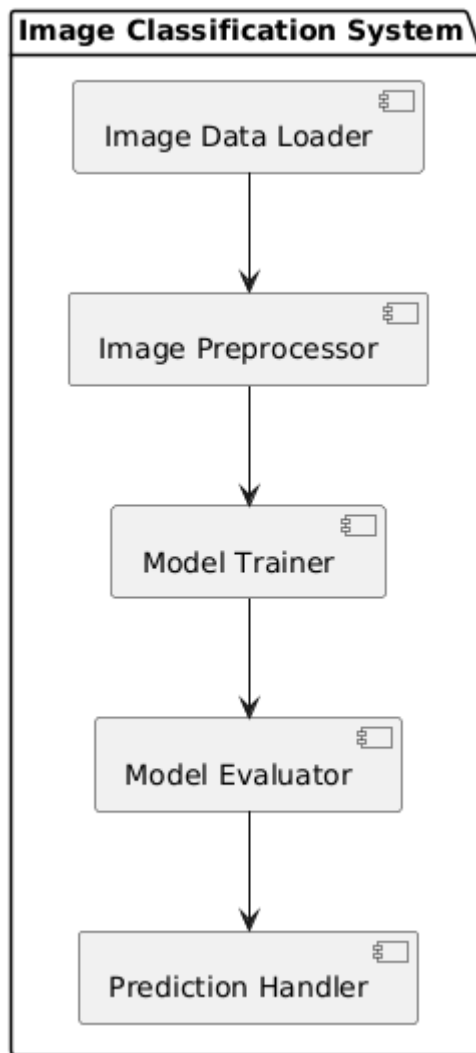


Fig. 5.5: Component diagram.

## 5.6 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

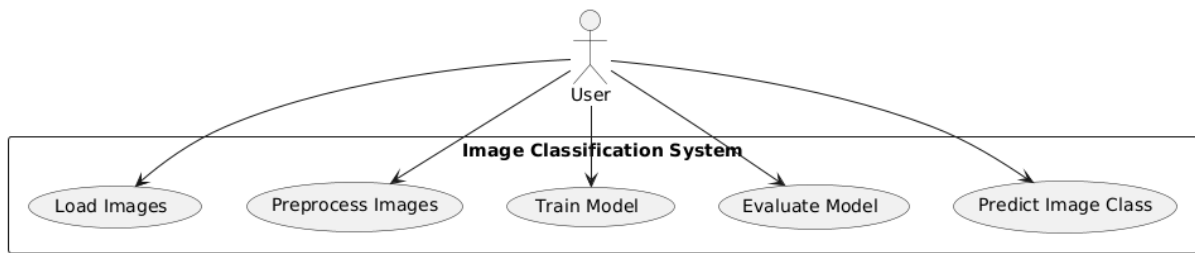


Fig. 5.6: Use case diagram.

## 5.7 Deployment Diagram

A deployment diagram in UML illustrates the physical arrangement of hardware and software components in the system. It visualizes how different software artifacts, such as data processing scripts and model training components, are deployed across hardware nodes and interact with each other, providing insight into the system's infrastructure and deployment strategy.

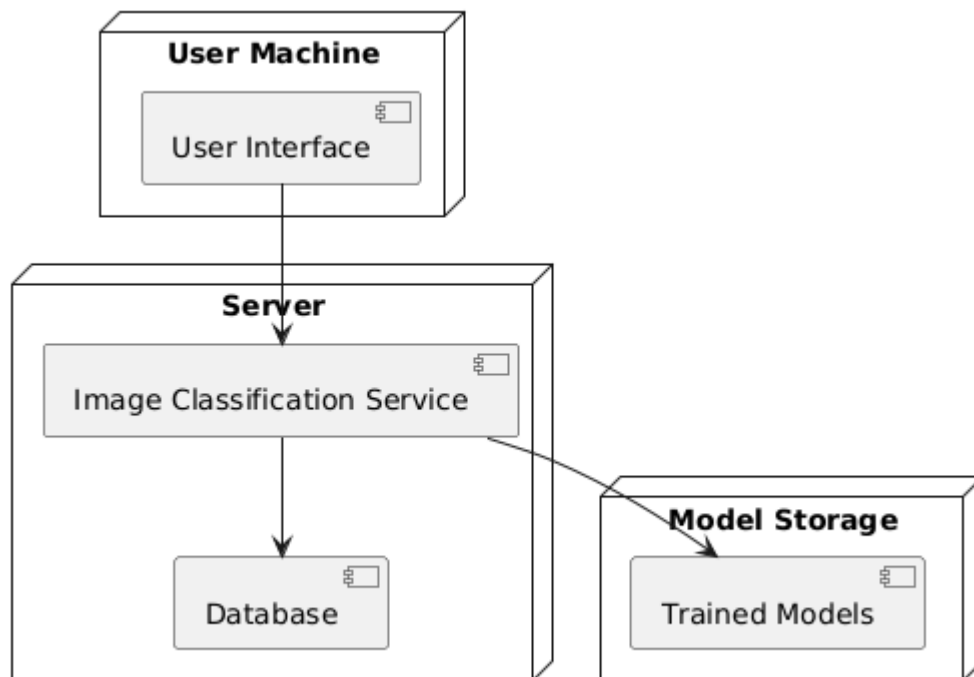


Fig. 5.7: Deployment diagram.

## CHAPTER 6

### SOFTWARE AND HARDWARE REQUIREMENTS

#### 6.1 Software Requirements

##### Python 3.7.6

Python 3.7.6 serves as a pivotal version for developers and researchers due to its robust features, backward compatibility, and widespread support across a variety of libraries and frameworks. Released during a time when machine learning and data science tools were rapidly evolving, Python 3.7.6 provided a stable and consistent platform. This version includes critical improvements like enhanced asyncio functionality for asynchronous programming, increased precision for floating-point numbers, and optimized data structures. It became the go-to version for compatibility with popular libraries like TensorFlow 2.0, PyTorch, and Pandas, ensuring seamless integration and efficient execution for both academic and industrial applications.

Compared to older Python versions, 3.7.6 introduced several features such as dataclasses, which simplified boilerplate code for object-oriented programming. The improved async and await syntax made concurrent programming more intuitive, while changes to the standard library enhanced usability and performance. Over newer versions, Python 3.7.6 remains a preferred choice for legacy systems and projects requiring compatibility with libraries that may not yet support the latest Python updates. Its combination of stability and maturity ensures that it is reliable for long-term projects, especially in environments where upgrading the Python interpreter might disrupt existing workflows.

##### Packages

```
python -m pip install --upgrade pip
```

```
pip install Cython
```

```
pip install tensorflow==1.14.0
```

```
pip install keras==2.3.1
```

```
pip install pandas==0.25.3
```

```
pip install scikit-learn==0.22.2.post1
```

```
pip install imutils
```

```
pip install matplotlib==3.1.1
```

```
pip install opencv-python==4.8.0.74
```

```
pip install seaborn==0.10.1
```

```
pip install h5py==2.10.0
```

```
pip install numpy==1.19.2
```

```
pip install jupyter
```

```
pip install protobuf==3.20.*
```

```
pip install scikit-image==0.16.0
```

## TensorFlow Environment

TensorFlow provides a comprehensive ecosystem for building, training, and deploying machine learning models. Its support for numerical computation and deep learning applications makes it a staple in AI research and development. By offering a flexible architecture, TensorFlow enables deployment across a variety of platforms, including desktops, mobile devices, and the cloud. The ability to scale across CPUs, GPUs, and TPUs ensures that TensorFlow is suitable for both small experiments and large-scale production systems.

TensorFlow's transition from older versions, like 1.x, to 2.x brought significant improvements in ease of use, including the introduction of the `tf.keras` API for building models, eager execution for dynamic computation, and enhanced debugging capabilities. Compared to newer frameworks, TensorFlow retains a strong advantage due to its mature community support, extensive documentation, and integration with TensorFlow Extended (TFX) for managing production pipelines. Its compatibility with other libraries and tools, such as Keras and TensorBoard, makes it a robust choice for end-to-end machine learning solutions.

## Packages Overview

**Keras:** Older versions of Keras required extensive configuration for custom model creation. Version 2.3.1 unified the APIs with TensorFlow integration, reducing overhead and enabling direct use of TensorFlow backends, ensuring faster execution and easier debugging. While newer versions focus on performance and distributed training, version 2.3.1 is lightweight and stable, making it ideal for smaller projects without the complexity introduced in later iterations, which are more suited for advanced workflows.

**NumPy:** Version 1.19.5 introduced critical bug fixes and performance enhancements over older versions, especially for operations involving large datasets. The improved random number generator and better handling of exceptions provide more reliable results for numerical computations. This version remains compatible with a wide range of dependent packages. While newer versions optimize speed further, 1.19.5 balances stability and compatibility, ensuring fewer compatibility issues with older software stacks.

**Pandas:** Version 0.25.3 brought significant speed improvements for large-scale data processing, particularly in operations like groupby. Enhancements in handling missing data and improved compatibility with external libraries made this version more robust for data analysis tasks. While newer versions does not add features like enhanced type checking, 0.25.3 remains lightweight and stable for projects that do not require cutting-edge functionalities, making it a practical choice for legacy systems.

