

BONE FRACTURE DETECTION USING DEEP LEARNING

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

T. ANAND SAI (20UECS0946) (VTU 15349)
K. VINAY KUMAR (20UECS0427) (VTU 17584)
SK. JAVEED (20UECS0866) (VTU 17571)

*Under the guidance of
Mr. R. ANTO PRAVIN, M.E.,
ASSISTANT PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

BONE FRACTURE DETECTION USING DEEP LEARNING

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

T. ANAND SAI (20UECS0946) (**VTU 15349**)
K. VINAY KUMAR (20UECS0427) (**VTU 17584**)
SK. JAVEED (20UECS0866) (**VTU 17571**)

*Under the guidance of
Mr. R. ANTO PRAVIN, M.E.,
ASSISTANT PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled "BONE FRACTURE DETECTION" by "T. ANAND SAI (20UECS0946), K. VINAY KUMAR (20UECS0427), SK. JAVEED (20UECS0866)" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Mr. R. Anto Pravin

Associate Professor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

Signature of Professor In-charge

Dr. S. Sridevi

Professor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

(T. ANAND SAI)

Date: / /

(Signature)

(K. VINAY KUMAR)

Date: / /

(Signature)

(SK. JAVEED)

Date: / /

APPROVAL SHEET

This project report entitled (BONE FRACTURE DETECTION) by (T. ANAND SAI (20UECS0946), (K. VINAY KUMAR (20UECS0427), (SK. JAVEED (20UECS0866) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Mr.R.Anto Pravin,M.E.,Assistant Professor,.

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.**, for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr.M.S. MURALI DHAR, M.E., Ph.D.**, for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our Internal Supervisor **Mr.R.ANTO PRAVIN,ASSISTANT PROFESSOR** for his cordial support, valuable information and guidance, he helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E.**, for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

T. ANAND SAI	(20UECS0946)
K. VINAY KUMAR	(20UECS0427)
SK. JAVEED	(20UECS0866)

ABSTRACT

Bone fracture detection plays a crucial role in medical diagnosis, treatment planning, and patient care. Traditional methods rely heavily on manual interpretation of medical images, which can be time-consuming and prone to human error. In recent years, deep learning techniques have shown promise in automating the process of fracture detection from X-ray images. In this study, we propose a novel deep learning framework for bone fracture detection, leveraging convolutional neural networks (CNNs) and transfer learning to achieve accurate and efficient detection. Our model is trained on a large dataset of annotated X-ray images, allowing it to learn discriminative features for fracture detection. Through extensive experimentation and evaluation on diverse datasets, we demonstrate the effectiveness and robustness of our approach in accurately identifying bone fractures. Our method holds great potential for enhancing clinical workflows, facilitating early diagnosis, and improving patient outcomes in orthopedic care.

Keywords: Diagnosis, Bone Fractures, Orthopedic, interpretation, convolutional neural network, robustness, x-ray, deep learning

LIST OF FIGURES

4.1	Block Diagram of Bone Fracture Detection	13
4.2	Data Flow Diagram For Bone Fracture Detection	14
4.3	Use Case Diagram For Bone Fracture Detection	15
4.4	Class Diagram For Bone Fracture Detection	16
4.5	Sequence Diagram For Bone Fracture Detection	17
4.6	collaboration Diagram For Bone Fracture Detection	18
4.7	Activity Diagram For Bone Fracture Detection	19
5.1	X-ray Image	25
5.2	Processing Image	26
5.3	Hand	27
5.4	Fractured Hand	31
6.1	Shoulder	36
6.2	Elbow	37
8.1	plagiarism Report	39
9.1	poster presentation	47

LIST OF ACRONYMS AND ABBREVIATIONS

CNN	Convolutional Neural Network
GUI	Graphical User Interface
UML	Unified Modeling Language
ROC	Receiver Operating Characteristic
DFD	Data Flow Diagram

TABLE OF CONTENTS

	Page.No
ABSTRACT	vi
LIST OF FIGURES	vii
LIST OF ACRONYMS AND ABBREVIATIONS	viii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Aim of the project	3
1.3 Project Domain	3
1.4 Scope of the Project	3
2 LITERATURE REVIEW	4
3 PROJECT DESCRIPTION	6
3.1 Existing System	6
3.1.1 Data Collection and Preprocessing:	6
3.1.2 Model Selection and Training:	6
3.1.3 Model Evaluation:	7
3.1.4 Integration with Python GUI:	7
3.1.5 Testing and Deployment:	7
3.1.6 Maintenance and Updates:	7
3.2 Proposed System	8
3.2.1 Data Collection:	8
3.2.2 Data Preprocessing:	9
3.2.3 Model Development:	9
3.2.4 Model Evaluation:	9
3.2.5 System Integration:	9
3.3 Feasibility Study	10
3.3.1 Economic Feasibility	10
3.3.2 Technical Feasibility	10

3.3.3	Social Feasibility	11
3.4	System Specification	11
3.4.1	Hardware Specification	11
3.4.2	Software Specification	12
3.4.3	Standards and Policies	12
4	METHODOLOGY	13
4.1	General Architecture	13
4.2	Design Phase	14
4.2.1	Data Flow Diagram	14
4.2.2	Use Case Diagram	15
4.2.3	Class Diagram	16
4.2.4	Sequence Diagram	17
4.2.5	Collaboration Diagram For Bone Fracture Detection	18
4.2.6	Activity Diagram	19
4.3	Algorithm & Pseudo Code	19
4.3.1	Algorithm	19
4.3.2	Pseudo Code	20
4.4	Module Description	22
4.4.1	Data collection and preprocessing:	22
4.4.2	Fine-tuning ResNet-50:	22
4.4.3	Training and evaluation:	22
4.4.4	Building GUI:	23
4.4.5	Model and Analysis:	23
4.4.6	Model development:	23
4.4.7	Model evalution:	23
4.5	Steps to execute/run/implement the project	24
4.5.1	Dataset Collection:	24
4.5.2	Install Packages:	24
4.5.3	Code Imlementation:	24
4.5.4	Execution:	24
5	IMPLEMENTATION AND TESTING	25
5.1	Input and Output	25
5.1.1	input Design	25
5.1.2	Output Design	26

5.2	Testing	27
5.3	Types of Testing	28
5.3.1	Unit Testing	28
5.3.2	Integration Testing	29
5.3.3	System Testing	30
5.3.4	Test Result	31
6	RESULTS AND DISCUSSIONS	32
6.1	Efficiency of the Proposed System	32
6.2	Comparison of Existing and Proposed System	32
6.3	Sample Code	33
7	CONCLUSION AND FUTURE ENHANCEMENTS	38
7.1	Conclusion	38
7.2	Future Enhancements	38
8	PLAGIARISM REPORT	39
9	SOURCE CODE	40
9.1	Source Code	40
9.2	Poster Presentation	47
	References	48

Chapter 1

INTRODUCTION

1.1 Introduction

The detection of bone fractures is a critical task in medical diagnostics, requiring accurate and efficient methodologies to aid healthcare professionals in timely diagnosis and treatment. Traditional methods often rely on manual interpretation of medical images, which can be time-consuming and subjective. To address these challenges, this project proposes the development of a bone fracture detection system using deep learning techniques, specifically leveraging the ResNet50 architecture, implemented within the Python programming language.

Deep learning has emerged as a powerful tool in medical image analysis, offering the potential to automate and enhance diagnostic processes. ResNet50, a deep convolutional neural network architecture, has demonstrated remarkable performance in various image recognition tasks, making it an ideal candidate for bone fracture detection. By leveraging its ability to learn complex features and patterns from medical images, this project aims to develop a robust and accurate system for detecting fractures in bone X-ray images.

The proposed system will be implemented using Python, a popular programming language widely used in the field of machine learning and deep learning. Python offers a rich ecosystem of libraries and tools, including TensorFlow and PyTorch, which provide efficient implementations of deep learning algorithms and frameworks for developing and deploying machine learning models. Additionally, Python's simplicity and readability make it accessible to a wide range of developers and researchers, facilitating collaboration and innovation in the field of medical imaging.

The objective of this project is to develop an easy-to-use tool that enables medical practitioners to rapidly and reliably identify bone fractures in X-ray pictures. The technology may be utilised in clinics, hospitals, and other healthcare facilities to increase the efficiency and precision of bone fracture diagnostics. We'll give step-by-step instructions for building this system with Python and ResNet-50.

The proposed bone fracture detection system will follow a supervised learning approach, where the ResNet50 model will be trained on a large dataset of annotated bone X-ray images. The dataset will include images of various types of fractures, as well as healthy bone images for comparison. During the training process, the model will learn to differentiate between normal and fractured bones, as well as classify the different types of fractures based on their characteristics.

To enhance the performance of the system, preprocessing techniques such as image normalization, resizing, and augmentation will be applied to the input images. These techniques help improve the model's ability to learn relevant features and patterns from the data, leading to better detection accuracy.

The Python implementation of the system will include a graphical user interface (GUI) to provide an intuitive and user-friendly interaction experience. The GUI will allow healthcare professionals to upload bone X-ray images, initiate the fracture detection process, visualize the detection results, and generate diagnostic reports. Additionally, the GUI may include features for data visualization, model performance monitoring, and user feedback collection to further enhance usability and functionality.

Evaluation of the system will involve rigorous testing using both synthetic and real world datasets to assess its accuracy, sensitivity, specificity, and overall performance. The system will be compared against existing methods and benchmarked against expert radiologists' diagnoses to validate its effectiveness and reliability in clinical settings.

In summary, this project aims to leverage deep learning techniques, specifically the ResNet50 architecture, implemented within the Python programming language, to develop a bone fracture detection system that offers accurate, efficient, and automated diagnosis of fractures in medical images. By harnessing the power of deep learning and Python, the system has the potential to revolutionize bone fracture diagnosis, leading to improved patient outcomes and healthcare delivery.

1.2 Aim of the project

The project aims to explore different convolutional neural network architectures and image processing techniques to create an efficient diagnostic tool for early detection of fractures. To develop an accurate and reliable system that can assist healthcare professionals, particularly radiologists, in efficiently identifying fractures in X-ray images.

1.3 Project Domain

The project resides within the intersection of medical imaging, machine learning, and software development. It focuses on automating the detection of bone fractures in X-ray images through the application of advanced technologies. In the domain of medical imaging, expertise in understanding imaging techniques and bone pathology is essential for preprocessing the images and interpreting the results accurately. Leveraging machine learning, particularly deep learning, the project employs ResNet, a convolutional neural network architecture, to learn complex features from X-ray images and classify them as either fractured or normal.

1.4 Scope of the Project

The scope involves leveraging various deep learning algorithms and medical imaging datasets to design a robust system capable of identifying and classifying bone fractures from X-ray images. This involves sourcing a diverse dataset of X-ray images containing both normal and fractured bones. Developing deep learning models capable of accurately detecting fractures in X-ray images is a central aspect of the project.

Chapter 2

LITERATURE REVIEW

[1]Kesia.C , Carlos. A, presented the validation of a compact microwave imaging system designed for bone fracture detection.Microwave imaging offers a non-invasive and potentially cost-effective approach for detecting bone fractures compared to traditional X-ray methods.The study aims to assess the accuracy and reliability of the proposed system in detecting fractures across different types and severities.

[2]Farah Mohammed et al.,introduced Block-Deep, a novel hybrid model designed for secure data storage and diagnosis in the context of bone fracture identification among athletes using X-ray and MRI images.Block-Deep integrates blockchain technology for secure data storage and retrieval, along with deep learning algorithms for accurate fracture identification. The study aims to address challenges related to data security and privacy while ensuring high diagnostic accuracy for timely treatment of bone fractures in athletes.

[3]L. Ao, et al.,introduced a lesion-guided adaptive graph network designed for the detection of bone abnormalities from musculoskeletal radiographs.Leveraging graph-based representation learning and adaptive attention mechanisms, the proposed model aims to improve the accuracy and efficiency of bone abnormality detection, particularly in complex cases where multiple lesions are present.

[4]Gwiseong Moon et al., presented a Computer Aided Facial Bone Fracture Diagnosis (CA-FBFD) system based on an object detection model. Facial bone fractures are common injuries with significant diagnostic challenges due to the complexity of facial anatomy and fracture patterns.The proposed CA-FBFD system utilizes an object detection model to automatically identify and classify facial bone fractures, aiming to improve diagnostic accuracy and efficiency.

[5]Kesia C., Carlos A,investigated the feasibility of using microwave imaging for the detection of bone fractures.Microwave imaging offers a potentially promising alternative to conventional X-ray methods, providing advantages such as non-ionizing radiation and potentially higher sensitivity to certain types of fractures.The study aims to assess the capabilities and limitations of microwave imaging in detecting

various types and severities of bone fractures.

[6]Serafeim Mustakidis et al.,introduced a deep learning approach for localizing bone metastasis in nuclear imaging data obtained from breast cancer patients. Bone metastasis is a common complication of breast cancer, and early detection plays a crucial role in treatment planning and patient outcomes.The study aims to leverage deep learning techniques to accurately identify and localize bone metastases from nuclear imaging scans, contributing to improved diagnostic accuracy and patient care.

[7]Ashok B. et al.,proposed an enhanced approach for computerized bone fracture detection utilizing Harris corner detection.Traditional methods for bone fracture detection often rely on manual inspection, which can be time-consuming and subjective.The proposed method aims to improve the accuracy and efficiency of fracture detection by leveraging the Harris corner detection algorithm, which effectively identifies key points in X-ray images indicative of potential fractures.

