International Research Journal of Education and Technology
Peer Reviewed Journal
ISSN 2581-7795

DETECTION OF HAND BONE FRACTURE IN X-RAY IMAGES USING HYBRID YOLO

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING – CYBER SECURITY

Submitted by

J.KRISHNA	(21WJ1A6219)
J.SAI	(21WJ1A6220)
K.POOJITHA	(21WJ1A6226)

Under the Esteemed guidance of

Mr. Md. Saleem
(Assistant Professor in Computer Science Engineering – Cyber Security)

Department of Computer Science Engineering – Cyber Security

School of Engineering
Guru Nanak Institutions Technical Campus
Ibrahimpatnam, Hyderabad, R.R. District – 501506

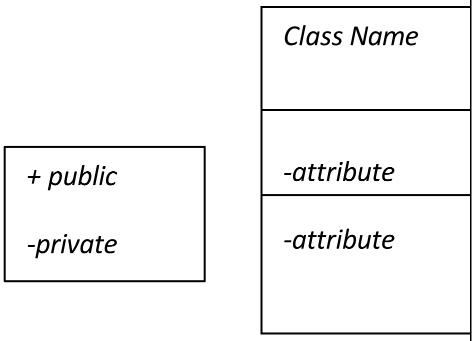
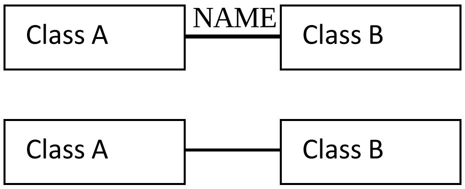
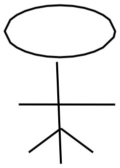
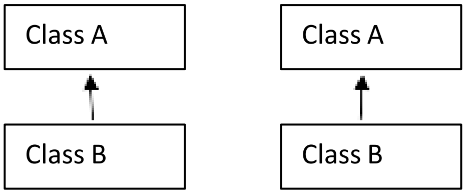
June - 2025

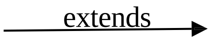
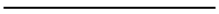
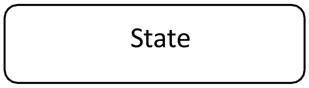
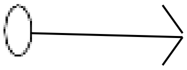
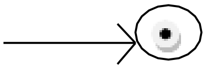
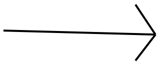
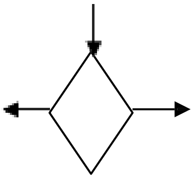
ABSTRACT

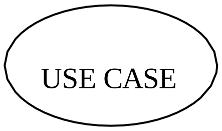
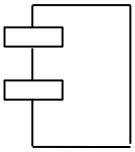
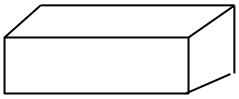
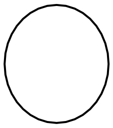
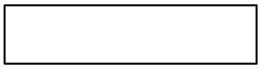
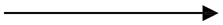
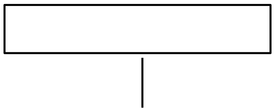
In the detection of bone fractures from X-ray images, accurate and timely diagnosis is crucial to prevent further complications and ensure proper healing. Existing models, such as **YOLO NAS** (You Only Look Once - Neural Architecture Search), have shown potential in object detection but have limitations in detecting subtle bone fractures, particularly small or hairline fractures. To address these shortcomings, the proposed model utilizes **YOLO V8**, an advanced version of the YOLO framework, which builds on previous models by offering improved accuracy, speed, and efficiency in real-time object detection tasks. YOLO V8 enhances the detection capabilities by refining the architecture and optimizing performance, making it better suited for medical image analysis. The model is trained on a comprehensive dataset of **1200 hand-bone X-ray images**, classified into six distinct fracture categories. A comparison of the YOLO V8 model with YOLO NAS highlights the improved ability of V8 to detect complex and subtle fractures, ensuring faster and more reliable diagnoses. This advancement is essential for clinical settings, where delays or misdiagnoses could lead to severe outcomes for patients.