**Imbalanced-learn:** Version 0.7.0 introduced optimized algorithms for handling class imbalances, such as improved SMOTE implementations. This update also enhanced the ease of integrating with scikit-learn pipelines. While newer versions do not contain experimental features, 0.7.0 is reliable and well-documented, ensuring robust performance in addressing data imbalance issues without unnecessary complexity.

**Scikit-learn:** Version 0.23.1 included improved support for cross-validation and hyperparameter optimization. Updates to RandomForestClassifier and GradientBoostingClassifier increased model accuracy and efficiency. 0.23.1 is widely tested and compatible with older hardware, making it a dependable choice for environments where the latest versions may introduce compatibility issues.

**Imutils:** This package provides an easy-to-use interface for image processing tasks. Its functions for resizing, rotating, and translating images simplified workflows compared to writing custom code. Its lightweight nature and stable functionality make it suitable for projects not requiring cutting-edge image manipulation techniques, balancing simplicity and capability.

**Matplotlib:** Version 3.x improved plot interactivity and introduced better 3D plotting capabilities. The `tight_layout` function and compatibility with modern libraries streamlined visualization workflows. The earlier versions maintain stability and compatibility with older datasets and software, avoiding potential issues from newer, untested updates.

**Seaborn:** Improved APIs in newer versions simplified aesthetic customization of plots. The addition of new themes and color palettes in 0.11.x enhanced visual appeal for exploratory data analysis. Older versions remain computationally less demanding, suitable for lightweight applications without requiring extensive customizations.

**OpenCV-Python:** Recent updates enhanced compatibility with deep learning frameworks and accelerated image processing pipelines, especially for real-time applications. Older versions are stable and resource-efficient, making them ideal for systems with limited computational capacity or for legacy applications.

**H5Py:** Version 2.10.0 improved file handling efficiency for large datasets. It introduced better support for advanced indexing, which is crucial for working with high-dimensional data. While newer versions support more advanced features, 2.10.0 ensures compatibility with older machine learning frameworks and models.

**Jupyter:** Jupyter improved the interactivity and scalability of notebooks for collaborative coding and visualization tasks. Integration with tools like Matplotlib made it a preferred environment for data exploration. Earlier versions are stable and lightweight, avoiding potential issues with dependencies introduced in newer releases.

## 6.2 Hardware Requirements

Python 3.7.6 can run efficiently on most modern systems with minimal hardware requirements. However, meeting the recommended specifications ensures better performance, especially for developers handling large-scale applications or computationally intensive tasks. By ensuring compatibility with hardware and operating system, can leverage the full potential of Python 3.7.6.

**Processor (CPU) Requirements:** Python 3.7.6 is a lightweight programming language that can run on various processors, making it highly versatile. However, for optimal performance, the following processor specifications are recommended:

- **Minimum Requirement:** 1 GHz single-core processor.
- **Recommended:** Dual-core or quad-core processors with a clock speed of 2 GHz or higher. Using a multi-core processor allows Python applications, particularly those involving multithreading or multiprocessing, to execute more efficiently.



**Memory (RAM) Requirements:** Python 3.7.6 does not demand excessive memory but requires adequate RAM for smooth performance, particularly for running resource-intensive applications such as data processing, machine learning, or web development.

- **Minimum Requirement:** 512 MB of RAM.
- **Recommended:** 4 GB or higher for general usage. For data-intensive operations, 8 GB or more is advisable.

Insufficient RAM can cause delays or crashes when handling large datasets or executing computationally heavy programs.

**Storage Requirements:** Python 3.7.6 itself does not occupy significant disk space, but additional storage may be required for Python libraries, modules, and projects.

- **Minimum Requirement:** 200 MB of free disk space for installation.
- **Recommended:** At least 1 GB of free disk space to accommodate libraries and dependencies.

Developers using Python for large-scale projects or data science should allocate more storage to manage virtual environments, datasets, and frameworks like TensorFlow or PyTorch.

**Compatibility with Operating Systems:** Python 3.7.6 is compatible with most operating systems but requires hardware that supports the respective OS. Below are general requirements for supported operating systems:

- **Windows:** 32-bit and 64-bit systems, Windows 7 or later.
- **macOS:** macOS 10.9 or later.
- **Linux:** Supports a wide range of distributions, including Ubuntu, CentOS, and Fedora.

The hardware specifications for the OS directly impact Python's performance, particularly for modern software development.

## **CHAPTER 7**

### **FUNCTIONAL REQUIREMENTS**

Below are the detailed functional requirements for the project:

#### **1. User Authentication and Role Management:**

- The system must provide distinct signup and login interfaces for two user roles: Hospital Administrators (Admin) and Patients (User).
- It should securely store and retrieve user credentials from a MySQL database, ensuring that only authenticated users can access their respective functionalities.
- The system must display role-specific interfaces once a user logs in, with admins having access to data management and model training features, and patients having access to the prediction functionalities.

#### **2. Dataset Management and Preprocessing:**

- The system must allow users (primarily admins) to upload a dataset through a file dialog interface. The dataset should be organized in directories representing different manufacturers.
- Once the dataset is uploaded, the system must automatically identify the classes (manufacturers) based on the directory names.
- The application should preprocess the images by performing operations such as resizing each image to a fixed dimension (e.g., 64×64 pixels), flattening the image into a one-dimensional array, and excluding any non-image files (e.g., Thumbs.db).

#### **3. Data Balancing and Splitting:**

- The system must detect and handle class imbalances using techniques like SMOTE, ensuring that the training data is balanced across different manufacturers.
- It must provide functionality to split the preprocessed dataset into training and testing sets, with configurable ratios (e.g., 70% training, 30% testing).

#### **4. Machine Learning Model Training and Evaluation:**

- The system should implement two separate machine learning models: a Support Vector Classifier (SVC) and a Random Forest Classifier (RFC).
- It must check if pre-trained models exist (stored in files such as SVM\_model.pkl and RFC\_Model.pkl) and load them; if not, it should train new models using the training data and save the trained models for future use.
- After model training, the system must evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1-score.
- The evaluation results, including a confusion matrix visualization, must be displayed in the application's output/logging area.

#### **5. Prediction Functionality:**

- The system must provide a functionality where users can upload a new shoulder implant X-ray image for prediction.
- Upon selecting an image, the system should preprocess it in the same manner as the training data and use the trained classifier to predict the manufacturer.
- The predicted output must be overlaid on the image and displayed to the user.

#### **6. Graphical User Interface (GUI):**

- The entire application should have an intuitive and visually appealing GUI built using Tkinter.
- The interface should include clear navigation elements, a background image, a text area for logging system messages, and buttons that trigger various functionalities.
- The GUI must dynamically update based on the user role, ensuring that both admin-specific and user-specific options are available post-login.

These functional requirements ensure that the system effectively bridges machine learning with medical imaging, providing a robust, user-friendly application for classifying shoulder implant X-rays by manufacturer.

## CHAPTER 8

### SOURCE CODE

```
from tkinter import messagebox

from tkinter import *

from tkinter import simpledialog

import tkinter

import warnings

warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from tkinter import ttk

from tkinter import filedialog

import pymysql

import tkinter as tk

import os

from skimage.transform import resize

from skimage.io import imread

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.feature_selection import SelectKBest
```

```
from sklearn.feature_selection import f_classif

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import fl_score

from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

import seaborn as sns

from skimage import io, transform

from sklearn import preprocessing

import numpy as np

import joblib

import cv2

from imblearn.over_sampling import SMOTE

from PIL import Image, ImageTk
```

```
model_folder = "model"
```

```
def uploadDataset():
```

```
    global filename, categories
```

```
    text.delete('1.0', END)
```

```

filename = filedialog.askdirectory(initialdir=".")

categories = [d for d in os.listdir(filename) if os.path.isdir(os.path.join(filename, d))]

text.insert(END,'Dataset loaded\n')

text.insert(END,"Classes found in dataset: "+str(categories)+"\n")

```

```

def DatasetPreprocessing():

```

```

    text.delete('1.0', END)

```

```

    global X, Y, dataset, label_encoder

```

```

    X_file = os.path.join(model_folder, "X.txt.npy")

```

```

    Y_file = os.path.join(model_folder, "Y.txt.npy")

```

```

    if os.path.exists(X_file) and os.path.exists(Y_file):

```

```

        X = np.load(X_file)

```

```

        Y = np.load(Y_file)

```

```

        print("X and Y arrays loaded successfully.")