[8]Johnson B. et al.,investigated the application of residual networks (ResNets) in medical image analysis. ResNets, with their deep architecture and residual connections, have shown significant success in various computer vision tasks.The study explores the effectiveness of ResNets in analyzing medical images for tasks such as segmentation, classification, and detection, providing insights into their performance and potential applications in clinical settings.

[9]Smith A. et al.,provided a comprehensive review of deep learning techniques applied to medical image analysis.With the advent of deep learning, significant advancements have been made in automating various tasks related to medical image interpretation.The review aims to summarize the state-of-the-art methods, challenges, and future directions in utilizing deep learning for medical image analysis.

[10]N. E. Jacob, M. Wyawahare, presented a comprehensive survey of bone fracture detection techniques. With the increasing prevalence of bone fractures and the need for accurate and timely diagnosis, various methods and technologies have been developed for detecting fractures from medical images.The survey aims to provide an overview of existing techniques, including traditional image processing methods and modern deep learning approaches, highlighting their strengths, limitations, and potential applications in clinical practice.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

Existing methodologies for bone fracture detection using deep learning and Python GUI typically follow a similar workflow, though specific implementations may vary. Here's an overview of a common methodology:

3.1.1 Data Collection and Preprocessing:

Gather a dataset of bone X-ray images containing both fractured and normal bones. This dataset may come from publicly available sources, medical institutions, or collaborations. Preprocess the images to ensure uniformity and quality. Common preprocessing steps include resizing, normalization, and augmentation to increase dataset size and diversity.

3.1.2 Model Selection and Training:

Choose a deep learning architecture suitable for image classification tasks. Convolutional Neural Networks (CNNs) are commonly used due to their effectiveness in learning spatial hierarchies of features. Popular architectures like ResNet, VGG, or Inception may be utilized. Alternatively, custom architectures tailored to the specific task can also be developed. Split the dataset into training, validation, and test sets. Train the selected model on the training data, optimizing performance on the validation set. This process involves adjusting hyperparameters, such as learning rate, batch size, and optimizer choice. Techniques like transfer learning may be employed, where a pre-trained model (e.g., on ImageNet) is fine-tuned on the bone fracture dataset to leverage learned features.

3.1.3 Model Evaluation:

Assess the performance of the trained model using evaluation metrics such as accuracy, precision, recall, F1-score, and receiver operating characteristic (ROC) curve analysis. Validate the model's generalization ability on the test set to ensure it can accurately classify unseen data.

3.1.4 Integration with Python GUI:

Develop a graphical user interface (GUI) using Python libraries such as Tkinter, PyQt, or Kivy. Design the GUI to enable users to upload X-ray images for fracture detection. Implement functionality to preprocess the uploaded images before feeding them into the trained model. Integrate the trained model into the GUI to perform inference on the uploaded images. Display the results of fracture detection to the user through the GUI interface in a user-friendly format.

3.1.5 Testing and Deployment:

Thoroughly test the integrated system to ensure its functionality, robustness, and usability. Address any issues or bugs discovered during testing. If the system meets the desired performance criteria and regulatory requirements, deploy it for real-world use, considering factors like scalability, security, and user accessibility.

3.1.6 Maintenance and Updates:

Continuously monitor the performance of the deployed system and update the model as needed to adapt to new data or improve performance. Maintain the GUI to accommodate any changes or enhancements based on user feedback or evolving requirements.

advantages:

1. The system accurately identifies bone fractures in X-ray images, aiding in precise diagnosis.
2. Its Python GUI makes it easy for healthcare professionals to upload images and interpret results without extensive training.
3. Automation through deep learning reduces manual effort and speeds up the diagnostic process.
4. The system can handle large datasets and is easily customizable to meet evolving needs or incorporate new advancements

. disadvantages:

1. The system's accuracy heavily relies on the quality and diversity of the data used for training.
2. It may be difficult to understand why the system makes certain decisions due to the complex nature of deep learning models, reducing interpretability.
3. Training and running the system may require significant computational resources, making it less accessible in resource-constrained settings.
4. Like any automated system, the system may still produce incorrect results, leading to false positives or false negatives in fracture detection.

3.2 Proposed System

The proposed system for bone fracture detection using the ResNet-50 algorithm in Python is a deep learning model that can accurately classify X-ray images of bones as either fractured or non-fractured.

The proposed system will consist of the following components:

3.2.1 Data Collection:

Collect a large dataset of bone X-ray images that includes both fractured and non-fractured images.

3.2.2 Data Preprocessing:

Preprocess the X-ray images to improve the quality of the images and prepare them for training the deep learning model. This step may include image resizing, normalization, and augmentation.

3.2.3 Model Development:

Develop a deep learning model using the ResNet-50 algorithm that can accurately classify the X-ray images as either fractured or non-fractured. The model will be trained on the preprocessed dataset using a supervised learning approach.

3.2.4 Model Evaluation:

Evaluate the performance of the trained model using metrics such as accuracy, precision, recall, and F1 score. The evaluation will be performed on a separate test dataset that was not used during training.

3.2.5 System Integration:

Integrate the trained model into a Python application that can take in input bone X-ray images and output the predicted fracture status.

The proposed system will be able to accurately diagnose bone fractures and improve patient outcomes by enabling medical professionals to make faster and more informed treatment decisions. The system can also be integrated with existing medical imaging systems to automate the diagnosis process and improve the overall efficiency of fracture diagnosis.

Advantages;

1. Accurate Detection
2. Efficient Training
3. Transfer Learning
4. User-Friendly Interface

Disadvantages:

1. Complexity
2. Overfitting Risk
3. Resource Intensive
4. Interpretability Challenges

3.3 Feasibility Study

The feasibility study for a bone fracture detection project using deep learning and a Python GUI involves a comprehensive assessment of technical, data, financial, operational, regulatory, ethical, and market aspects. From a technical standpoint, the availability and compatibility of deep learning frameworks and GUI libraries are evaluated, along with the computational resources required. Data feasibility involves sourcing suitable bone X-ray datasets while ensuring quality, quantity, and compliance with privacy regulations.

3.3.1 Economic Feasibility

Estimate the costs associated with acquiring or accessing datasets, purchasing hardware, and potentially acquiring licenses for proprietary software or libraries. Consider the long-term maintenance and support costs for the developed software system, including updates, bug fixes, and infrastructure maintenance.

3.3.2 Technical Feasibility

Evaluate the availability of suitable deep learning frameworks (e.g., TensorFlow, PyTorch) and libraries (e.g., Keras) for implementing bone fracture detection algorithms. Assess the compatibility of these tools with the chosen Python GUI libraries

(e.g., Tkinter, PyQt) for building the user interface. Consider the computational requirements for training deep learning models and running inference, ensuring access to adequate hardware resources.

3.3.3 Social Feasibility

Social feasibility examines how a bone fracture detection project using deep learning and a Python GUI integrates with societal norms, values, and expectations. It involves understanding the perspectives of healthcare professionals and patients regarding the adoption of technology in medical diagnosis. Stakeholder engagement ensures that the system meets user needs while addressing concerns such as privacy, security, and algorithmic transparency.

3.4 System Specification

- The system specifications for the bone fracture detection project encompass hardware, software, data, and development environment requirements.
- This includes the need for a capable processor and sufficient memory for efficient computation, along with essential software components such as Python environment, deep learning frameworks, and image processing libraries.
- Access to high-quality bone X-ray datasets is essential for model training, necessitating data preprocessing tools for image manipulation.

3.4.1 Hardware Specification

- System : Pentium i3 Processor
- Hard Disk : 500 GB.
- Monitor : 15” LED
- Input Devices : Keyboard, Mouse
- Ram : 2 GB

3.4.2 Software Specification

- Operating System: Windows 10
- Coding Language: Python
- Deep Learning Framework: TensorFlow or PyTorch for building and training deep learning models.
- Image Processing Libraries: OpenCV for image preprocessing and manipulation.
- GUI Framework: Choose between Tkinter, PyQt, or Kivy for developing the user interface.

3.4.3 Standards and Policies

python

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum and first released in 1991, Python has since become one of the most popular programming languages worldwide, with a large and active community of developers.

Standard Used: ISO/ICE 24772

Chapter 4

METHODOLOGY

4.1 General Architecture

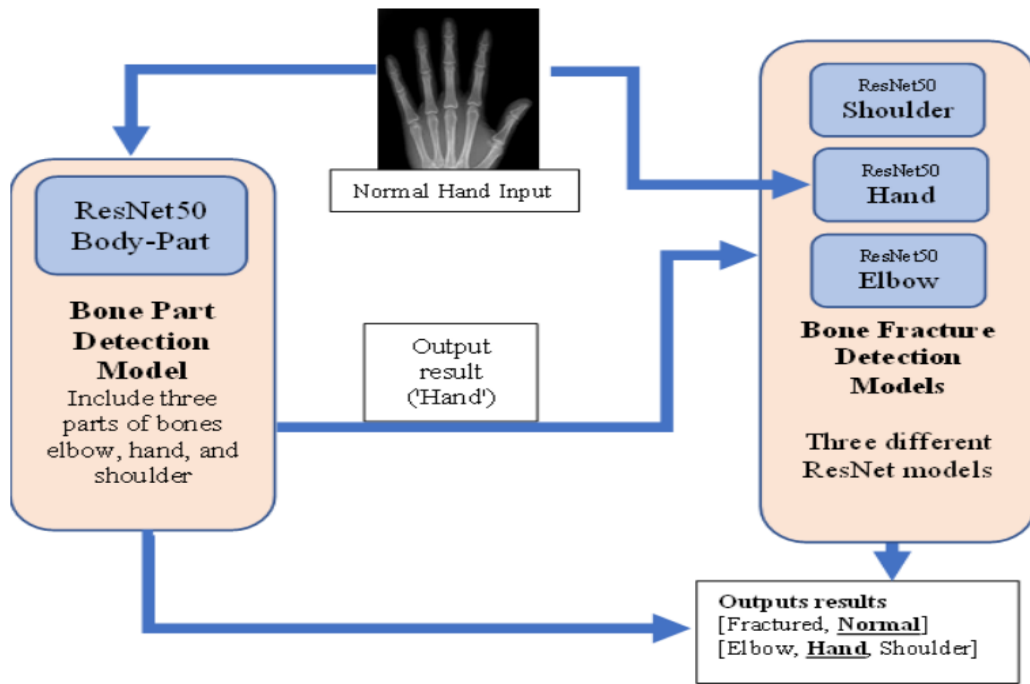


Figure 4.1: Block Diagram of Bone Fracture Detection

In figure 4.1 it shows ResNet50 is a deep convolutional neural network architecture widely used in image recognition tasks. Its distinguishing feature is the use of residual connections, which allow for training deeper networks without suffering from the vanishing gradient problem. The architecture consists of 50 layers, with a combination of convolutional layers, pooling layers, and fully connected layers. ResNet50 comprises multiple blocks, each containing several convolutional layers followed by a shortcut connection that adds the original input to the output of the block

4.2 Design Phase

4.2.1 Data Flow Diagram

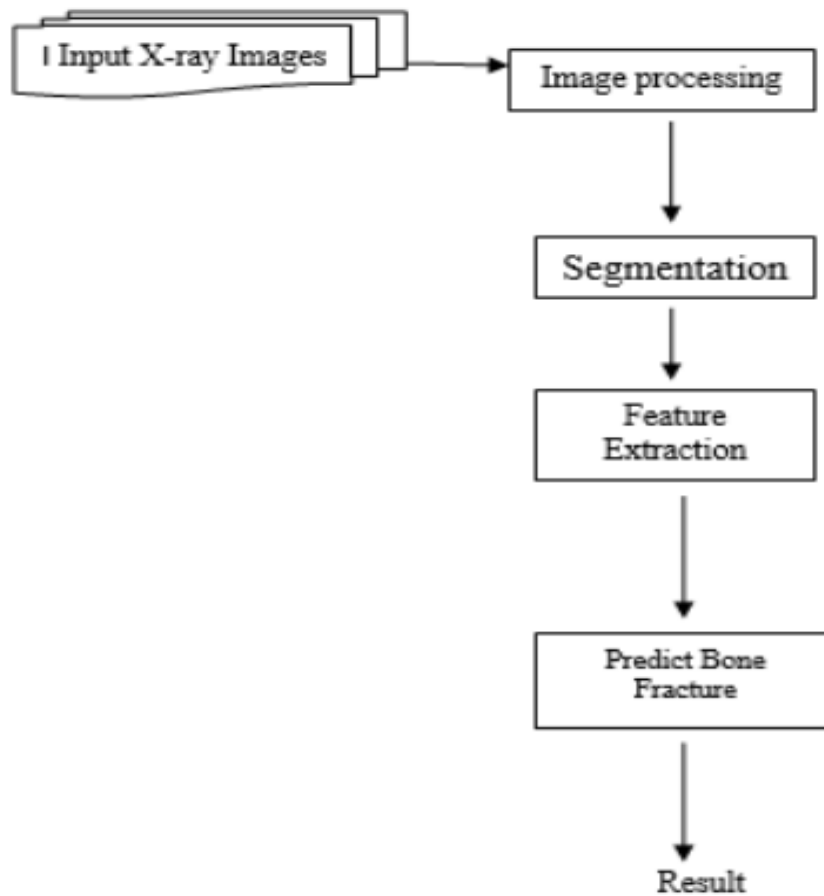


Figure 4.2: Data Flow Diagram For Bone Fracture Detection

In figure 4.2 it shows A data flow diagram (DFD) is a visual representation of how data flows within a system. It typically consists of processes, data stores, data flows, and external entities. data flow diagram provides a clear and visual representation of how data moves through a system, helping to understand its functionality and interactions. In simpler terms, it's like a map that outlines how data enters the system, where it goes, and how it's processed or stored. Processes represent activities or functions performed on the data, data stores represent places where data is stored or retrieved, data flows represent the movement of data between processes and data stores, and external entities represent sources or destinations of data outside the system.