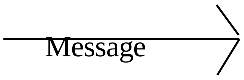
LIST OF FIGURES

LIST OF SYMBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class	 Class Name -attribute -attribute + public -private	Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation		Interaction between the system and external environment

5.	Relation (uses)	Uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint

13.	Use case		Interact ion between the system and external environment.
14.	Component		Represents physical modules which are a collection of components.
15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.

20.	Message		Represents the message exchanged.

CHAPTER-1

INTRODUCTION

1.1 GENERAL

This project aims to improve the detection of bone fractures, particularly small or hairline fractures, in X-ray images using advanced object detection techniques. Accurate and timely diagnosis of fractures is crucial to prevent further complications and ensure proper healing. While existing models like YOLO NAS (You Only Look Once - Neural Architecture Search) have demonstrated potential in object detection tasks, they struggle with identifying subtle fractures in medical images. To address this limitation, the project leverages YOLO V8, a more refined version of the YOLO framework, known for its enhanced accuracy, speed, and efficiency. YOLO V8 improves upon previous models by optimizing its architecture and performance, making it more suitable for medical image analysis. The model is trained on a diverse dataset of 1200 hand-bone X-ray images, categorized into six distinct fracture types. A comparative analysis between YOLO V8 and YOLO NAS highlights the superior ability of YOLO V8 to detect complex and subtle fractures, providing faster, more reliable diagnoses. This advancement is crucial for clinical settings, where accurate and quick diagnoses are essential to prevent adverse patient outcomes.

1.2 SCOPE OF THE PROJECT

The scope of this project is focused on enhancing the detection of bone fractures, particularly subtle or hairline fractures, in X-ray images using YOLO V8, a state-of-the-art object detection model. The primary goal is to develop a model that can accurately and efficiently identify fractures in hand-bone X-rays, ensuring faster and more reliable diagnoses in clinical environments. The project involves training YOLO V8 on a comprehensive dataset of 1200 X-ray images, categorized into six distinct fracture types. It also includes comparing YOLO V8's performance with previous models like YOLO NAS to demonstrate the improvements in detecting complex fractures. The scope extends to optimizing the model for real-time applications, ensuring that it can be integrated into clinical workflows for immediate diagnostic support. By advancing the capabilities of object detection in medical imaging, the project aims to reduce diagnostic delays and errors, ultimately contributing to better patient outcomes and more effective treatment planning.

1.3 OBJECTIVE

The objective of this project is to develop a highly accurate and efficient system for detecting bone fractures in X-ray images using the advanced YOLO V8 object detection model. The primary goal is to improve the detection of subtle fractures, such as hairline fractures, that are challenging for traditional models to identify. The project aims to optimize YOLO V8 for real-time diagnosis, enabling fast and reliable fracture detection in clinical environments. The model will be trained on a comprehensive dataset of 1200 hand-bone X-ray images, categorized into six distinct fracture types, ensuring the model is robust and generalizable. A key objective is to compare the performance of YOLO V8 with previous models like YOLO NAS to demonstrate its enhanced accuracy in detecting complex fractures. Additionally, the project focuses on integrating the developed system into clinical workflows, assisting radiologists in providing timely and accurate diagnoses. By reducing diagnostic errors and improving the efficiency of fracture detection, the project seeks to contribute to better patient outcomes and optimized treatment planning.

1.4 EXISTING SYSTEM

- The existing abstract describes the use of advanced technologies for detecting hand bone fractures through X-ray images using a hybrid model. The model combines deep learning algorithms, including **YOLO NAS** (You Only Look Once - Neural Architecture Search), **EfficientDet**, and **DETR3** (DEtection TRansformer), all of which are known for their excellent object detection capabilities. YOLO NAS optimizes the neural network architecture for better performance, while EfficientDet focuses on computational efficiency without compromising accuracy, enabling fast detection of fractures. DETR3, an improved version of the DETR model, uses transformer-based techniques to enhance object detection, making it suitable for complex medical image analysis.