```

```

    else:

```

```

        X = [] # input array

```

```

        Y = [] # output array

```

```

        for root, dirs, directory in os.walk(path):

```

```

            for j in range(len(directory)):

```

```

                name = os.path.basename(root)

```

```

                print(f>Loading category: {dirs}')

```

```

                print(name+" "+root+"/"+directory[j])

```

```

if 'Thumbs.db' not in directory[j]:

    img_array = cv2.imread(root+"/"+directory[j])

    img_resized = resize(img_array, (64, 64, 3))

    # Append the input image array to X

    X.append(img_resized.flatten())

    # Append the index of the category in categories list to Y

    Y.append(categories.index(name))

X = np.array(X)

Y = np.array(Y)

np.save(X_file, X)

np.save(Y_file, Y)

text.insert(END, "Dataset Normalization & Preprocessing Task Completed\n\n")

def Dataset_SMOTE():

    text.delete('1.0', END)

    global X, Y, dataset, label_encoder

    labels, label_count = np.unique(Y, return_counts=True)

    smote = SMOTE(random_state=42)

    X, Y = smote.fit_resample(X, Y)

    labels_resampled, label_count_resampled = np.unique(Y, return_counts=True)

    plt.figure(figsize=(10, 5))

# Before SMOTE

```

```
plt.subplot(1, 2, 1)

plt.bar(labels, label_count, color='skyblue', alpha=0.8)

plt.xlabel("Output Type")

plt.ylabel("Count")

plt.title("Before SMOTE")
```

```
# After SMOTE
```

```
plt.subplot(1, 2, 2)

plt.bar(labels_resampled, label_count_resampled, color='lightgreen', alpha=0.8)

plt.xlabel("Output Type")

plt.ylabel("Count")

plt.title("After SMOTE")

plt.tight_layout()

plt.show()
```

```
def Train_test_splitting():
```

```
    text.delete('1.0', END)
```

```
    global X, Y, dataset, label_encoder
```

```
    global X_train, X_test, y_train, y_test, scaler
```

```
    #splitting dataset into train and test where application using 80% dataset for training and
    20% for testing
```



```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3) #split dataset into train
and test
```

```
text.insert(END,"Dataset Train & Test Splits\n")
```

```
text.insert(END,"Total images found in dataset : "+str(X.shape[0])+"\n")
```

```
text.insert(END,"70% dataset used for training : "+str(X_train.shape[0])+"\n")
```

```
text.insert(END,"30% dataset user for testing  : "+str(X_test.shape[0])+"\n")
```

```
def calculateMetrics(algorithm, testY, predict):
```

```
    global categories
```

```
    labels = categories
```

```
    p = precision_score(testY, predict,average='macro') * 100
```

```
    r = recall_score(testY, predict,average='macro') * 100
```

```
    f = fl_score(testY, predict,average='macro') * 100
```

```
    a = accuracy_score(testY,predict)*100
```

```
    accuracy.append(a)
```

```
    precision.append(p)
```

```
    recall.append(r)
```

```
    fscore.append(f)
```

```
    text.insert(END,algorithm+" Accuracy : "+str(a)+"\n")
```

```
    text.insert(END,algorithm+" Precision : "+str(p)+"\n")
```

```
    text.insert(END,algorithm+" Recall   : "+str(r)+"\n")
```

```
    text.insert(END,algorithm+" FSCORE    : "+str(f)+"\n\n")
```

```
    conf_matrix = confusion_matrix(testY, predict)
```

```

ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True,
cmap="viridis" ,fmt ="g");

ax.set_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()


#now train existing algorithm

def Existing_Classifier():

    text.delete('1.0', END)

    global accuracy, precision, recall, fscore

    global X_train, y_train, X_test, y_test

    accuracy = []

    precision = []

    recall = []

    fscore = []


    if os.path.exists('model/SVM_model.pkl'):

        classifier = joblib.load('model/SVM_model.pkl')

    else:

        classifier = SVC(kernel='poly', C=1.0, gamma='scale', random_state=42,)

        classifier.fit(X_train, y_train)

        joblib.dump(classifier, 'model/SVM_model.pkl')

```

```

y_pred_bnb = classifier.predict(X_test)

calculateMetrics("SVC Model", y_test, y_pred_bnb)


def Proposed_Classifier():

    global classifier

    text.delete('1.0', END)

    global X_train, y_train, X_test, y_test

    if os.path.exists('model/RFC_Model.pkl'):

        # Load the model from the pkl file

        classifier = joblib.load('model/RFC_Model.pkl')

    else:

        # Train the classifier on the training data

        classifier = RandomForestClassifier()

        classifier.fit(X_train, y_train)

        # Save the model weights to a pkl file

        joblib.dump(classifier, 'model/RFC_Model.pkl')


y_pred = classifier.predict(X_test)

calculateMetrics("RFC Model", y_test, y_pred)


def predict():

    global classifier, categories

    filename = filedialog.askopenfilename(initialdir="testImages")

```

```

img = cv2.imread(filename)

img_resize=resize(img,(64,64,3))

img_preprocessed=[img_resize.flatten()]

output_number=classifier.predict(img_preprocessed)[0]

output_name=categories[output_number]


plt.imshow(img)

plt.text(10, 10, f'Predicted Output: {output_name}', color='white', fontsize=12,
weight='bold', backgroundcolor='black')

plt.axis('off')

plt.show()


def connect_db():

    return pymysql.connect(host='localhost', user='root', password='root', database='sparse_db')


# Signup Functionality

def signup(role):

    def register_user():

        username = username_entry.get()

        password = password_entry.get()


    if username and password:

        try:

            conn = connect_db()

            cursor = conn.cursor()

```

```

        query = "INSERT INTO users (username, password, role) VALUES (%s, %s, %s)"

        cursor.execute(query, (username, password, role))

        conn.commit()

        conn.close()

        messagebox.showinfo("Success", f"{role} Signup Successful!")

        signup_window.destroy()

    except Exception as e:

        messagebox.showerror("Error", f"Database Error: {e}")

    else:

        messagebox.showerror("Error", "Please enter all fields!")


signup_window = tk.Toplevel(main)

signup_window.geometry("400x300")

signup_window.title(f"{role} Signup")


tk.Label(signup_window, text="Username").pack(pady=5)

username_entry = tk.Entry(signup_window)

username_entry.pack(pady=5)


tk.Label(signup_window, text="Password").pack(pady=5)

password_entry = tk.Entry(signup_window, show="*")

password_entry.pack(pady=5)


tk.Button(signup_window, text="Signup", command=register_user).pack(pady=10)

```

```
# Login Functionality
```

```
def login(role):
```

```
    def verify_user():
```

```
        username = username_entry.get()
```

```
        password = password_entry.get()
```

```
    if username and password:
```

```
        try:
```

```
            conn = connect_db()
```

```
            cursor = conn.cursor()
```

```
            query = "SELECT * FROM users WHERE username=%s AND password=%s AND  
role=%s"
```

```
            cursor.execute(query, (username, password, role))
```

```
            result = cursor.fetchone()
```

```
            conn.close()
```

```
            if result:
```

```
                messagebox.showinfo("Success", f"{role} Login Successful!")
```

```
                login_window.destroy()
```

```
                if role == "Admin":
```

```
                    show_admin_buttons()
```

```
                elif role == "User":
```

```
                    show_user_buttons()
```

```
            else:
```

```
                messagebox.showerror("Error", "Invalid Credentials!")
```

```
        except Exception as e:
```

```

        messagebox.showerror("Error", f"Database Error: {e}")

    else:

        messagebox.showerror("Error", "Please enter all fields!")


login_window = tk.Toplevel(main)

login_window.geometry("400x300")

login_window.title(f'{role} Login')


tk.Label(login_window, text="Username").pack(pady=5)

username_entry = tk.Entry(login_window)

username_entry.pack(pady=5)


tk.Label(login_window, text="Password").pack(pady=5)

password_entry = tk.Entry(login_window, show="*")

password_entry.pack(pady=5)


tk.Button(login_window, text="Login", command=verify_user).pack(pady=10)


# Admin Button Functions

def show_admin_buttons():

    clear_buttons()

    tk.Button(main, text="Upload UW Shoulder Dataset", command=uploadDataset,
font=font1).place(x=100, y=150)

    tk.Button(main, text="Preprocessing", command=DatasetPreprocessing,
font=font1).place(x=390, y=150)

```

```
tk.Button(main, text="Data Balancing using SMOTE", command=Dataset_SMOTE,
font=font1).place(x=560, y=150)
```

```
tk.Button(main, text="Data Splitting", command=Train_test_splitting,
font=font1).place(x=870, y=150)
```

```
tk.Button(main, text="Build & Train SVC Model", command=Existing_Classifier,
font=font1).place(x=100, y=200)
```

```
tk.Button(main, text="Build & Train Proposed RFC model",
command=Proposed_Classifier, font=font1).place(x=390, y=200)
```

```
# User Button Functions
```

```
def show_user_buttons():
```

```
    clear_buttons()
```

```
    tk.Button(main, text="Prediction", command=predict, font=font1).place(x=550, y=200)
```

```
    #tk.Button(main, text="Comparison Graph", command=comparison_graph,
font=font1).place(x=400, y=400)
```

```
# Clear buttons before adding new ones
```

```
def clear_buttons():
```

```
    for widget in main.winfo_children():
```

```
        if isinstance(widget, tk.Button) and widget not in [admin_button, user_button]:
```

```
            widget.destroy()
```

```
# Main tkinter window
```

```
main = tk.Tk()
```



```

main.title("Machine Learning-Based Classification of Shoulder Implant X-Rays for
Manufacturer Identification") #designing main screen

screen_width = main.winfo_screenwidth()

screen_height = main.winfo_screenheight()

main.geometry(f'{screen_width}x{screen_height}')

# Load and set background image

bg_image_path = "background.jpg" # Replace with your image file path

bg_image = Image.open(bg_image_path)

bg_image = bg_image.resize((screen_width, screen_height), Image.LANCZOS)

bg_photo = ImageTk.PhotoImage(bg_image)

# Create a background label

bg_label = tk.Label(main, image=bg_photo)

bg_label.place(x=0, y=0, relwidth=1, relheight=1)


# Title

font = ('times', 18, 'bold')

title = tk.Label(main, text="Machine Learning-Based Classification of Shoulder Implant X-
Rays for Manufacturer Identification", bg='white', fg='black', font=font, height=2, width=100)

title.pack()


font1 = ('times', 12, 'bold')

text=Text(main,height=22,width=170)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=100,y=250)