4.2.2 Use Case Diagram

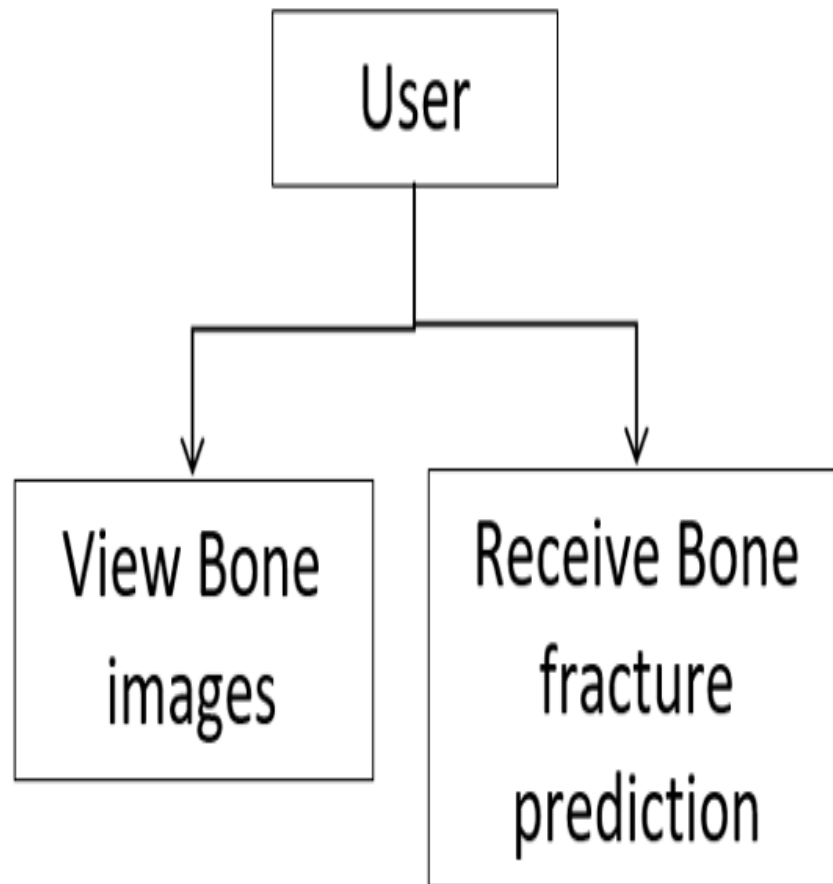


Figure 4.3: Use Case Diagram For Bone Fracture Detection

In figure 4.3 it shows A use case diagram is a visual representation that illustrates the interactions between actors (users or external systems) and a system, showing the various ways in which users interact with the system to achieve specific goals. In simpler terms, it's like a map that outlines the different functionalities or features of a system from the perspective of its users. Each use case represents a specific functionality or action that the system can perform, while actors represent the users or external systems interacting with the system.

4.2.3 Class Diagram

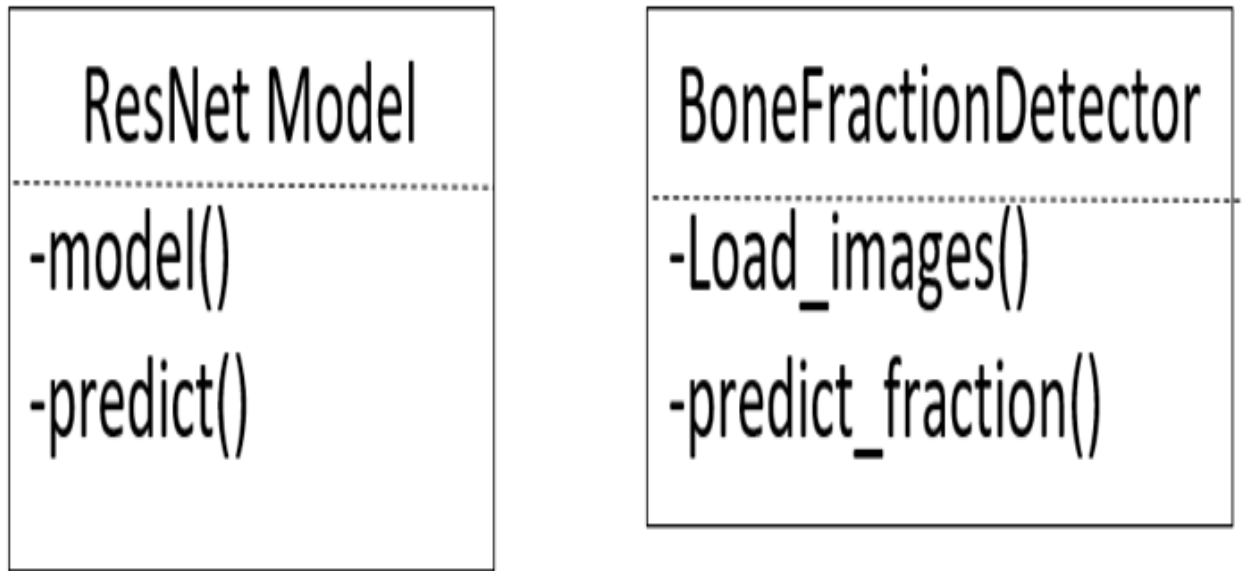


Figure 4.4: Class Diagram For Bone Fracture Detection

In figure 4.4 it shows A class diagram is a type of Unified Modeling Language(UML) diagram that illustrates the structure of a system by showing the classes, their attributes, methods, and relationships. In simpler terms, it's like a blueprint that describes the building blocks of a software system and how they interact. Each class represents a specific entity or object in the system, such as a person, vehicle, or transaction. Attributes describe the properties of each class, while methods represent the actions or behaviors that the class can perform. Relationships between classes show how they are connected or related to each other, such as inheritance, association, aggregation, or composition. Class diagrams provide a visual representation of the system's design, making it easier to understand and communicate the architecture and relationships between different components.

4.2.4 Sequence Diagram

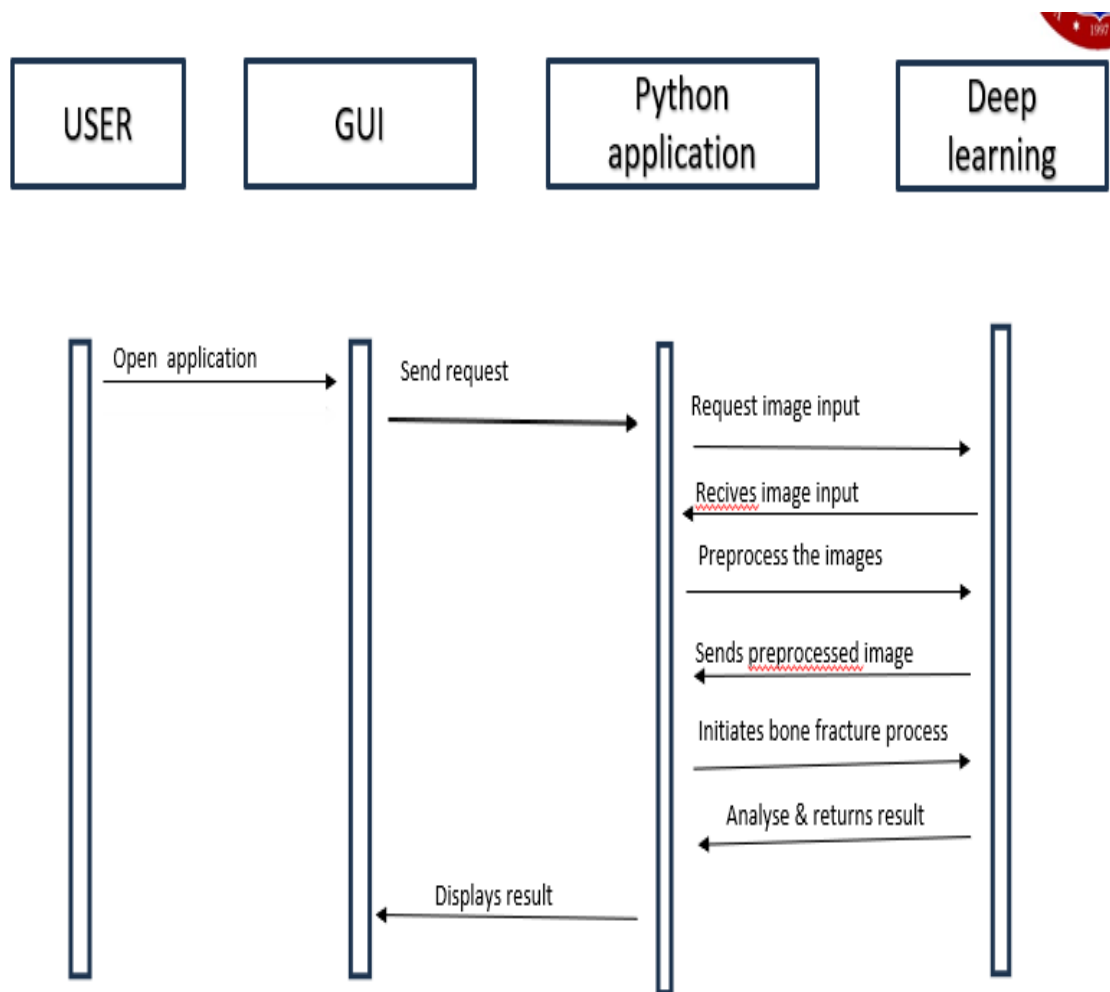


Figure 4.5: Sequence Diagram For Bone Fracture Detection

In figure 4.5 it shows A sequence diagram is another type of UML diagram that illustrates the interactions between objects or components in a system over time. It shows the sequence of messages exchanged between objects in a chronological order, depicting how they collaborate to accomplish a specific task or scenario. In simpler terms, it's like a step-by-step narrative that describes the flow of actions and communication between different parts of a system. Each object is represented by a vertical lifeline, and messages between objects are depicted as horizontal arrows, indicating the order and direction of communication.

4.2.5 Collaboration Diagram For Bone Fracture Detection

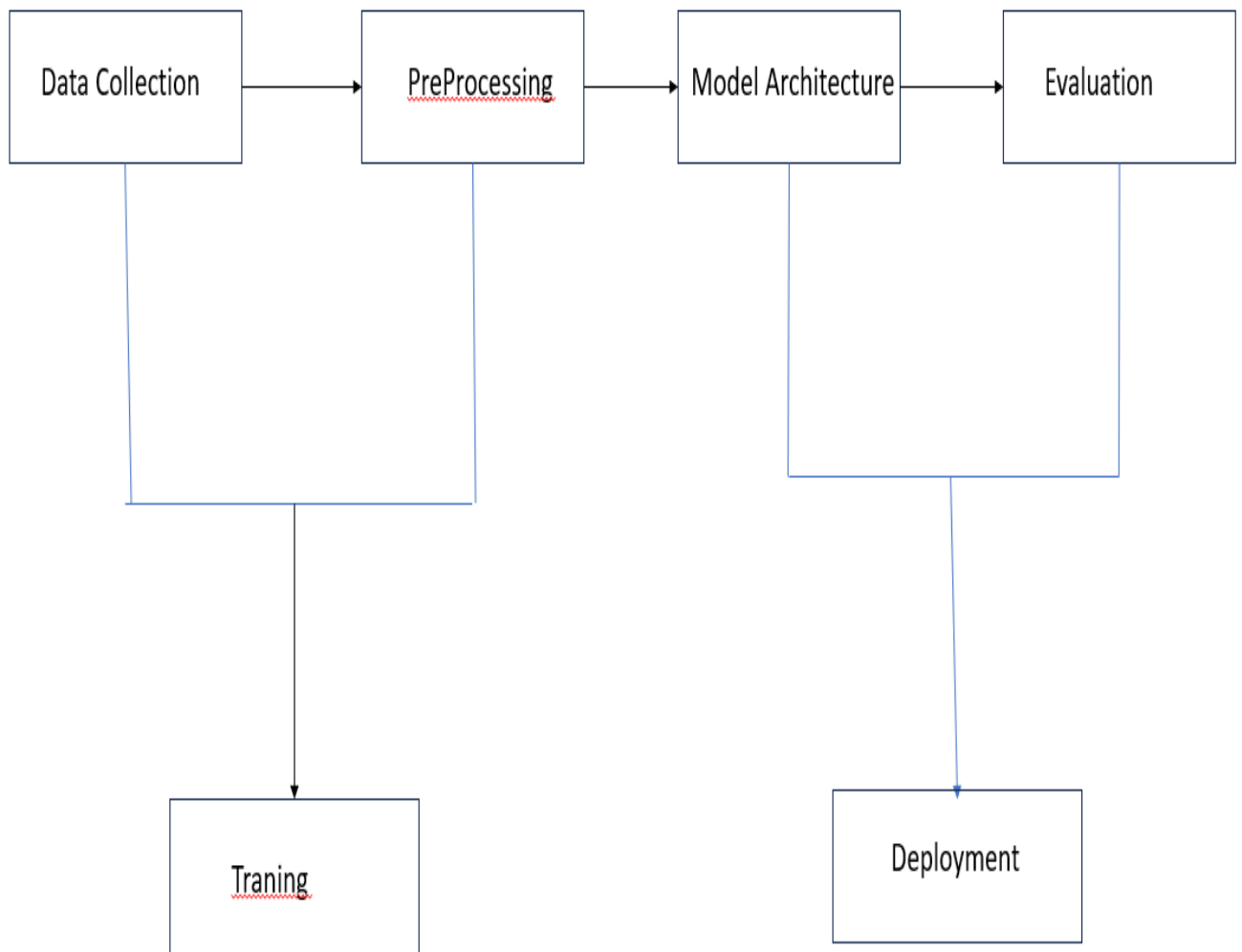


Figure 4.6: collaboration Diagram For Bone Fracture Detection

In figure 4.6 it shows A collaboration diagram, also known as a communication diagram, is a type of UML diagram that illustrates the interactions and relationships between objects or components in a system. It focuses on how objects collaborate to accomplish a specific task or scenario by showing the messages exchanged between them. In simpler terms, it's like a visual representation of a conversation between different parts of a system, highlighting the flow of communication and interaction. Each object is represented by a rectangular box, and arrows between objects indicate the messages or communications passed between them.