- The model is trained on a hybrid dataset of 4,736 hand-bone X-ray images, classified into six different fracture types, and is evaluated by comparing its performance against existing algorithms.

1.4.1 EXISTINGSYSTEM DISADVANTAGES

- Suboptimal accuracy
- Slow inference
- Limited precision
- Missed fractures.

CHAPTER-2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

[1] TITLE: Detection of Hand Bone Fractures in X-ray Images using Hybrid YOLO NAS

AUTHORS: Sai Charan Medaramatla, Chennupati Veda Samhitha, Sagar Dhanraj Pande, Surendra reddy Vinta

YEAR: 2024

DESCRIPTION: The majority of bones that have fractured in humans are hand bones. As we use our hands widely, they need early and accurate detection to be diagnosed. Fractures in the hands are most frequently brought on by blunt force trauma, sports injuries, and bone fragility. Getting an X-ray of the affected area of the bone and then discussing the results with a medical practitioner or radiologist is the standard procedure for determining whether or not a fracture exists in the bone. The majority of medical professionals and radiologists use X-rays to diagnose hand fractures; however, in some instances, they might miss small or hairline fractures. Additionally, it might be difficult to find a good radiologist who can detect the fracture properly and in time, because a delay in diagnosis can cause the injury to be more severe, and the bone might not be recovered properly. Therefore, to detect hand bone and joint fractures through X-rays, a hybrid model was developed that uses deep learning algorithms YOLO NAS, Efficient Det, and DETR3, which are widely recognized for their exact object detection capabilities. The dataset used for this model is a hybrid dataset of 4736 hand-bone X-ray images, they were further classified into 6 classes based on their types. To evaluate the performance the best method is to compare the proposed model with the existing models, hence, the model was compared with various existing algorithms and result analysis was done.

[2] **TITLE:** Fracture Detection in Wrist X-ray Images Using Deep Learning-Based Object Detection Models

AUTHORS: Fırat Hardalaç , Fatih Uysal , Ozan Peker , Murat Çiçeklidağ , Tolga Tolunay , Nil Tokgöz , Uğurhan Kutbay , Boran Demirciler 5, Fatih Mert

YEAR: 2022

DESCRIPTION: Hospitals, especially their emergency services, receive a high number of wrist fracture cases. For correct diagnosis and proper treatment of these, images obtained from various medical equipment must be viewed by physicians, along with the patient's medical records and physical examination. The aim of this study is to perform fracture detection by use of deep-learning on wrist X-ray images to support physicians in the diagnosis of these fractures, particularly in the emergency services. Using SABL, RegNet, RetinaNet, PAA, Libra R-CNN, FSAF, Faster R-CNN, Dynamic R-CNN and DCN deep-learning-based object detection models with various backbones, 20 different fracture detection procedures were performed on Gazi University Hospital's dataset of wrist X-ray images. To further improve these procedures, five different ensemble models were developed and then used to reform an ensemble model to develop a unique detection model, 'wrist fracture detection-combo (WFD-C)'. From 26 different models for fracture detection, the highest detection result obtained was 0.8639 average precision (AP50) in the WFD-C model. Huawei Turkey R&D Center supports this study within the scope of the ongoing cooperation project coded 071813 between Gazi University, Huawei and Medskor.

[3] **TITLE:** Deep Learning Approach for Arm Fracture Detection Based on an Improved YOLOv8 Algorithm

AUTHORS: Gerardo Meza ,Deepak Ganta , andSergio Gonzalez Torres

YEAR: 2024

DESCRIPTION: Artificial intelligence (AI)-assisted computer vision is an evolving field in medical imaging. However, accuracy and precision suffer when using the existing AI models for small, easy-to-miss objects such as bone fractures, which affects the models' applicability and effectiveness in a clinical setting. The proposed integration of the Hybrid-Attention (HA) mechanism into the YOLOv8 architecture offers a robust solution to improve accuracy, reliability, and speed in medical imaging applications. Experimental results demonstrate that our HA-modified YOLOv8 models achieve a 20% higher Mean Average Precision (mAP 50) and improved processing speed in arm fracture detection.