```

```
text.config(font=font1)
```

```
# Admin and User Buttons
```

```
font1 = ('times', 14, 'bold')
```

```
tk.Button(main, text="Hospital Signup", command=lambda: signup("Admin"), font=font1,  
width=20, height=1, bg='Lightpink').place(x=100, y=100)
```

```
tk.Button(main, text="Patient Signup", command=lambda: signup("User"), font=font1,  
width=20, height=1, bg='Lightpink').place(x=400, y=100)
```

```
admin_button = tk.Button(main, text="Hospital Login", command=lambda: login("Admin"),  
font=font1, width=20, height=1, bg='Lightpink')
```

```
admin_button.place(x=700, y=100)
```

```
user_button = tk.Button(main, text="Patient Login", command=lambda: login("User"),  
font=font1, width=20, height=1, bg='Lightpink')
```

```
user_button.place(x=1000, y=100)
```

```
main.config(bg='Lightblue')
```

```
main.mainloop()
```

## CHAPTER 9

### TESTING

#### Unit Testing

- Unit testing focuses on verifying individual functions or components in isolation. In your project, this could include testing functions for image preprocessing (like resizing, normalization), feature extraction, model loading, and individual layers in your model architecture.
- Each function should be tested with both expected and edge-case inputs to ensure they return correct outputs. Unit tests are typically automated and written using testing frameworks like `pytest` or `unittest` in Python. These tests help catch bugs early in development, making debugging easier.
- Unit testing ensures the correctness of the smallest code units and prevents future code changes from breaking existing functionality. It promotes modular and reliable code development. Writing thorough unit tests also makes refactoring code safer.

#### Integration Testing

- Integration testing checks how different parts of the system work together. For your ML pipeline, this includes integration between image input, preprocessing pipeline, model inference, and result output.
- It ensures that the data flows correctly from one module to another without breaking. For instance, an image uploaded by the user must be properly preprocessed and passed to the model, and the prediction must be displayed without errors.
- Integration tests also detect issues like incompatible data formats or failed API calls. Tools like `pytest` can also support integration tests with the help of fixtures. This testing confirms that the modules are not only correct individually but also collaborate effectively as a system.

#### System Testing

- System testing involves testing the entire application as a complete system to verify that it meets its specified requirements. For your project, this means uploading an X-ray image and receiving the correct implant manufacturer label as output.

- It tests the system end-to-end, including front-end (if any), backend, database (if used), model inference, and display of results. The goal is to simulate a real user's interaction with the application and check if all functionalities perform well together.
- System testing often includes both functional and non-functional tests like performance and usability. It's the final step before acceptance testing and ensures the full system functions cohesively.

### **Validation Testing**

- Validation testing in machine learning is used to evaluate the model on a validation dataset (unseen during training) to tune hyperparameters and make decisions like early stopping. It's a crucial step to avoid overfitting, ensuring the model generalizes well to new data.
- For example, if your model performs very well on training data but poorly on validation data, it may be memorizing rather than learning. Using techniques like stratified sampling ensures that class distributions are maintained across folds.
- Validation testing also provides insight into model consistency and reliability. This phase helps in selecting the best model version before testing on the final test set.

### **Performance Testing**

- Performance testing in ML focuses on evaluating how well the model performs in terms of metrics like accuracy, precision, recall, F1-score, confusion matrix, and AUC-ROC. It also includes measuring inference time and memory usage.
- For medical applications like yours, precision and recall are crucial to ensure minimal false positives/negatives. Performance testing ensures that the model not only gives correct outputs but does so efficiently under expected workloads.
- These metrics help compare different models and select the best-performing one. Additionally, it can help identify bias in class predictions and guide model improvements. Visualizations like ROC curves or confusion matrices are commonly used.

### **Cross-validation**

- Cross-validation is a robust method to ensure that the ML model performs consistently across different subsets of data. It involves dividing the dataset into  $k$  parts, training on  $k-1$  parts, and testing on the remaining one.

- This is repeated  $k$  times ( $k$ -fold cross-validation), and the average score is used for evaluation. It's especially useful in small or imbalanced datasets where a simple train-test split may not give a reliable performance estimate.
- Cross-validation helps reduce variance and bias in model evaluation. Stratified  $k$ -fold is often used when class distribution is imbalanced, as it ensures each fold has a similar distribution of classes.

### **User Acceptance Testing (UAT)**

- UAT is the final phase of testing where actual users (e.g., radiologists or healthcare professionals) interact with the system to verify its usefulness and correctness in a real-world scenario.
- The goal is to confirm that the system meets their needs and performs its intended function. Users might test the system by uploading real X-ray images and verifying if the predicted implant manufacturer is correct. They may also give feedback on the interface, prediction clarity, and system response.
- UAT helps catch usability or accuracy issues missed in earlier stages. It also boosts confidence before deployment in a clinical setting.

### **Stress Testing**

- Stress testing involves evaluating how the system behaves under extreme conditions, such as large batches of high-resolution X-ray images or continuous usage for a long time.
- It helps uncover performance bottlenecks, memory leaks, or crashes that might occur in real-world usage. For example, if the system is deployed in a hospital setting and many users upload images at the same time, the model should still process them efficiently.
- Stress testing can include increasing data input volume, reducing system resources, or simulating multiple user access. This ensures the application is reliable and scalable under pressure.

### **Regression Testing**

- Regression testing ensures that newly added features or changes in the codebase do not break the existing functionalities. For instance, if you upgrade your model or change the preprocessing logic, regression testing will re-run all old test cases to verify consistent output.

- It's important in projects where multiple iterations and model improvements are made. Automated regression tests can help quickly identify if a change introduces bugs or inconsistencies.
- It maintains stability and quality over time. This type of testing becomes more crucial as the codebase grows and multiple components are integrated.

### **Security Testing**

- Security testing ensures that the system handles user data securely, especially important in medical applications where patient privacy is critical. This involves testing for vulnerabilities such as unauthorized access, data leaks, or improper input handling (e.g., uploading non-image files).
- Ensuring encrypted transmission, secure storage of images, and access control mechanisms are key parts of this testing. The system should prevent injection attacks, brute-force access attempts, or file path manipulations. Compliance with standards like HIPAA (if applicable) may be necessary. Security testing builds user trust and protects sensitive healthcare information.

### **Exploratory Testing**

- Exploratory testing is a manual testing approach where testers explore the system without predefined test cases. The goal is to discover unexpected issues through creative usage patterns. For your project, this might involve uploading X-rays with unusual resolutions, noise, or poor lighting to see how the system reacts.
- Testers might also attempt mixed-class images or modified implants. This helps identify weaknesses not covered by automated tests. It combines learning, test design, and execution simultaneously, and is especially useful in early prototypes or unfamiliar systems. It encourages thinking beyond typical use cases.

### **Usability Testing**

- Usability testing evaluates how user-friendly and intuitive the system is. In your project, this includes checking how easily a doctor or technician can upload an X-ray, view the results, and interpret the manufacturer label.
- Enhancing usability improves system adoption and reliability in clinical workflows.

## CHAPTER 10

### RESULTS AND DISCUSSION

#### 10.1 Implementation Description

This research is a complete Python application that combines machine learning, image processing, and a graphical user interface (GUI) to classify shoulder implant X-rays and identify their manufacturers. It integrates several components:

##### 1. Imports and Dependencies

- **GUI Libraries:**
  - Uses various modules from tkinter (including messagebox, simpledialog, ttk, filedialog) for building the GUI.
  - Uses PIL (Python Imaging Library) via Image and ImageTk to handle image loading and display.
- **Data Processing and Visualization:**
  - Uses numpy and pandas for data handling.
  - Uses matplotlib.pyplot for plotting graphs.
  - Uses seaborn for advanced visualization, particularly for plotting confusion matrices.
- **Machine Learning:**
  - Imports models and utilities from sklearn, including RandomForestClassifier, SVC for classification, and tools for feature selection and evaluation metrics.
  - Uses joblib for saving and loading machine learning models.
- **Image Processing:**
  - Uses cv2 (OpenCV) for reading and processing images.
  - Uses skimage functions (like resize and imread) to manipulate and standardize image dimensions.

- **Database Interaction:** Uses pymysql to connect to a MySQL database, which supports user authentication and role management.
- **Data Imbalance Handling:** Uses SMOTE from imblearn.over\_sampling to address class imbalance in the dataset.
- **Warnings Management:** Suppresses warnings with the warnings module.

## 2. Core Functionalities and Functions

### Data Loading and Preprocessing:

- **uploadDataset():** Opens a directory dialog to load a dataset and lists all the classes (subdirectories) found. It then logs the classes in a text widget.
- **DatasetPreprocessing():** Checks if preprocessed dataset arrays (X and Y) exist in a specified model folder. If not, it walks through the dataset directory, reads and resizes each image to a standardized shape (64x64 pixels with 3 channels), flattens the image data, and creates corresponding labels based on the directory name. The processed arrays are saved for future runs.
- **Dataset\_SMOTE():** Applies SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset. It plots bar charts showing the distribution of classes before and after applying SMOTE.