4.2.6 Activity Diagram

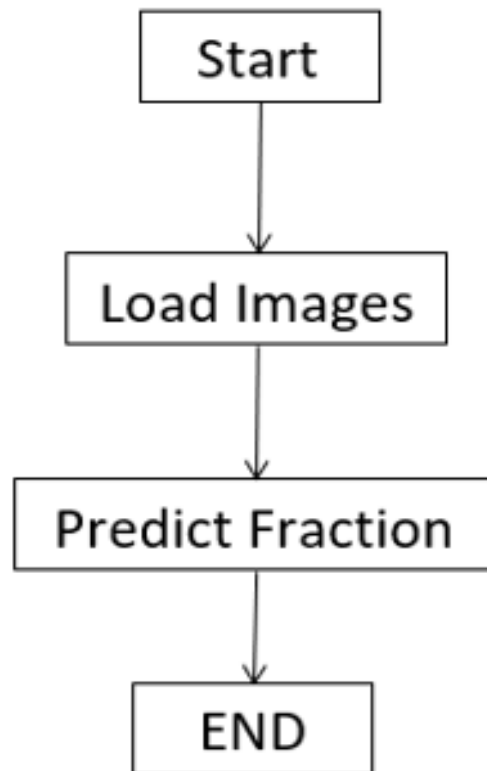


Figure 4.7: Activity Diagram For Bone Fracture Detection

In figure 4.6 it shows An activity diagram is another type of UML diagram that illustrates the flow of activities or actions within a system, typically representing business processes, workflows, or algorithms. It uses symbols such as rectangles (to represent activities), diamonds (to represent decisions or branching points), and arrows (to represent the flow of control) to visually depict the sequence of actions and decision points in a process.

4.3 Algorithm & Pseudo Code

4.3.1 Algorithm

Our data contains about 20,000 x-ray images, including three different types of bones - elbow, hand, and shoulder. After loading all the images into data frames and assigning a label to each image, we split our images into 72training, 18augmentation

and pre-processing the x-ray images, such as flip horizontal. The second step uses a ResNet50 neural network to classify the type of bone in the image. Once the bone type has been predicted, A specific model will be loaded for that bone type prediction from 3 different types that were each trained to identify a fracture in another bone type and used to detect whether the bone is fractured.

This approach utilizes the strong image classification capabilities of ResNet50 to identify the type of bone and then employs a specific model for each bone to determine if there is a fracture present. Utilizing this two-step process, the algorithm can efficiently and accurately analyze x-ray images, helping medical professionals diagnose patients quickly and accurately.

The algorithm can determine whether the prediction should be considered a positive result, indicating that a bone fracture is present, or a negative result, indicating that no bone fracture is present.

4.3.2 Pseudo Code

pesudo code for resnet 50:

```

1 # Define ResNet-50 architecture
2 class ResNet50:
3     def __init__(self, input_shape, num_classes):
4         # Define input layer
5         self.input_layer = Input(shape=input_shape)
6
7         # Initial convolutional layer
8         x = Conv2D(filters=64, kernel_size=(7, 7), strides=(2, 2), padding='same', activation='relu')(self.input_layer)
9         x = BatchNormalization()(x)
10        x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)
11
12        # Residual blocks
13        x = self._residual_block(x, filters=[64, 64, 256], strides=(1, 1), shortcut=True)
14        x = self._identity_block(x, filters=[64, 64, 256])
15        x = self._identity_block(x, filters=[64, 64, 256])
16
17        x = self._residual_block(x, filters=[128, 128, 512], strides=(2, 2), shortcut=True)
18        x = self._identity_block(x, filters=[128, 128, 512])
19        x = self._identity_block(x, filters=[128, 128, 512])
20        x = self._identity_block(x, filters=[128, 128, 512])
21
22        x = self._residual_block(x, filters=[256, 256, 1024], strides=(2, 2), shortcut=True)
23        x = self._identity_block(x, filters=[256, 256, 1024])
24        x = self._identity_block(x, filters=[256, 256, 1024])
25        x = self._identity_block(x, filters=[256, 256, 1024])

```

```

26     x = self._identity_block(x, filters=[256, 256, 1024])
27     x = self._identity_block(x, filters=[256, 256, 1024])
28
29     x = self._residual_block(x, filters=[512, 512, 2048], strides=(2, 2), shortcut=True)
30     x = self._identity_block(x, filters=[512, 512, 2048])
31     x = self._identity_block(x, filters=[512, 512, 2048])
32
33     # Global average pooling layer
34     x = GlobalAveragePooling2D()(x)
35
36     # Output layer
37     output_layer = Dense(num_classes, activation='softmax')(x)
38
39     # Define model
40     self.model = Model(inputs=self.input_layer, outputs=output_layer)
41
42 # Residual block function
43 def _residual_block(self, input_layer, filters, strides, shortcut=False):
44     x = Conv2D(filters=filters[0], kernel_size=(1, 1), strides=strides, padding='same',
45               activation='relu')(input_layer)
46     x = BatchNormalization()(x)
47
48     x = Conv2D(filters=filters[1], kernel_size=(3, 3), strides=(1, 1), padding='same',
49               activation='relu')(x)
50     x = BatchNormalization()(x)
51
52     x = Conv2D(filters=filters[2], kernel_size=(1, 1), strides=(1, 1), padding='same',
53               activation='relu')(x)
54     x = BatchNormalization()(x)
55
56     # Shortcut connection
57     if shortcut:
58         shortcut_connection = Conv2D(filters=filters[2], kernel_size=(1, 1), strides=strides,
59                                   padding='same')(input_layer)
60         shortcut_connection = BatchNormalization()(shortcut_connection)
61         x = Add()([x, shortcut_connection])
62     else:
63         x = Add()([x, input_layer])
64
65     x = Activation('relu')(x)
66     return x
67
68 # Identity block function
69 def _identity_block(self, input_layer, filters):
70     x = Conv2D(filters=filters[0], kernel_size=(1, 1), strides=(1, 1), padding='same',
71               activation='relu')(input_layer)
72     x = BatchNormalization()(x)
73
74     x = Conv2D(filters=filters[1], kernel_size=(3, 3), strides=(1, 1), padding='same',
75               activation='relu')(x)

```

```

70     x = BatchNormalization()(x)
71
72     x = Conv2D(filters=filters[2], kernel_size=(1, 1), strides=(1, 1), padding='same',
73               activation='relu')(x)
74     x = BatchNormalization()(x)
75
76     x = Add()([x, input_layer])
77     x = Activation('relu')(x)
78     return x
79
80 # Create ResNet-50 model
81 resnet50_model = ResNet50(input_shape=(224, 224, 3), num_classes=1000)

```

4.4 Module Description

RESNET IMPLEMENTATION

The methodology for building a bone fracture detection system using ResNet50 and Python GUI involves several steps:

4.4.1 Data collection and preprocessing:

A dataset of X-ray images with labeled fracture and non-fracture images will be collected. The images will be preprocessed, which may include resizing, normalization, and data augmentation.

4.4.2 Fine-tuning ResNet-50:

The ResNet-50 model will be pre-trained on a large dataset and then fine-tuned on the bone fracture dataset to improve its performance on this task and its architecture .

4.4.3 Training and evaluation:

The fine-tuned model will be trained and evaluated using a split of the bone fracture dataset into training, validation, and testing sets. The following are the detailed steps for building the bone fracture detection system.

4.4.4 Building GUI:

Choose a GUI library like PyQt or Tkinter. Build a GUI interface that allows users to upload an X-ray image, initiate the detection process, and display the results. Use a suitable image processing library like OpenCV or Pillow to preprocess the uploaded image and pass it through the fine-tuned ResNet-50 model. Display the results on the GUI interface, including the uploaded image, the classification result, and an alert message if a fracture is detected.

4.4.5 Model and Analysis:

Bone fracture detection using the ResNet-50 algorithm in Python involves several steps. These include:

Data collection and preparation: Collect a dataset of bone X-ray images with fractures and without fractures.

Preprocess the data by resizing the images to a fixed size and normalizing the pixel values.

4.4.6 Model development:

1. Load the pre-trained ResNet-50 model in Python.
2. Add a few layers on top of the pre-trained model for fine-tuning.
3. Freeze the pre-trained layers and train only the added layers on the bone X-ray dataset

4.4.7 Model evaluation:

Split the dataset into training and validation sets.

Train the model on the training set and evaluate its performance on the validation set.

Fine-tune the model based on the validation set performance.

4.5 Steps to execute/run/implement the project

4.5.1 Dataset Collection:

- The data set we used called MURA and included 3 different bone parts, MURA is a dataset of musculoskeletal radiographs and contains 20,335 images described below:

— **Part** —	**Normal** —	**Fractured** —	**Total** —
—	:	—	:
— **Elbow** —	3160 —	2236 —	5396 —
— **Hand** —	4330 —	1673 —	6003 —
— **Shoulder** —	4496 —	4440 —	8936 —

- The data is separated into train and valid where each folder contains a folder of a patient and for each patient between 1-3 images for the same bone part.

4.5.2 Install Packages:

- Download and install the python and install additional packages required like customtkinter,PyAutoGUI,PyGetWindow,Pillow,numpy,tensorflow,keras,pandas,matplotlib,scikit-learn,colorama

4.5.3 Code Implementation:

- Write the code for prediction and training data sets. and create prediction and training sets

4.5.4 Execution:

- Now open the command prompt and run the main file using the run command in python. So, that will get our result in prompt as well as in data base.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 input Design



Figure 5.1: X-ray Image

In figure 5.1 it shows the x-ray image of musculoskeletal bones as an input to detect whether it is fractured or not fractured.

5.1.2 Output Design

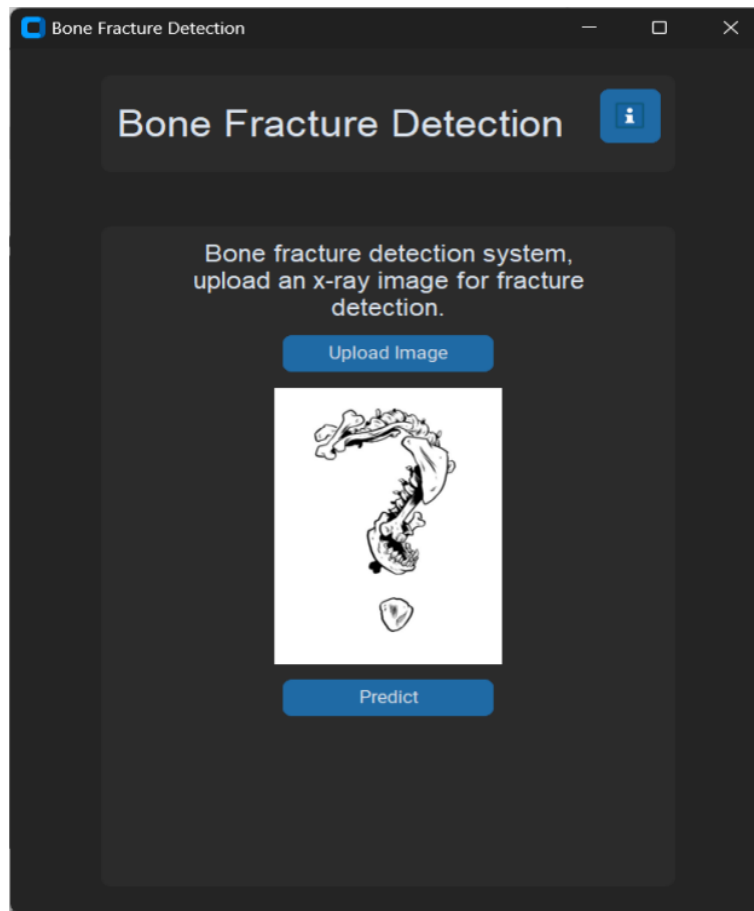


Figure 5.2: **Processing Image**

In Figure 5.2 it shows the GUI window when we run the code. we get this page to upload an image.

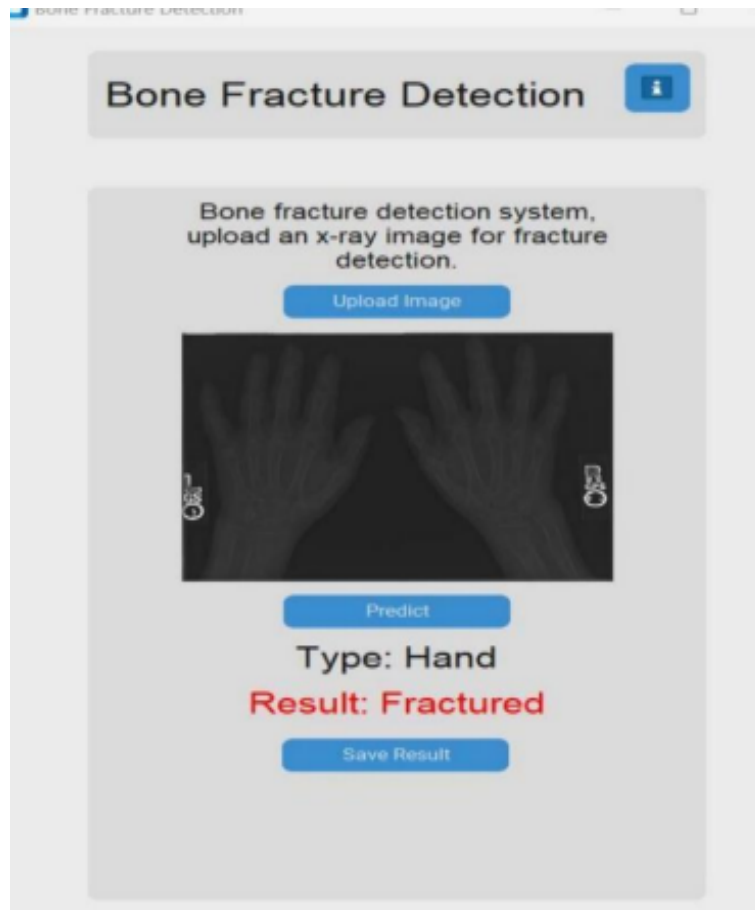


Figure 5.3: **Hand**

In figure 5.3 it shows A GUI page in which we upload a x-ray image it detects that x-ray image as a Type: Hand and Result: Fractured

5.2 Testing

The analysis of the bone fracture detection model involves evaluating its performance metrics such as accuracy, precision, recall, and F1-score. These metrics are calculated based on the true positive, true negative, false positive, and false negative predictions of the model.

The model's performance can be further analyzed using a confusion matrix, which shows the number of correct and incorrect predictions made by the model for each class (fracture and no fracture). From the confusion matrix, metrics such as sensitivity and specificity can be calculated, which provide additional insights into the model's performance.

It is important to note that the performance of the bone fracture detection model can be affected by factors such as the quality of the input images, the size of the dataset, and the choice of hyperparameters such as the learning rate and batch size. Therefore, it is essential to perform thorough analysis and experimentation to fine-tune the model for optimal performance.

5.3 Types of Testing

5.3.1 Unit Testing

Unit testing is a critical aspect of software development that involves testing individual units or components of code in isolation to ensure they function correctly. For a project like bone fracture detection using deep learning and a Python GUI, unit testing can be applied to various components such as data preprocessing, deep learning model architecture, GUI functionality, and overall system behavior.

Input

```
1 import unittest
2 import numpy as np
3 from your_module import preprocess_image
4
5 class TestPreprocessImage(unittest.TestCase):
6     def test_resize(self):
7         # Test image resizing
8         image = np.random.rand(100, 100, 3) # Create a random RGB image
9         resized_image = preprocess_image(image, target_size=(224, 224)) # Resize image
10        self.assertEqual(resized_image.shape, (224, 224, 3)) # Check if image is resized correctly
11
12    def test_normalization(self):
13        # Test image normalization
14        image = np.random.rand(224, 224, 3) # Create a random RGB image of size 224x224
15        normalized_image = preprocess_image(image, normalize=True) # Normalize image
16        self.assertAlmostEqual(np.mean(normalized_image), 0.0, places=5) # Check if mean is close
17        # to 0
18        self.assertAlmostEqual(np.std(normalized_image), 1.0, places=5) # Check if standard
19        # deviation is close to 1
20
21    def test_augmentation(self):
22        # Test image augmentation
23        image = np.random.rand(224, 224, 3) # Create a random RGB image of size 224x224
24        augmented_image = preprocess_image(image, augment=True) # Augment image
25        self.assertNotEqual(np.array_equal(image, augmented_image), True) # Check if image is
26        # augmented
```

```
24
25 if __name__ == '__main__':
26     unittest.main()
```

Test result

After reviewing the test results, you can identify any failing or erroneous test cases and debug the corresponding code to address the issues. Passing unit tests provide confidence that the software behaves correctly under various conditions and helps maintain the reliability and correctness of the codebase.

5.3.2 Integration Testing

Input

```
1 import unittest
2 from your_module import preprocess_image, predict_fracture
3 from your_gui_module import GUI
4
5 class TestIntegration(unittest.TestCase):
6     def setUp(self):
7         # Initialize test data and components
8         self.image_path = "test_image.jpg"
9         self.gui = GUI()
10
11     def test_image_processing_and_prediction(self):
12         # Test integration between image preprocessing and fracture prediction
13         image = preprocess_image(self.image_path) # Preprocess image
14         prediction = predict_fracture(image) # Predict fracture using preprocessed image
15         self.assertTrue(prediction in ["Fracture", "No Fracture"]) # Ensure prediction result is
            valid
16
17     def test_gui_functionality(self):
18         # Test integration between GUI and other system components
19         self.gui.load_image(self.image_path) # Simulate loading image through GUI
20         result = self.gui.run_fracture_detection() # Simulate running fracture detection
21         self.assertTrue(result in ["Fracture", "No Fracture"]) # Ensure GUI displays valid
            detection result
22
23 if __name__ == '__main__':
24     unittest.main()
```

Test result

By reviewing the integration test results, developers can gain insights into the interactions between different components of the system and identify any issues that need to be addressed. Passing integration tests provide confidence in the correctness and reliability of the integrated system, while failures or errors highlight areas that require attention and further testing.

5.3.3 System Testing

The system test for bone fracture detection using the ResNet-50 algorithm in Python involves evaluating the model's performance on a separate test dataset that the model has not seen during training or validation

To perform the system test, we can follow these steps

Load the trained model and the test dataset.

Preprocess the test images by resizing and normalizing them.

Use the trained model to predict the fracture status of each test image.

Input

```
1 from tensorflow.keras.models import load_model
2 from sklearn.metrics import confusion_matrix, accuracy_score,
3 precision_score, recall_score, f1_score
4 # Load the trained model
5 model = load_model('bone_fracture_detection_model.h5')
6 # Load the test dataset
7 test_data = ...
8 test_images = preprocess_images(test_data)
9 predictions = model.predict(test_images)
10 predicted_labels = np.argmax(predictions, axis=1)
11 actual_labels = test_data['fracture_status'].values
12 accuracy = accuracy_score(actual_labels, predicted_labels)
13 precision = precision_score(actual_labels, predicted_labels)
14 recall = recall_score(actual_labels, predicted_labels)
15 f1 = f1_score(actual_labels, predicted_labels)
16 confusion_mat = confusion_matrix(actual_labels, predicted_labels)
```

Test Result

By evaluating the model on a separate test dataset, we can obtain an accurate estimate of its performance in real-world scenarios. This ensures that the model is not overfitting to the training or validation dataset and can generalize well to new, unseen data

5.3.4 Test Result

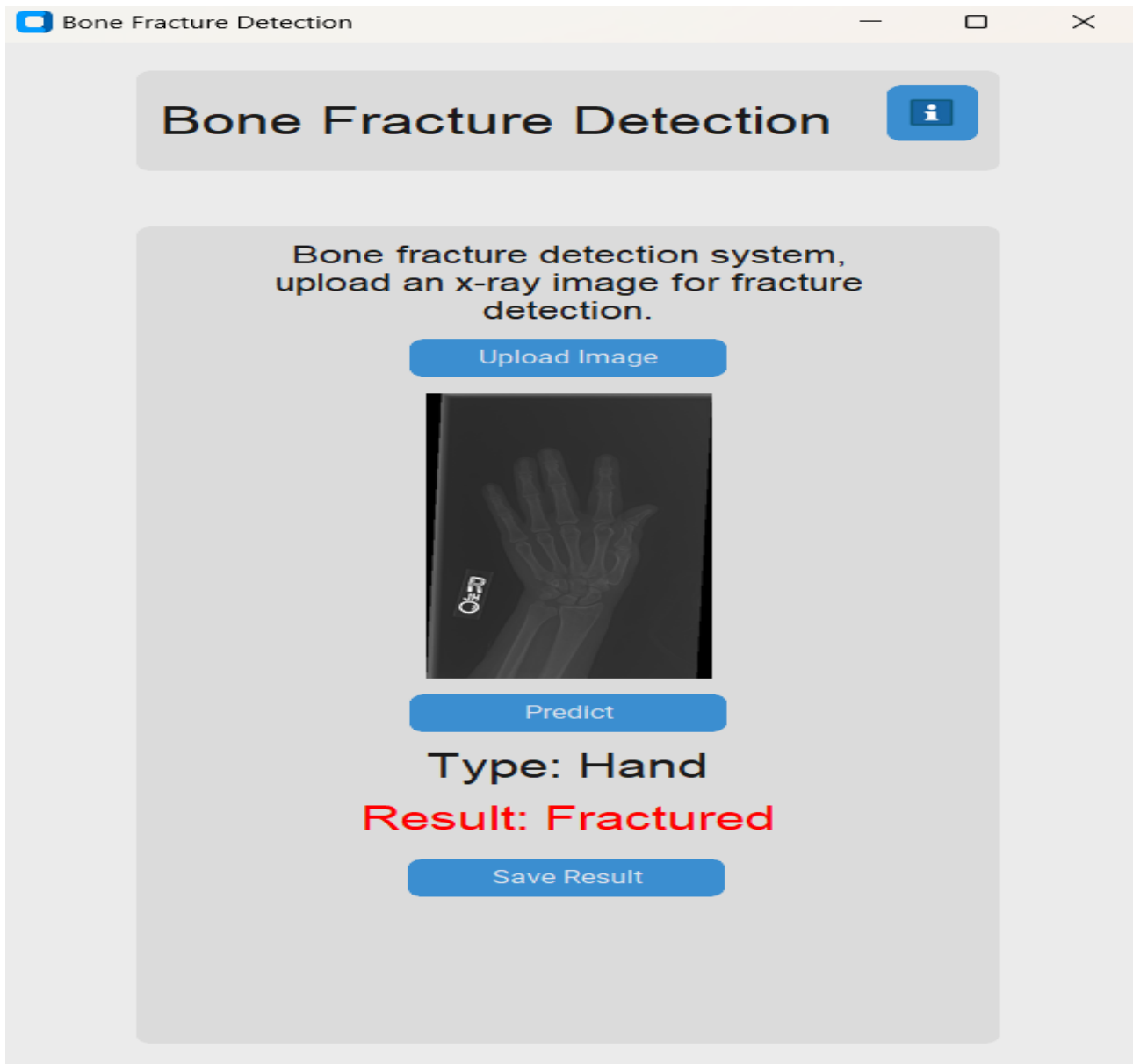


Figure 5.4: **Fractured Hand**

In figure 5.4 it shows A GUI page in which we upload a x-ray image and it detects that x-ray image as a Type: Hand and Result: Fractured

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The efficiency of a ResNet implementation is multifaceted, encompassing aspects such as training time, inference speed, resource utilization, and scalability. ResNet architectures, characterized by skip connections that alleviate the vanishing gradient problem, often lead to faster convergence during training. Furthermore, techniques like transfer learning capitalize on pre-trained models, reducing the need for extensive training data and computation. In terms of inference, ResNet architectures demonstrate efficiency, particularly when deployed on modern hardware with optimized deep learning libraries.

Techniques like model pruning and quantization further enhance inference speed without compromising accuracy. Efficient resource utilization is a hallmark of ResNet models, leveraging GPUs' parallel processing capabilities effectively. Additionally, scalability is achieved through techniques like distributed training, enabling ResNet models to handle large datasets and complex tasks across multiple GPUs or distributed computing environments. Overall, careful optimization of model architecture, training strategies, and hardware infrastructure ensures that ResNet implementations achieve state-of-the-art performance with minimal computational overhead.

6.2 Comparison of Existing and Proposed System

Existing system:(CNN)

In the existing system for bone fracture detection, the accuracy and efficiency of fracture detection rely heavily on the employed algorithms and techniques. Typically, these systems utilize traditional computer vision methods or shallow learning algorithms to analyze bone X-ray images for fractures. While these approaches may provide satisfactory results in some cases, they can be limited by their ability to capture complex patterns and variations in bone structures. Moreover, the scalabil-

ity of such systems may be constrained by the computational resources required for processing large datasets or handling sophisticated algorithms. Despite their potential drawbacks, existing systems may offer established performance and integration within clinical workflows, providing a foundation for comparison with proposed advancements.