### Data Splitting and Evaluation:

- **Train\_test\_splitting():** Splits the preprocessed data into training and testing sets using a 70-30 split (though the comment states 80-20, the actual code uses 30% for testing). It then logs the sizes of the training and testing sets.
- **calculateMetrics():** Takes in a classifier's name, actual test labels, and predicted labels, then computes key evaluation metrics: accuracy, precision, recall, and F1-score. It appends these metrics to global lists and displays them in the text widget. A confusion matrix is also generated and visualized using Seaborn.

### Model Training and Prediction:

- **Existing\_Classifier():** Loads an SVC (Support Vector Classifier) model if it exists; otherwise, it trains a new SVC model using the training data, saves it, and evaluates its performance on the test set.



- **Proposed\_Classifier():** Similarly, it loads or trains a Random Forest Classifier (RFC) model. The model is evaluated using the same metric calculation function.
- **predict():** Provides functionality to select an image file via a file dialog, processes the image (resizes and flattens it), uses the trained classifier to predict the class (manufacturer), and displays the image with an overlay showing the predicted manufacturer.

#### **Database and User Authentication:**

- **connect\_db():** Establishes a connection to a MySQL database, which is used for user management.
- **signup(role):** Implements user signup functionality. It opens a new window where a user can register with a username and password, and the provided role (Admin or User) is stored in the database.
- **login(role):** Implements login functionality. It verifies credentials against the database. Upon successful login, it displays different sets of buttons (for Admins or Users) using `show_admin_buttons()` or `show_user_buttons()`.

#### **GUI Button Management:**

- **show\_admin\_buttons() and show\_user\_buttons():** These functions dynamically update the main GUI window with buttons relevant to the logged-in user's role. Admins get options to load data, preprocess, balance, split, and train models, whereas users (patients) get the option for prediction.
- **clear\_buttons():** Clears current buttons (except the main role buttons) from the GUI before displaying a new set.

### **3. GUI Setup and Main Loop**

- **Main Window Initialization:** A main Tkinter window is created with a title and dimensions matching the screen size.
- **Background Setup:** A background image is loaded, resized to the screen dimensions, and set as the background using a label.

- **Layout Elements:** The window contains a title label, a text widget (with scrollbar) for logging output, and several buttons for user authentication (signup and login for Hospital and Patient roles).
- **Main Loop:** Finally, the application enters the Tkinter main loop, which keeps the GUI active and responsive.

## 10.2 Results:

Figure 10.1 showcases the different categories present in the dataset. It visually depicts the class labels, helping to understand the distribution of image categories. The dataset represents medical images or health-related data, classified into multiple X-ray categories. Identifying the categories is crucial for training classification models, as it sets the foundation for defining the problem and the expected output classes.

```
['Cofield.jpg', 'Depuy.jpg', 'tornier.jpg', 'Zimmer.jpg']
```

Fig. 9.1: Categories of Dataset.

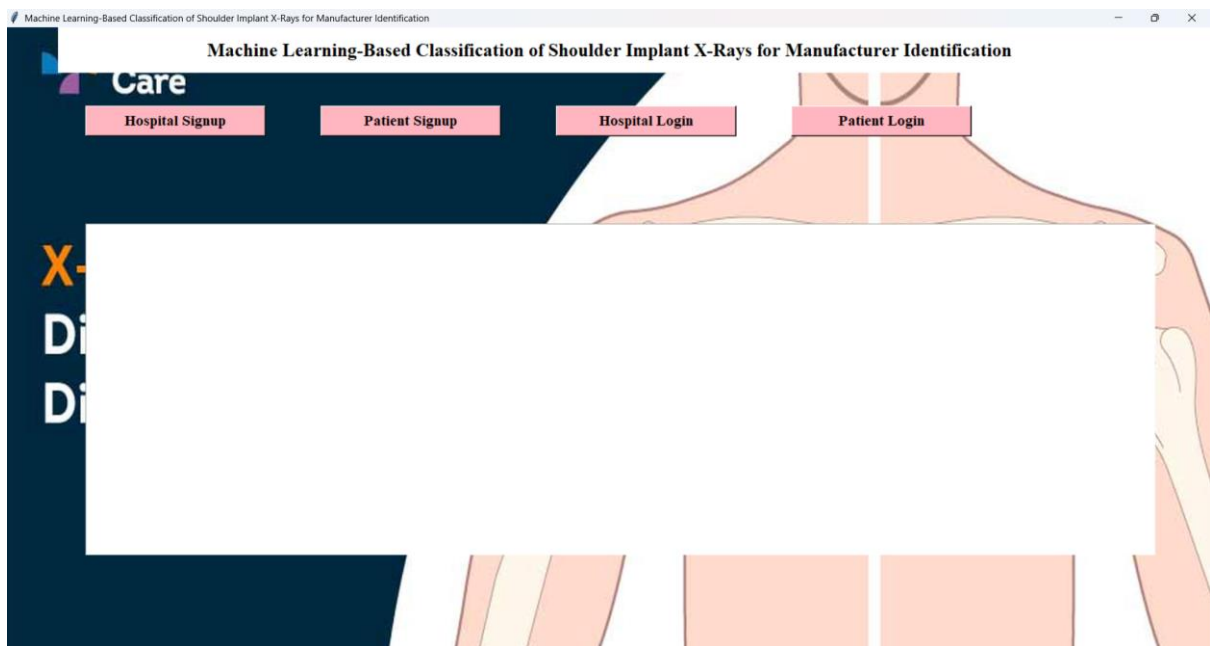


Fig. 10.2: GUI application of proposed ML-based classification of shoulder Implant X-rays for manufacturer identification.

Figure 10.2 presents the main interface of the application where the entire process of classifying shoulder implant X-rays is integrated. The application features a clear and prominent title that outlines its purpose, accompanied by an aesthetically pleasing background image that enhances

the user experience. The interface is designed with multiple navigation buttons that allow users to upload datasets, preprocess images, train models, and perform predictions. Additionally, a dedicated logging or output area is visible, where the system displays real-time status updates, progress reports, and error messages. The design is role-aware, providing distinct options for hospital administrators and patients, thereby ensuring that users have access to functionalities relevant to their respective roles.

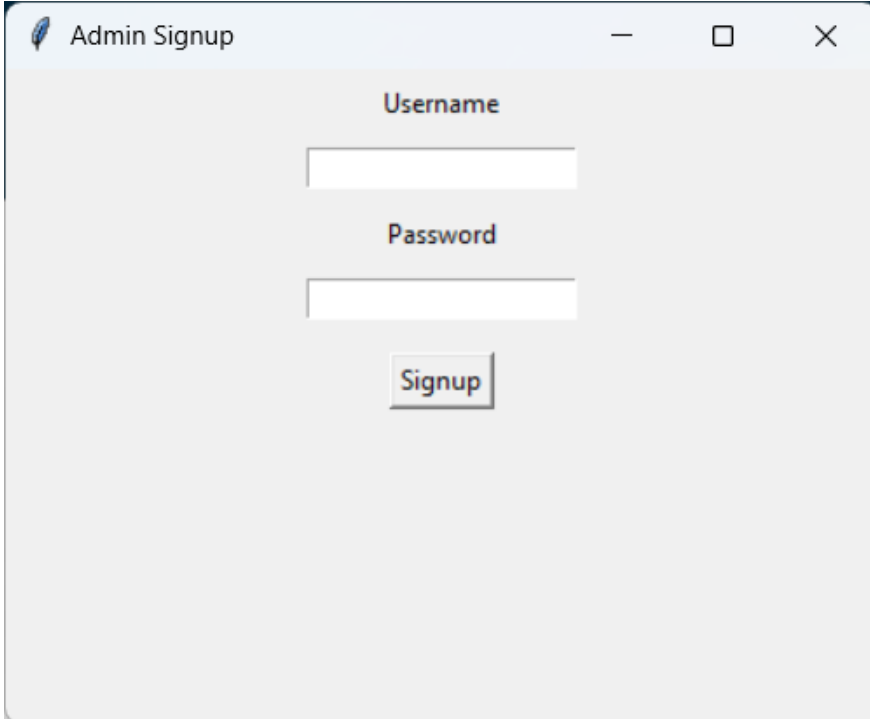
A screenshot of a web application window titled "Admin Signup". The window has a light gray background and a standard title bar with a feather icon, a minus sign, a square icon, and a close button. The form is centered and contains three elements: a label "Username" in blue text above a white input field, a label "Password" in blue text above another white input field, and a "Signup" button with a blue border and text. The button is positioned below the password field.

Fig. 10.3: Admin (hospital) signup interface.

In Figure 10.3, the signup interface designed specifically for hospital administrators is illustrated. The interface is streamlined to facilitate the quick registration of new hospital users. It contains clearly labeled input fields where the user is required to enter a username and a password. There is also a prominently placed signup button to submit the entered information. The layout is clean and user-friendly, ensuring that hospital personnel can easily register without encountering any confusion or unnecessary complexity. This interface plays a critical role in ensuring that only authorized hospital administrators gain access to the system's administrative functionalities.

Figure 10.4 depicts the signup interface for patients, which, while similar in structure to the admin signup, is tailored for the patient role. The design includes input fields for a username and password, along with a clear signup button for submitting the registration details. Although the overall layout mirrors that of the admin signup interface to maintain consistency, subtle

differences in styling or labelling help distinguish the patient signup process from that of the hospital administrators. The interface is designed to be intuitive, ensuring that patients can easily register and subsequently access patient-specific features, such as the prediction functionality and viewing of personal results.