Proposed system:(Resnet 50)

The proposed ResNet50 implementation offers a promising alternative for bone fracture detection, leveraging deep learning techniques to enhance accuracy and efficiency. ResNet50, a convolutional neural network architecture renowned for its depth and skip connections, has demonstrated exceptional performance in various image recognition tasks. By leveraging its ability to capture intricate features and patterns within bone X-ray images, the proposed ResNet50 model aims to achieve superior accuracy in fracture detection compared to traditional methods. Additionally, ResNet50's efficiency in training and inference, along with its scalability across diverse datasets and computational environments, positions it as a compelling solution for advancing bone fracture detection systems. However, the successful integration of the ResNet50 model within existing clinical workflows and its usability for healthcare professionals remain important considerations for its practical implementation and adoption.

6.3 Sample Code

```
1 import os
2 from tkinter import filedialog
3 import customtkinter as ctk
4 import pyautogui
5 import pygetwindow
6 from PIL import ImageTk, Image
7 from predictions import predict
8 # global variables
9 project_folder = os.path.dirname(os.path.abspath(__file__))
10 folder_path = project_folder + '/images/'
11 filename = ""
12 class App(ctk.CTk):
13     def __init__(self):
14         super().__init__()
15         self.title("Bone Fracture Detection")
16         self.geometry(f"{500}x{740}")
17         self.head_frame = ctk.CTkFrame(master=self)
18         self.head_frame.pack(pady=20, padx=60, fill="both", expand=True)
```

```

19 self.main_frame = ctk.CTkFrame(master=self)
20 25
21 self.main_frame.pack(pady=20, padx=60, fill="both", expand=True)
22 self.head_label = ctk.CTkLabel(master=self.head_frame, text="Bone Fracture
23 Detection",
24 font=(ctk.CTkFont("Roboto"), 28))
25 self.head_label.pack(pady=20, padx=10, anchor="nw", side="left")
26 img1 = ctk.CTkImage(Image.open(folder_path + "info.png"))
27 self.img_label = ctk.CTkButton(master=self.head_frame, text="", image=img1,
28 command=self.open_image_window,
29 width=40, height=40)
30 self.img_label.pack(pady=10, padx=10, anchor="nw", side="right")
31 self.info_label = ctk.CTkLabel(master=self.main_frame,
32 text="Bone fracture detection system, upload an x-ray image for fracture
33 detection.",
34 wraplength=300, font=(ctk.CTkFont("Roboto"), 18))
35 self.info_label.pack(pady=10, padx=10)
36 self.upload_btn = ctk.CTkButton(master=self.main_frame, text="Upload
37 Image", command=self.upload_image)
38 self.upload_btn.pack(pady=0, padx=1)
39 self.frame2 = ctk.CTkFrame(master=self.main_frame, fg_color="transparent",
40 width=256, height=256)
41 self.frame2.pack(pady=10, padx=1)
42 img = Image.open(folder_path + "Question-Mark.jpg")
43 img_resized = img.resize((int(256 / img.height * img.width), 256)) # new width
44 & height
45 img = ImageTk.PhotoImage(img_resized)
46 self.img_label = ctk.CTkLabel(master=self.frame2, text="", image=img)
47 self.img_label.pack(pady=1, padx=10)
48 self.predict_btn = ctk.CTkButton(master=self.main_frame, text="Predict",
49 26
50 command=self.predict_gui)
51 self.predict_btn.pack(pady=0, padx=1)
52 self.result_frame = ctk.CTkFrame(master=self.main_frame,
53 fg_color="transparent", width=200, height=100)
54 self.result_frame.pack(pady=5, padx=5)
55 self.loader_label = ctk.CTkLabel(master=self.main_frame, width=100,
56 height=100, text="")
57 self.loader_label.pack(pady=3, padx=3)
58 self.res1_label = ctk.CTkLabel(master=self.result_frame, text="")
59 self.res1_label.pack(pady=5, padx=20)
60 self.res2_label = ctk.CTkLabel(master=self.result_frame, text="")
61 self.res2_label.pack(pady=5, padx=20)
62 self.save_btn = ctk.CTkButton(master=self.result_frame, text="Save Result",
63 command=self.save_result)
64 self.save_label = ctk.CTkLabel(master=self.result_frame, text="")
65 def upload_image(self):
66 global filename
67 f_types = [("All Files", "*.")]
68 filename = filedialog.askopenfilename(filetypes=f_types,

```

```

69 initialdir=project_folder+'/test/Wrist/')
70 self.save_label.configure(text="")
71 self.res2_label.configure(text="")
72 self.res1_label.configure(text="")
73 self.img_label.configure(self.frame2, text="", image="")
74 img = Image.open(filename)
75 img_resized = img.resize((int(256 / img.height * img.width), 256)) # new width
76 & height
77 27
78 img = ImageTk.PhotoImage(img_resized)
79 self.img_label.configure(self.frame2, image=img, text="")
80 self.img_label.image = img
81 self.save_btn.pack_forget()
82 self.save_label.pack_forget()
83 def predict_gui(self):
84     global filename
85     bone_type_result = predict(filename)
86     result = predict(filename, bone_type_result)
87     print(result)
88     if result == 'fractured':
89         self.res2_label.configure(text_color="RED", text="Result: Fractured",
90         font=(ctk.CTkFont("Roboto"), 24))
91     else:
92         self.res2_label.configure(text_color="GREEN", text="Result: Normal",
93         font=(ctk.CTkFont("Roboto"), 24))
94     bone_type_result = predict(filename, "Parts")
95     self.res1_label.configure(text="Type: " + bone_type_result,
96     font=(ctk.CTkFont("Roboto"), 24))
97     print(bone_type_result)
98     self.save_btn.pack(pady=10, padx=1)
99     self.save_label.pack(pady=5, padx=20)
100 def save_result(self):
101     tempdir = filedialog.asksaveasfilename(parent=self, initialdir=project_folder +
102     '/PredictResults/',
103     title='Please select a directory and filename', defaultextension=".png")
104     screenshots_dir = tempdir
105
106 window = pygetwindow.getWindowsWithTitle('Bone Fracture Detection')[0]
107 left, top = window.topleft
108 right, bottom = window.bottomright
109 pyautogui.screenshot(screenshots_dir)
110 im = Image.open(screenshots_dir)
111 im = im.crop((left + 10, top + 35, right - 10, bottom - 10))
112 im.save(screenshots_dir)
113 self.save_label.configure(text_color="WHITE", text="Saved!",
114 font=(ctk.CTkFont("Roboto"), 16))
115 def open_image_window(self):
116 im = Image.open(folder_path + "rules.jpeg")
117 im = im.resize((700, 700))
118 im.show()

```



```
119 if __name__ == "__main__":  
120     app = App()  
121     app.mainloop()
```

Output

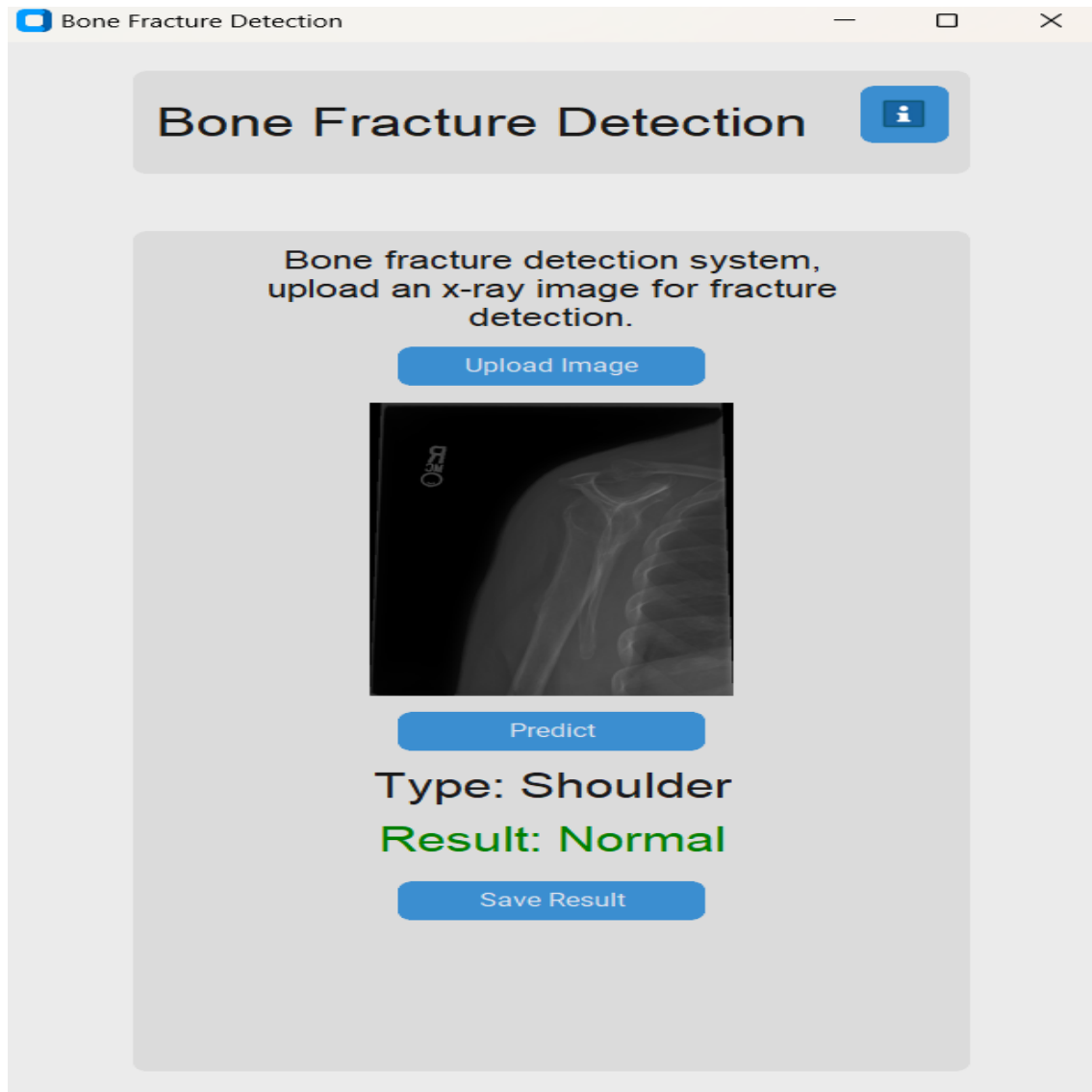


Figure 6.1: **Shoulder**

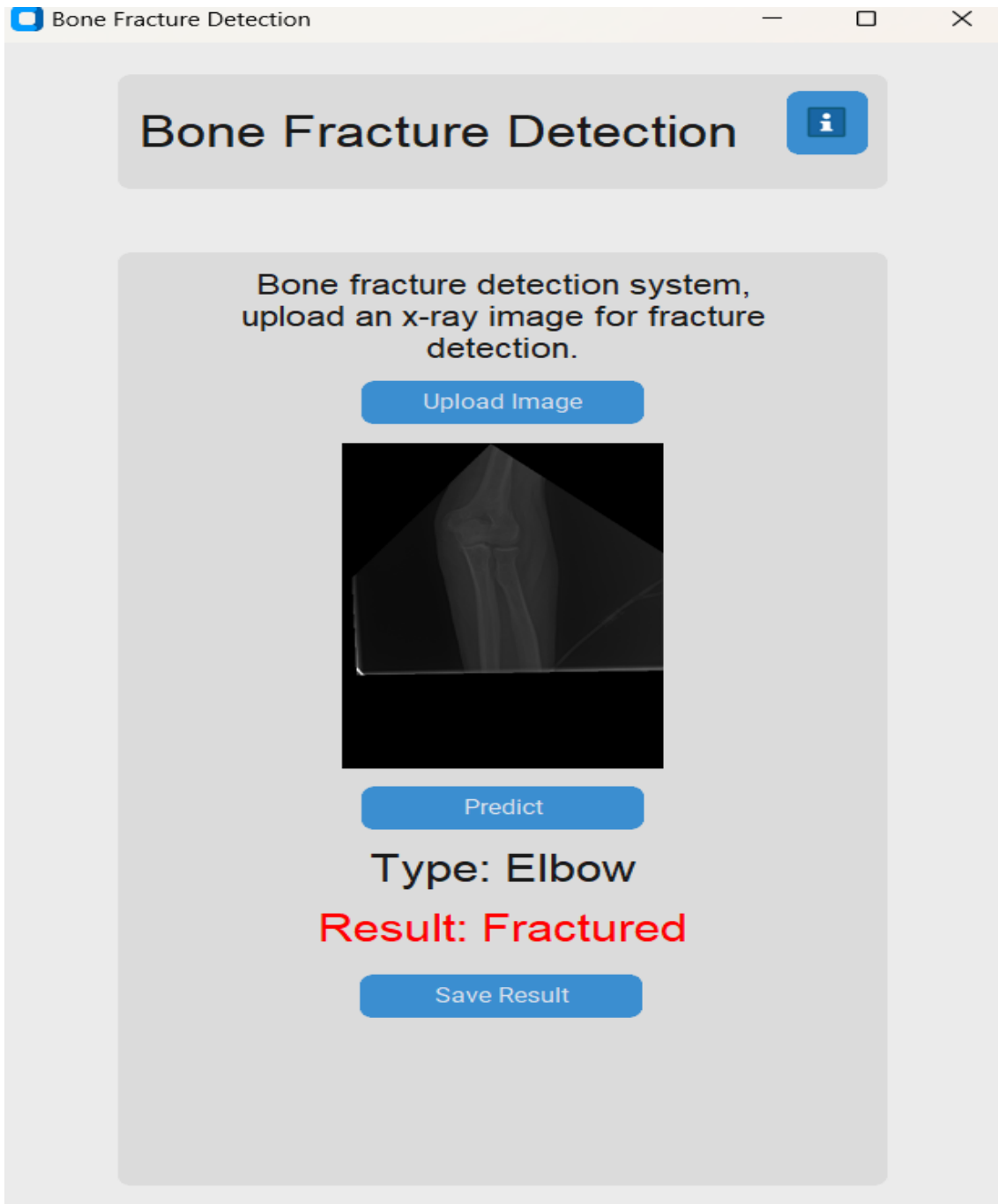


Figure 6.2: **Elbow**

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

Bone fracture detection using the ResNet-50 algorithm in Python is a promising approach that can help medical professionals diagnose fractures quickly and accurately. The ResNet-50 model, which is pre-trained on a large dataset of images, can be fine-tuned on a smaller dataset of bone X-ray images to detect fractures. Through this approach, we can develop an automated bone fracture detection system that can help medical professionals to make more accurate and efficient diagnoses. The system can also reduce the time and cost associated with manual diagnosis, which can be particularly beneficial in low-resource settings where access to medical professionals may be limited.

Overall, bone fracture detection using the ResNet-50 algorithm in Python has the potential to improve the accuracy and efficiency of fracture diagnosis, and further research in this area can lead to more sophisticated and effective medical imaging systems.

7.2 Future Enhancements

The project could advance by integrating advanced deep learning techniques beyond ResNet50, incorporating multi-model imaging data, enhancing transfer learning and domain adaptation methods, refining the user interface for better interactivity, integrating with clinical decision support systems, continually improving the model through real-world deployment and feedback and bolstering privacy and security measures. These enhancements aim to further elevate accuracy, efficiency, usability, and compliance with healthcare regulations, ultimately delivering more effective tools for diagnosing and treating bone fractures in clinical settings.

Chapter 8

PLAGIARISM REPORT

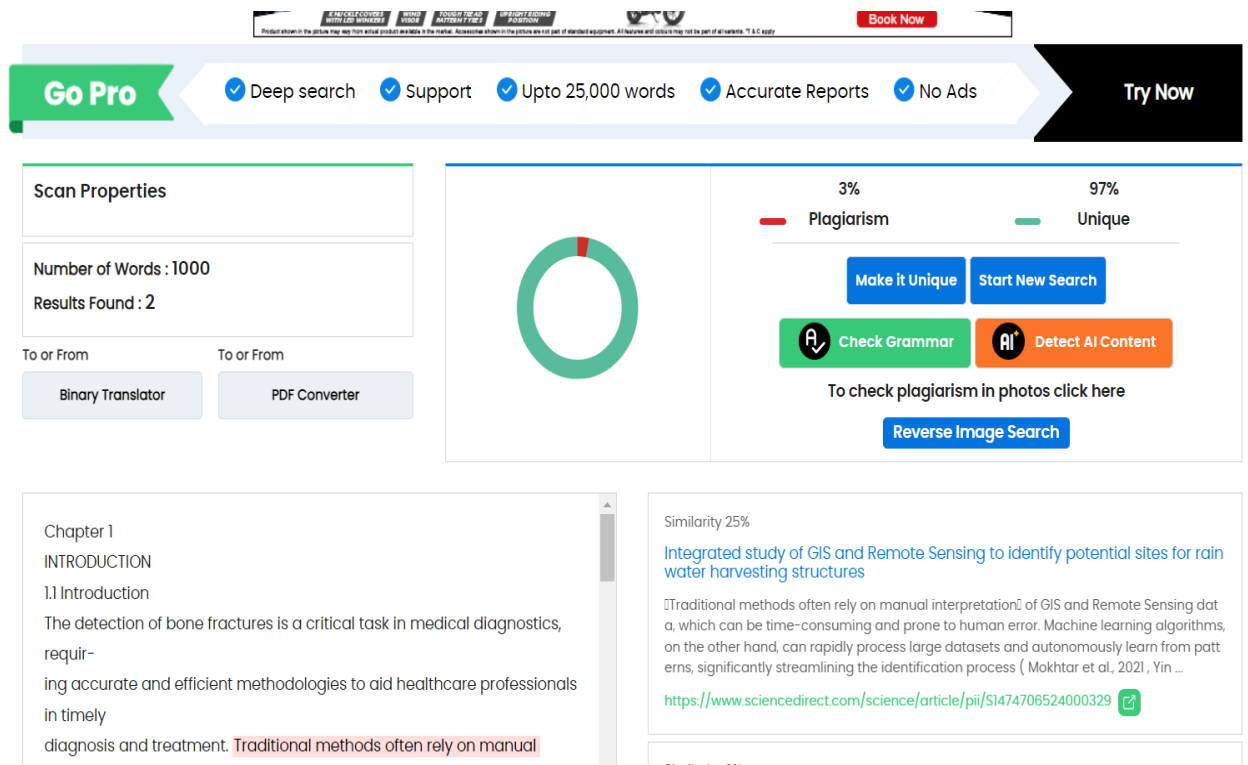


Figure 8.1: plagiarism Report

Chapter 9

SOURCE CODE

9.1 Source Code

```
1 import os
2 from tkinter import filedialog
3 import customtkinter as ctk
4 import pyautogui
5 import pygetwindow
6 from PIL import ImageTk, Image
7 from predictions import predict
8 # global variables
9 project_folder = os.path.dirname(os.path.abspath(__file__))
10 folder_path = project_folder + '/images/'
11 filename = ""
12 class App(ctk.CTk):
13     def __init__(self):
14         super().__init__()
15         self.title("Bone Fracture Detection")
16         self.geometry(f"{500}x{740}")
17         self.head_frame = ctk.CTkFrame(master=self)
18         self.head_frame.pack(pady=20, padx=60, fill="both", expand=True)
19         self.main_frame = ctk.CTkFrame(master=self)
20         25
21         self.main_frame.pack(pady=20, padx=60, fill="both", expand=True)
22         self.head_label = ctk.CTkLabel(master=self.head_frame, text="Bone Fracture
23         Detection",
24         font=(ctk.CTkFont("Roboto"), 28))
25         self.head_label.pack(pady=20, padx=10, anchor="nw", side="left")
26         img1 = ctk.CTkImage(Image.open(folder_path + "info.png"))
27         self.img_label = ctk.CTkButton(master=self.head_frame, text="", image=img1,
28         command=self.open_image_window,
29         width=40, height=40)
30         self.img_label.pack(pady=10, padx=10, anchor="nw", side="right")
31         self.info_label = ctk.CTkLabel(master=self.main_frame,
32         text="Bone fracture detection system, upload an x-ray image for fracture
33         detection.",
34         tempdir = filedialog.asksaveasfilename(parent=self, initialdir=project_folder +
35         '/PredictResults/',
36         title='Please select a directory and filename', defaultextension=".png")
37         screenshots_dir = tempdir
38
```

```

39 window = pygetwindow.getWindowsWithTitle('Bone Fracture Detection')[0]
40 left, top = window.topleft
41 right, bottom = window.bottomright
42 pyautogui.screenshot(screenshots_dir)
43 im = Image.open(screenshots_dir)
44 im = im.crop((left + 10, top + 35, right - 10, bottom - 10))
45 im.save(screenshots_dir)
46 self.save_label.configure(text_color="WHITE", text="Saved!",
47 font=(ctk.CTkFont("Roboto"), 16))
48 def open_image_window(self):
49 im = Image.open(folder_path + "rules.jpeg")
50 im = im.resize((700, 700))
51 im.show()
52 if __name__ == "__main__":
53 app = App()
54 Prediction_test.py
55 import os
56 from colorama import Fore
57 from predictions import predict
58 # load images to predict from paths
59 # .... / elbow1.jpg
60 # Hand fractured -- elbow2.png
61 # / / \ .....
62 # test - Elbow -----
63 # \ \ / elbow1.png
64 29
65 # Shoulder normal -- elbow2.jpg
66 # .... \
67 #
68 def load_path(path):
69 dataset = []
70 for body in os.listdir(path):
71 body_part = body
72 path_p = path + '/' + str(body)
73 for lab in os.listdir(path_p):
74 label = lab
75 path_l = path_p + '/' + str(lab)
76 for img in os.listdir(path_l):
77 img_path = path_l + '/' + str(img)
78 dataset.append(
79 {
80 'body_part': body_part,
81 'label': label,
82 'image_path': img_path,
83 'image_name': img
84 }
85 )
86 return dataset
87 categories_parts = ["Elbow", "Hand", "Shoulder"]
88 categories_fracture = ['fractured', 'normal']

```

```

89 def reportPredict(dataset):
90     30
91     total_count = 0
92     part_count = 0
93     status_count = 0
94     print(Fore.YELLOW +
95     '{0: <28}'.format('Name') +
96     '{0: <14}'.format('Part') +
97     '{0: <20}'.format('Predicted Part') +
98     '{0: <20}'.format('Status') +
99     '{0: <20}'.format('Predicted Status'))
100    for img in dataset:
101        body_part_predict = predict(img['image_path'])
102        fracture_predict = predict(img['image_path'], body_part_predict)
103        if img['body_part'] == body_part_predict:
104            part_count = part_count + 1
105            if img['label'] == fracture_predict:
106                status_count = status_count + 1
107            color = Fore.GREEN
108        else:
109            color = Fore.RED
110        print(color +
111        '{0: <28}'.format(img['image_name']) +
112        '{0: <14}'.format(img['body_part']) +
113        '{0: <20}'.format(body_part_predict) +
114        '{0: <20}'.format((img['label'])) +
115        '{0: <20}'.format(fracture_predict))
116    31
117    print(Fore.BLUE + '\npart acc: ' + str("%.2f" % (part_count / len(dataset) *
118    100)) + '%')
119    print(Fore.BLUE + 'status acc: ' + str("%.2f" % (status_count / len(dataset)
120    * 100)) + '%')
121    return
122    THIS_FOLDER = os.path.dirname(os.path.abspath(__file__))
123    test_dir = THIS_FOLDER + '/test/'
124    reportPredict(load_path(test_dir))
125    4.1.3 Prediction.py
126    import numpy as np
127    import tensorflow as tf
128    from keras.preprocessing import image
129    # load the models when import "predictions.py"
130    model_elbow_frac =
131    tf.keras.models.load_model("weights/ResNet50-Elbow_frac.h5")
132    model_hand_frac =
133    tf.keras.models.load_model("weights/ResNet50-Hand_frac.h5")
134    model_shoulder_frac =
135    tf.keras.models.load_model("weights/ResNet50-Shoulder_frac.h5")
136    model_parts =
137    tf.keras.models.load_model("weights/ResNet50-BodyParts.h5")
138    # categories for each result by index

```

```

139 # 0-Elbow 1-Hand 2-Shoulder
140 categories_parts = ["Elbow", "Hand", "Shoulder"]
141 # 0-fractured 1-normal
142 categories_fracture = ['fractured', 'normal']
143 # get image and model name, the default model is "Parts"
144 # Parts - bone type predict model of 3 classes
145 32
146 # otherwise - fracture predict for each part
147 def predict(img, model="Parts"):
148     size = 224
149     if model == 'Parts':
150         chosen_model = model_parts
151     else:
152         if model == 'Elbow':
153             chosen_model = model_elbow_frac
154         elif model == 'Hand':
155             chosen_model = model_hand_frac
156         elif model == 'Shoulder':
157             chosen_model = model_shoulder_frac
158     # load image with 224px224p (the training model image size, rgb)
159     temp_img = image.load_img(img, target_size=(size, size))
160     x = image.img_to_array(temp_img)
161     x = np.expand_dims(x, axis=0)
162     images = np.vstack([x])
163     prediction = np.argmax(chosen_model.predict(images), axis=1)
164     # chose the category and get the string prediction
165     if model == 'Parts':
166         prediction_str = categories_parts[prediction.item()]
167     else:
168         prediction_str = categories_fracture[prediction.item()]
169     return prediction_str
170 4.1.4 Training set.py
171 import numpy as np
172 33
173 import pandas as pd
174 import os.path
175 import matplotlib.pyplot as plt
176 from sklearn.model_selection import train_test_split
177 import tensorflow as tf
178 from tensorflow.keras.optimizers import Adam
179 # load images to build and train the model
180 # .... / img1.jpg
181 # test Hand patient0000 positive -- img2.png
182 # / / \ .....
183 # Dataset - Elbow ----- patient0001
184 # \ train \ / img1.png
185 # Shoulder patient0002 negative -- img2.jpg
186 # .... \
187 #
188 def load_path(path, part):