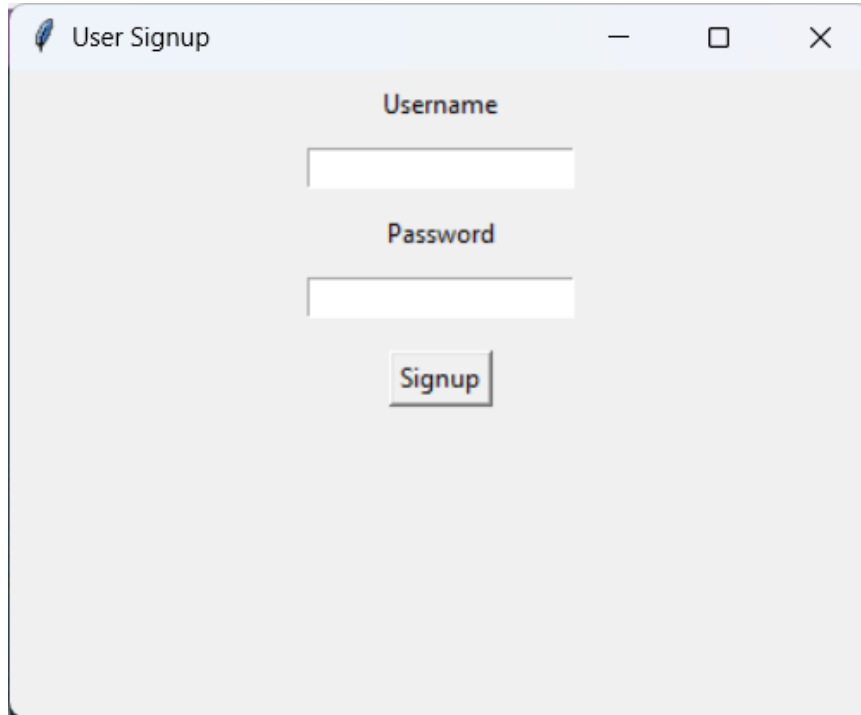
A screenshot of a web application window titled "User Signup". The window has a light blue header bar with the title and standard window controls (minimize, maximize, close). The main content area is light gray and contains three centered elements: a label "Username" above a white text input field, a label "Password" above another white text input field, and a "Signup" button below the fields. The button is a simple rectangle with a thin border and the text "Signup" in the center.

Fig. 10.4: User (patient) signup interface.

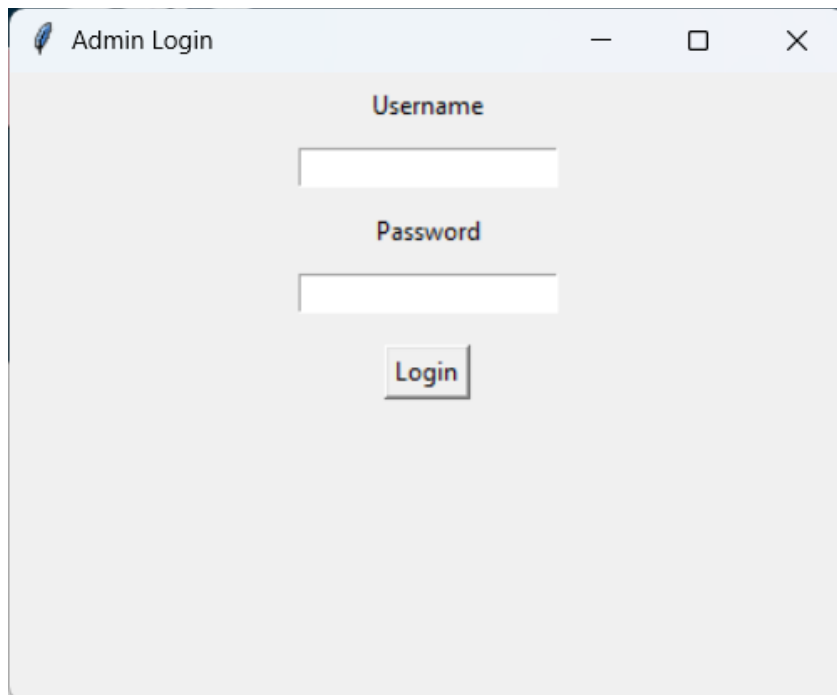
A screenshot of a web application window titled "Admin Login". The window has a light blue header bar with the title and standard window controls (minimize, maximize, close). The main content area is light gray and contains three centered elements: a label "Username" above a white text input field, a label "Password" above another white text input field, and a "Login" button below the fields. The button is a simple rectangle with a thin border and the text "Login" in the center.

Fig. 10.5: Admin login interface.

Figure 10.5 focuses on the login screen for hospital administrators. This interface includes dedicated input fields for the username and password, specifically configured for admin access. Clear instructional labels guide the user in providing the correct credentials, and a prominent login button is available to initiate the authentication process. The design of the login interface prioritizes security and ease of use, ensuring that hospital staff can quickly and securely access the system without confusion. This interface serves as the gateway for administrators to the full range of application functionalities, including data management and model training.

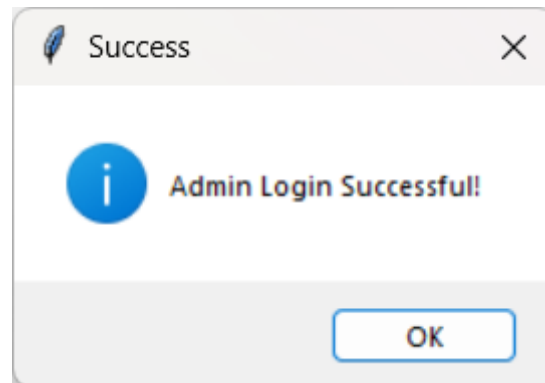


Fig. 10.6: Admin login successful interface after entering the authenticated username and password.

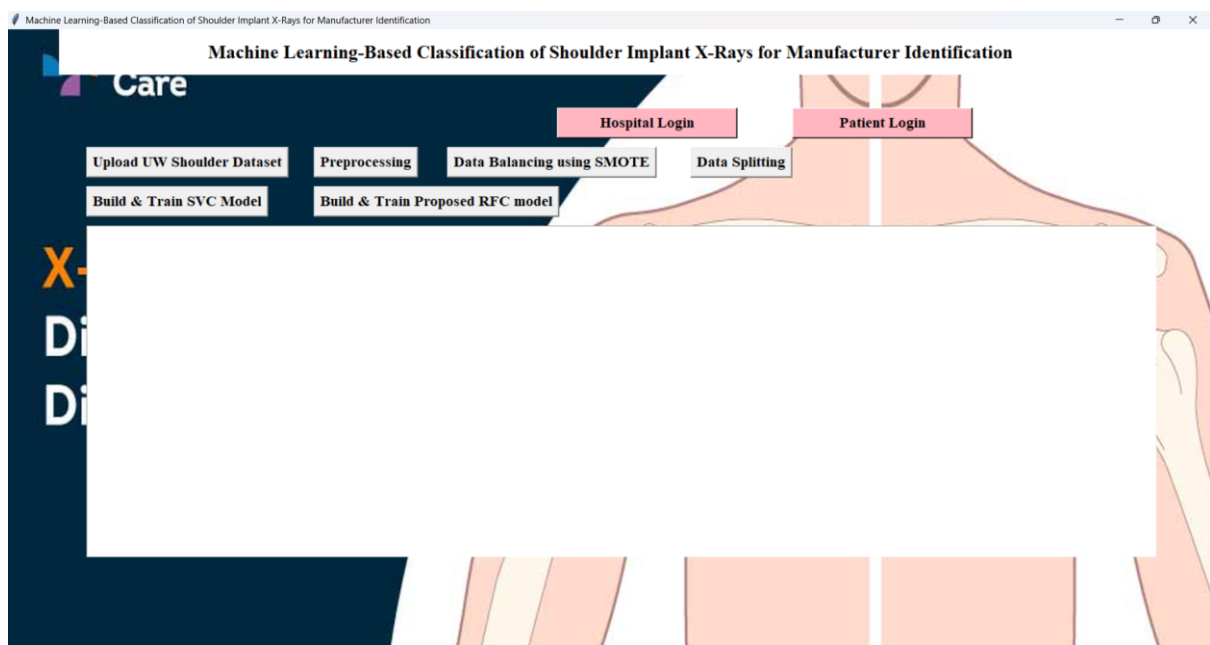


Fig. 10.7: GUI application after successful login of an admin (hospital).

After successfully entering the correct credentials, Fig. 9.6 illustrates the screen displayed to hospital administrators upon successful authentication. A confirmation cue is typically present,

clearly indicating that the login process has been completed successfully. This interface acts as an acknowledgment of successful login and signals the transition from authentication to operational mode, where the administrator can now access specialized features such as dataset upload, preprocessing, and model training.

Figure 10.7 represents the main dashboard of the application following a successful login by a hospital administrator. The interface now displays a set of admin-specific buttons and options that are not available to other user roles. These options include functionalities such as dataset upload, image preprocessing, data balancing using SMOTE, train/test splitting, and the training of both SVC and RFC machine learning models. Additionally, a text area continues to serve as a logging window, providing real-time updates on operations, processes, and potential error messages. The design of this interface ensures that all the necessary tools for managing and executing the classification workflow are accessible from a single, consolidated dashboard, thereby streamlining the administrative tasks involved in the machine learning process.

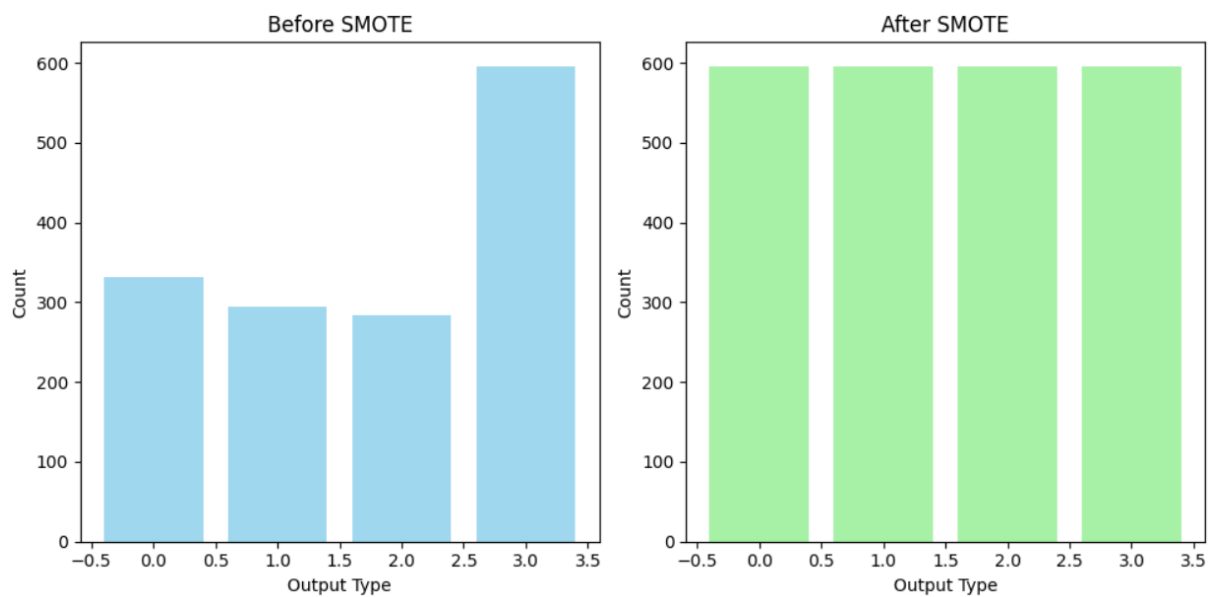
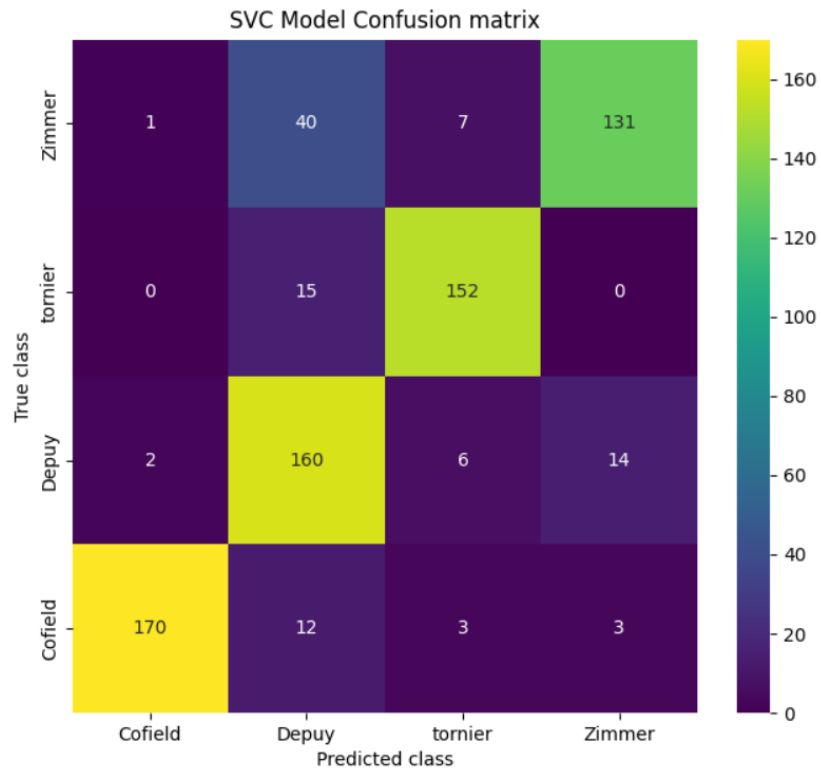
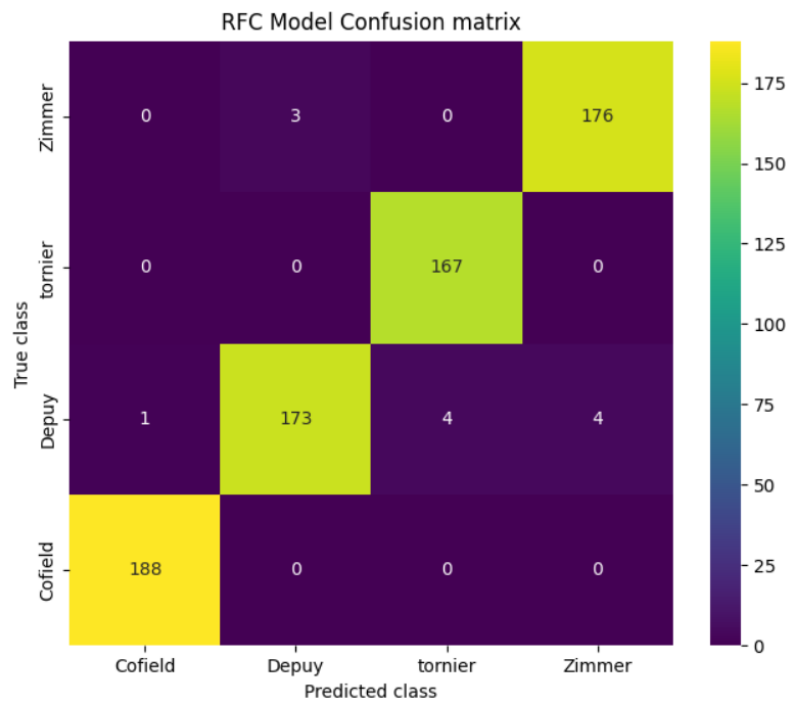


Fig. 10.8: Count distribution versus type of manufacturer. Before applying SMOTE (left).  
After applying SMOTE (right).

Figure 10.8 (left) displays the count plot, which shows the number of samples available in each category. It provides insight into the distribution of the dataset and helps identify potential class imbalances. Here, some categories have significantly fewer samples than others, which could impact model performance. Thus, SMOTE (Synthetic Minority Over-sampling Technique) is used to address such imbalances by oversampling the minority classes to ensure that the classifier is not biased toward the majority class as demonstrated in Fig. 10.8 (right).



(a)



(b)

Fig. 10.9: Confusion matrices obtained using (a) SVC model. (b) Proposed RFC model.

Figure 10.9 presents the confusion matrices obtained using SVC model, and proposed RFC model. The confusion matrix summarizes the performance of the models by showing how many instances were correctly or incorrectly classified. Each cell represents the number of true positive, false positive, true negative, and false negative predictions. This matrix is vital for evaluating the classifier's precision, recall, and accuracy, giving a deeper understanding of how well the model is performing across all categories. This figure allows you to compare the two classifiers side-by-side and decide which model provides better results based on metrics such as accuracy, precision, and recall for each category.

Table 10.1: Performance comparison of quality metrics obtained using SVC model, and proposed RFC model.

Model/Metrics	Accuracy	Precision	Recall	F1-score
SVC model	85.61	86.93	85.63	85.82
Proposed RFC model	98.32	98.30	98.34	98.31

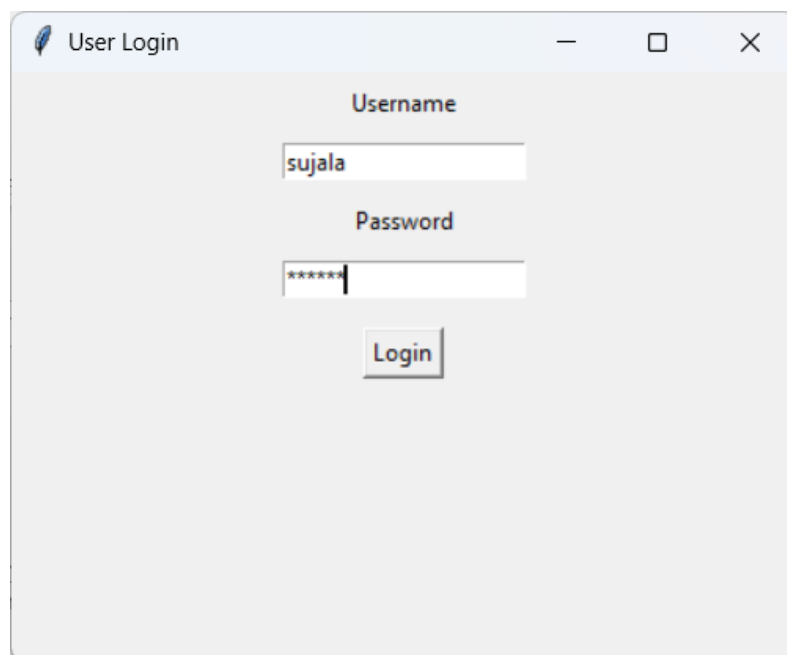


Fig. 10.10: User (patient) login interface.