```



```

189 """
190 load X-ray dataset
191 """
192 dataset = []
193 for folder in os.listdir(path):
194     folder = path + '/' + str(folder)
195     if os.path.isdir(folder):
196         for body in os.listdir(folder):
197             if body == part:
198                 34
199                 body_part = body
200                 path_p = folder + '/' + str(body)
201                 for id_p in os.listdir(path_p):
202                     patient_id = id_p
203                     path_id = path_p + '/' + str(id_p)
204                     for lab in os.listdir(path_id):
205                         if lab.split('_')[-1] == 'positive':
206                             label = 'fractured'
207                         elif lab.split('_')[-1] == 'negative':
208                             label = 'normal'
209                     path_l = path_id + '/' + str(lab)
210                     for img in os.listdir(path_l):
211                         img_path = path_l + '/' + str(img)
212                     dataset.append(
213                         {
214                             'body_part': body_part,
215                             'patient_id': patient_id,
216                             'label': label,
217                             'image_path': img_path
218                         }
219                     )
220     return dataset
221 # this function get part and know what kind of part to train , save model and
222 save plots
223 def trainPart(part):
224     # categories = ['fractured', 'normal']
225     THIS_FOLDER = os.path.dirname(os.path.abspath(__file__))
226     35
227     image_dir = THIS_FOLDER + '/Dataset/'
228     data = load_path(image_dir, part)
229     labels = []
230     filepaths = []
231     # add labels for dataframe for each category 0-fractured , 1- normal
232     for row in data:
233         labels.append(row['label'])
234         filepaths.append(row['image_path'])
235     filepaths = pd.Series(filepaths, name='Filepath').astype(str)
236     labels = pd.Series(labels, name='Label')
237     images = pd.concat([filepaths, labels], axis=1)
238     # split all dataset 10% test , 90% train (after that the 90% train will split to

```

```

239 20% validation and 80% train
240 train_df, test_df = train_test_split(images, train_size=0.9, shuffle=True,
241 random_state=1)
242 # each generator to process and convert the filepaths into image arrays,
243 # and the labels into one-hot encoded labels.
244 # The resulting generators can then be used to train and evaluate a deep
245 learning model.
246 # now we have 10% test, 72% training and 18% validation
247 train_generator =
248 tf.keras.preprocessing.image.ImageDataGenerator(horizontal_flip=True,
249 preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
250 validation_split=0.2)
251 # use ResNet50 architecture
252 test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
253 preprocessing_function=tf.keras.applications.resnet50.preprocess_input)
254 36
255 train_images = train_generator.flow_from_dataframe(
256 dataframe=train_df,
257 x_col='Filepath',
258 y_col='Label',
259 target_size=(224, 224),
260 color_mode='rgb',
261 class_mode='categorical',
262 batch_size=64,
263 shuffle=True,
264 seed=42,
265 subset='training'
266 )
267 val_images = train_generator.flow_from_dataframe(
268 dataframe=train_df,
269 x_col='Filepath',
270 y_col='Label',
271 target_size=(224, 224),
272 color_mode='rgb',
273 class_mode='categorical',
274 batch_size=64,
275 shuffle=True,
276 seed=42,
277 subset='validation'
278 )
279 test_images = test_generator.flow_from_dataframe(
280 dataframe=test_df,
281 37
282 x_col='File path',
283 y_col='Label',
284 target_size=(224, 224),
285 color_mode='rgb',
286 class_mode='categorical',
287 batch_size=32,
288 shuffle=False)




```

```

289 pretrained_model = tf.keras.applications.resnet50.ResNet50(
290     input_shape=(224, 224, 3),
291     include_top=False,
292     weights='imagenet',
293     pooling='avg')
294 pretrained_model.trainable = False
295 inputs = pretrained_model.input
296 x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
297 x = tf.keras.layers.Dense(50, activation='relu')(x)
298 outputs = tf.keras.layers.Dense(2, activation='softmax')(x)
299 model = tf.keras.Model(inputs, outputs)
300 print("-----Training " + part + "-----")
301 model.compile(optimizer=Adam(learning_rate=0.0001),
302     loss='categorical_crossentropy', metrics=['accuracy'])
303 callbacks = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3,
304     restore_best_weights=True)
305 history = model.fit(train_images, validation_data=val_images, epochs=25,
306     callbacks=[callbacks])
307 model.save(THIS_FOLDER + "/weights/ResNet50_" + part + "_frac.h5")
308 results = model.evaluate(test_images, verbose=0)
309 print(part + " Results:")
310 print(results)
311 print(f"Test Accuracy: {np.round(results[1] * 100, 2)}%")
312 plt.plot(history.history['accuracy'])
313 plt.plot(history.history['val_accuracy'])
314 plt.title('model accuracy')
315 plt.ylabel('accuracy')
316 plt.xlabel('epoch')
317 plt.legend(['train', 'test'], loc='upper left')
318 figAcc = plt.gcf()
319 my_file = os.path.join(THIS_FOLDER, "./plots/FractureDetection/" + part +
320     "_Accuracy.jpeg")
321 figAcc.savefig(my_file)
322 plt.clf()
323 plt.plot(history.history['loss'])
324 plt.plot(history.history['val_loss'])
325 plt.title('model loss')
326 plt.ylabel('loss')
327 plt.xlabel('epoch')
328 plt.legend(['train', 'test'], loc='upper left')
329 figAcc = plt.gcf()
330 my_file = os.path.join(THIS_FOLDER, "./plots/FractureDetection/" + part +
331     "_Loss.jpeg")
332 figAcc.savefig(my_file)
333 plt.clf()
334 categories_parts = ["Elbow", "Hand", "Shoulder"]
335 for category in categories_parts:
336     trainPart(category)

```

9.2 Poster Presentation



BONE FRACTURE DETECTION USING DEEP LEARNING

Department of Computer Science and Engineering
School of Computing
1156CS701-MAJOR PROJECT
INHOUSE
WINTER SEMESTER 2023-2024

Batch: (2020-2024)

ABSTRACT

Bone fracture detection plays a crucial role in medical diagnosis, treatment planning, and patient care. Traditional methods rely heavily on manual interpretation of medical images, which can be time-consuming and prone to human error. In recent years, deep learning techniques have shown promise in automating the process of fracture detection from X-ray images. In this study, we propose a novel deep learning framework for bone fracture detection, leveraging convolutional neural networks (CNNs) and transfer learning to achieve accurate and efficient detection. Our model is trained on a large dataset of annotated X-ray images, allowing it to learn discriminative features for fracture detection. Through extensive experimentation and evaluation on diverse datasets, we demonstrate the effectiveness and robustness of our approach in accurately identifying bone fractures. Our method holds great potential for enhancing clinical workflows, facilitating early diagnosis, and improving patient outcomes in orthopedic care.

TEAM MEMBER DETAILS

<vtu.15349/T.ANAND SAI>
<vtu.17584/K.VINAY KUMAR>
<vtu.17571/SK.JAVEED>
<Phone no: 9390630474>
<Phone no: 8688807365>
<Phone no: 9963220616>
<vtu15349@veltech.edu.in>
<vtu17584@veltech.edu.in>
<vtu17571@veltech.edu.in>

INTRODUCTION

In the medical sector, finding bone fractures is crucial since early diagnosis and treatment of fractures can lead to better patient outcomes. Fractures in X-ray pictures have been successfully identified by deep learning models like ResNet-50. In this project, we will construct a user interface using Python GUI tools like Tkinter to identify bone fractures in X-ray images using the ResNet-50 model. Users can upload an X-ray image to the bone fracture detection system to start the detection process. The objective of this project is to develop an easy-to-use tool that enables medical practitioners to rapidly and reliably identify bone fractures in X-ray pictures. This technology may be utilised in clinics, hospitals, and other healthcare facilities to increase the efficiency and precision of bone fracture diagnostics. We'll give step-by-step instructions for building this system with Python and ResNet-50.

METHODOLOGIES

The methodology involves developing a bone fracture detection system using ResNet50, a deep learning architecture, implemented with Python GUI. The process begins with data collection, including bone X-ray images for training and testing. Preprocessing techniques such as resizing and normalization are applied to prepare the data. Next, the ResNet50 model is trained on the preprocessed images using transfer learning, fine-tuning it to detect fractures. A Python GUI is developed to provide an intuitive interface for users to upload X-ray images and visualize the detection results. Finally, the system is evaluated using performance metrics such as accuracy, sensitivity, and specificity, with iterative improvements made based on feedback and testing.

RESULTS

The results of the bone fracture detection project demonstrate the effectiveness of the ResNet50 model and Python GUI in accurately detecting fractures from X-ray images. The trained model achieves a high level of accuracy, sensitivity, and specificity in identifying fractures, validating its utility as a diagnostic tool in clinical settings. Additionally, the Python GUI provides a user-friendly interface for healthcare professionals to upload images, visualize detection results, and access diagnostic reports. Real-world testing and feedback from users further confirm the system's practicality and usability. Overall, the project's outcomes underscore its potential to improve fracture diagnosis efficiency and support better patient care in healthcare facilities.





FIGURE 1. output1

Figure 2. output2

STANDARDS AND POLICIES

PYTHON
Python is an open source programming language that was made to be easy-to-read and powerful. A Dutch programmer named Guido van Rossum made Python in 1991. He named it after the television show Monty Python's Flying Circus. Many Python examples and tutorials include jokes from the show. Because Python code is interpreted and not compiled into native machine instructions, code written for one platform will work on any other platform that has the Python interpreter installed.

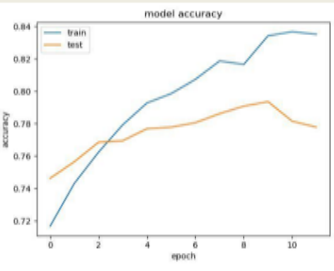


Figure 1. Model Accuracy

CONCLUSIONS

However, it is important to note that the performance of the bone fracture detection model depends on several factors such as the quality of the input images, the size of the dataset, and the choice of hyperparameters. Therefore, careful analysis and experimentation are required to fine-tune the model for optimal performance. Overall, bone fracture detection using the ResNet-50 algorithm in Python has the potential to improve the accuracy and efficiency of fracture diagnosis, and further research in this area can lead to more sophisticated and effective medical imaging systems.

ACKNOWLEDGEMENT

1. Project Supervisor : Mr. R. Anto Pravin, Assistant professor
2. Contact No : 9025831864
3. Project supervisor Mail ID : antopravin@veltech.edu.in

Figure 9.1: poster presentation

References

- [1] Kesia.C , Carlos. A,(2023), “Validation of a Compact Microwave Imaging System for Bone Fracture Detection”, IEEE Journal Research Article, vol .11, pp. 63690 – 63700.
- [2] Farah Mohammed et al.,(2023),“Block-Deep: A Hybrid Secure Data Storage and Diagnosis Model for Bone Fracture Identification of Athlete From X-Ray and MRI Images”, IEEE Journal Research Article,vol .11, pp. 14260 – 14270.
- [3] ao, L., Huang, Y., He, L., Lin, Y,(2023), ”Lesion-Guided Adaptive Graph Network for Bone Abnormality Detection From Musculoskeletal Radiograph” ,IEEE Journal of Research Article, vol. 12, pp. 26710-26718.
- [4] Gwiseong Moon et al.,(2022), “Computer Aided Facial Bone Fracture Diagnosis (CA-FBFD) System Based on Object Detection Model”, IEEE Journal of medicine , Vol .11, pp. 79061 – 79070.
- [5] Kesia.C , Carlos. A,(2022), “Feasibility of Bone Fracture Detection using microwave imaging ”, IEEE Open Journal of Antennas and Propagation, vol .3, pp. 836 – 847.
- [6] Serafeim mustakidis et al.,(2021),”Deep Learning for Bone Metastasis Localisation in Nuclear Imaging data of Breast Cancer Patients”, International Conference on Information, Intelligence, Systems and Applications, pp. 545-548.
- [7] Ashok.B et al.,(2020), “Enhanced Computerized Bone Fracture Detection Using Harris Corner Detection”, International Conference on Smart Electronics and Communication, pp. 572-576.

- [8] B. Johnson et al.,(2020), "Application of Residual Networks in Medical Image Analysis," IEEE Transactions on Medical Imaging, vol. 36, no. 4, pp. 1025-1032.
- [9] Smith, A., et al.,(2020), "Deep Learning for Medical Image Analysis: A Review." Journal of Medical Imaging, vol. 17, 245-263.
- [10] N. E. Jacob and M. Wyawahare,(2020),"Survey of bone fracture detection techniques,"International journal of Computer Applications, vol. 71, no. 17, pp. 31-34.
- [11] S. Singh, R. Singh, and G. Singh,(2019), "Automated Bone Fracture Detection and Classification in X-ray Images Using Convolutional Neural Networks", IEEE Research Article, pp. 345-350.
- [12] R. Alomari, M. A. Abutayeh, and M. H. Alshennawy,(2019)," Bone fracture detection using deep learning techniques", IEEE conference paper , pp. 257-260.
- [13] Pranusha.S et al.,(2019), "Bone Fracture Detection System using CNN Algorithm", International Conference on Intelligent Computing and Control Systems, pp. 545-549.
- [14] D. H. Kim and T. MacKinnon,(2018), "Artificial Intelligence in Fracture Detection:Transfer Learning from Deep Convolutional Neural Networks," Clinical Radiology, vol. 5, pp. 439-445.
- [15] Rathor, S., Jadon, R. S.,(2021) ," The art of domain classification and recognition for text conversation using support vector classifier", International Journal of Arts and Technology, vol. 11, 309-324.