Table 10.1 presents a side-by-side comparison of key performance metrics between the SVC model and the proposed RFC model. The metrics include Accuracy, Precision, Recall, and F1-score. The SVC model records an accuracy of 85.61%, with precision, recall, and F1-score around 86.93%, 85.63%, and 85.82% respectively. In contrast, the proposed RFC model



significantly outperforms the SVC, achieving an accuracy of 98.32% and nearly identical precision, recall, and F1-score values (98.30%, 98.34%, and 98.31% respectively). This comparison clearly highlights the superior performance of the RFC model in accurately classifying shoulder implant X-rays for manufacturer identification, demonstrating its robust predictive capabilities.

Figure 9.10 depicts the login interface specifically designed for patients. This screen includes straightforward input fields for the patient to enter their username and password. The layout is clean and user-friendly, ensuring that patients can easily navigate the login process without any complications. Clear labels and a prominent login button guide users through the authentication process, ensuring that only registered patients gain access to the system. The design of this interface is intended to provide a secure yet accessible entry point for patients, enabling them to access patient-specific functionalities such as testing new shoulder implant X-ray images.

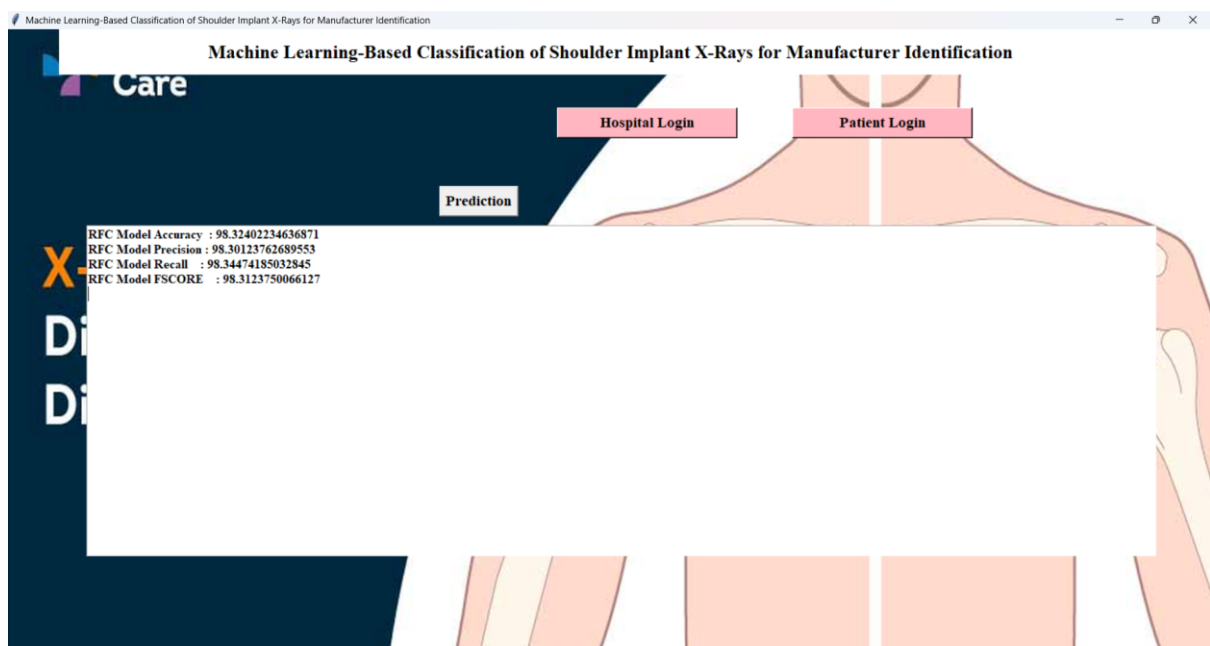


Fig. 10.11: GUI application after login as a user for testing the new shoulder implant X-ray images.

Figure 10.11 illustrates the main dashboard that patients see after successfully logging into the application. This user interface is tailored for patient interactions, focusing on the testing and prediction aspect of the application. The screen provides access to functionalities that allow patients to upload and test new shoulder implant X-ray images. Upon uploading an image, the system processes the image, applies the trained machine learning model, and displays the predicted manufacturer along with any additional relevant information. The layout remains

intuitive and uncluttered, ensuring that patients can effortlessly navigate through the testing process while receiving real-time feedback and results on the classification of the new implant images.



Fig. 10.12: Sample predictions on test images.

Figure 12 illustrates the outcome of the prediction on sample test data using proposed RFC model. The test image is processed through the classification pipeline, and the predicted category is displayed along with the actual image. The visual overlay of the prediction on the test image provides an intuitive understanding of the model's effectiveness in real-world scenarios. It helps in verifying whether the model is correctly classifying unseen data.

## CHAPTER 11

### CONCLUSION AND FUTURE SCOPE

The project demonstrates a comprehensive approach to classifying shoulder implant X-ray images for manufacturer identification using machine learning techniques. The system integrates a robust preprocessing pipeline that includes image resizing, flattening, and class balancing using SMOTE to ensure reliable input data. Two distinct classification models were developed—a Support Vector Classifier (SVC) and a Random Forest Classifier (RFC)—to compare their performance in accurately identifying the manufacturer from the processed X-ray images. While the SVC model achieved respectable performance with an accuracy of 85.61% and closely matched precision, recall, and F1-scores, the proposed RFC model significantly outperformed it, boasting an accuracy of 98.32% with similarly high values for precision, recall, and F1-score. This superior performance is primarily attributed to the ensemble nature of the Random Forest, which combines multiple decision trees and leverages majority voting to reduce overfitting and handle high-dimensional data effectively. The application is designed with an intuitive GUI using Tkinter, which supports both administrative (hospital) and user (patient) functionalities. The admin interface enables hospital staff to manage dataset uploads, preprocessing, model training, and evaluation, while the user interface allows patients to log in and test new X-ray images easily. This role-based design not only enhances security but also ensures that each user type can access the specific functionalities tailored to their needs. The integration of a MySQL database for user authentication further strengthens the system's reliability and usability.

Future work explores the integration of deep learning approaches for even higher accuracy and automated feature extraction. Additionally, incorporating real-time data and multi-modal imaging could further enhance the system's diagnostic capabilities.

## REFERENCES

- [1] AI-driven optimization in healthcare: the diagnostic process. Lyon J, Bogodistov Y, Moormann J. *Eur J Manage Issues*. 2021; 29:218–231
- [2] Evolving scenario of big data and artificial intelligence (AI) in drug discovery. Tripathi MK, Nath A, Singh TP, Ethayathulla AS, Kaur P. *Mol Divers*. 2021;25:1439–1460
- [3] Drawbacks of artificial intelligence and their potential solutions in the healthcare sector. Khan B, Fatima H, Qureshi A, Kumar S, Hanan A, Hussain J, Abdullah S. *Biomed Mater Devices*. 2023:1–8.
- [4] Artificial intelligence in breast cancer screening and diagnosis. Dileep G, Gianchandani Gyani SG. *Cureus*. 2022;14:0
- [5] A deep learning approach to generate contrast-enhanced computerized tomography angiograms without the use of intravenous contrast agents. Chandrashekar A, Shivakumar N, Lapolla P, et al. *Eur Heart J*. 2020;41:0.
- [6] Assessing the accuracy of an automated atrial fibrillation detection algorithm using smartphone technology: the iREAD Study. William AD, Kanbour M, Callahan T, et al. *Heart Rhythm*. 2018;15:1561–1565.
- [7] Using artificial intelligence to detect COVID-19 and community-acquired pneumonia based on pulmonary CT: evaluation of the diagnostic accuracy. Li L, Qin L, Xu Z, et al. *Radiology*.
- [8] Deep learning based software to identify large vessel occlusion on noncontrast computed tomography. Olive-Gadea M, Crespo C, Granes C, et al. *Stroke*
- [9] AI-driven decision making for auxiliary diagnosis of epidemic diseases. Lin K, Liu J, Gao J. *IEEE Transact Mol Biol Multi-Scale Commun*. 2022;8:9–16.
- [10] The future of artificial intelligence in neurosurgery: a narrative review. Iqbal J, Jahangir K, Mashkoor Y, et al. *Surg Neurol Int*. 2022;13:536.
- [11] Deep feature learning for sudden cardiac arrest detection in automated external defibrillators. Nguyen MT, Nguyen BV, Kim K. *Sci Rep*. 2018;8:17196
- [12] A survey on AI techniques for thoracic diseases diagnosis using medical images. Mostafa FA, Elrefaei LA, Fouda MM, Hossam A. *Diagnostics (Basel)* 2022;12:3034.

- [13] AI-driven clinical decision support: enhancing disease diagnosis exploiting patients similarity. Comito C, Falcone D, Forestiero A. *IEEE Access*. 2022;10:6878–6888.
- [14] Deep neural networks are superior to dermatologists in melanoma image classification. Brinker TJ, Hekler A, Enk AH, et al. *Eur J Cancer*. 2019;119:11–17.
- [15] Santosh K, Gaur L. *Artificial Intelligence and Machine Learning in Public Healthcare*. Singapore: Springer; 2021. AI solutions to public health issues; pp. 23–32.