[4] **TITLE:** Skeletal Fracture Detection with Deep Learning: A Comprehensive Review

AUTHORS: Zhihao , Afzan Adam , Mohammad Faidzul Nasrudin , Masri Ayob , Gauthamen Punganan

YEAR: 2023

DESCRIPTION: Deep learning models have shown great promise in diagnosing skeletal fractures from X-ray images. However, challenges remain that hinder progress in this field. Firstly, a lack of clear definitions for recognition, classification, detection, and localization tasks hampers the consistent development and comparison of methodologies. The existing reviews often lack technical depth or have limited scope. Additionally, the absence of explainable facilities undermines the clinical application and expert confidence in results. To address these issues, this comprehensive review analyzes and evaluates 40 out of 337 recent papers identified in prestigious databases, including WOS, Scopus, and EI. The objectives of this review are threefold. Firstly, precise definitions are established for the bone fracture recognition, classification, detection, and localization tasks within deep learning. Secondly, each study is summarized based on key aspects such as the bones involved, research objectives, dataset sizes, methods employed, results obtained, and concluding remarks. This process distills the diverse approaches into a generalized processing framework or workflow. Moreover, this review identifies the crucial areas for future research in deep learning models for bone fracture diagnosis. These include enhancing the network interpretability, integrating multimodal clinical information, providing therapeutic schedule recommendations, and developing advanced visualization methods for clinical application. By addressing these challenges, deep learning models can be made more intelligent and specialized in this domain. In conclusion, this review fills the gap in precise task definitions within deep learning for bone fracture diagnosis and provides a comprehensive analysis of the recent research. The findings serve as a foundation for future advancements, enabling improved interpretability, multimodal integration, clinical decision support, and advanced visualization techniques.

CHAPTER-3

DESIGN AND DEVELOPMENT

3.1 PROBLEM STATEMENT

Traditional object detection models like YOLO NAS struggle to accurately detect small and complex bone fractures in X-ray images, leading to potential misdiagnoses and delayed treatment. YOLO V8 enhances real-time medical image analysis by improving accuracy, speed, and efficiency, and is trained on a dataset of 1200 hand-bone X-ray images categorized into six distinct fracture types, making it more effective in detecting subtle fractures. The improved performance of YOLO V8 ensures faster and more reliable diagnoses, reducing the risks associated with misinterpretation and delays in medical intervention, ultimately improving patient outcomes.

3.2 SYSTEM ARCHITECTURE

3.2.1 PROPOSED SYSTEM

In the context of the project described in the abstract, the reference to **V8** likely pertains to a version or iteration of the proposed model, such as a refined or enhanced model architecture. While the exact details of **V8** are not explicitly provided in the abstract, it is reasonable to assume that **V8** refers to a more advanced version (Version 8) of the hybrid model that combines deep learning algorithms for detecting hand fractures. This version likely incorporates improvements or optimizations based on the feedback from earlier versions of the model, such as enhancements in accuracy, efficiency, and the ability to detect small or hairline fractures more effectively. It could also involve a more sophisticated use of the **YOLO NAS**, **EfficientDet**, and **DETR3** algorithms, or even the integration of additional techniques such as data augmentation, model fine-tuning, or new preprocessing steps to improve detection performance in medical imaging.

3.2.2 PROPOSED SYSTEM ADVANTAGES:

- YOLO V8's refined architecture enables more accurate detection of subtle and complex fractures in X-ray images.
- YOLO V8 processes images more efficiently, reducing detection time without compromising performance.
- The advanced model design ensures more precise bounding box predictions, minimizing false positives and false negatives.
- YOLO V8 generalizes well across diverse datasets, improving its performance on unseen fracture types and varying image conditions.

➢ 3.2.3 SYSTEM ARCHITECTURE

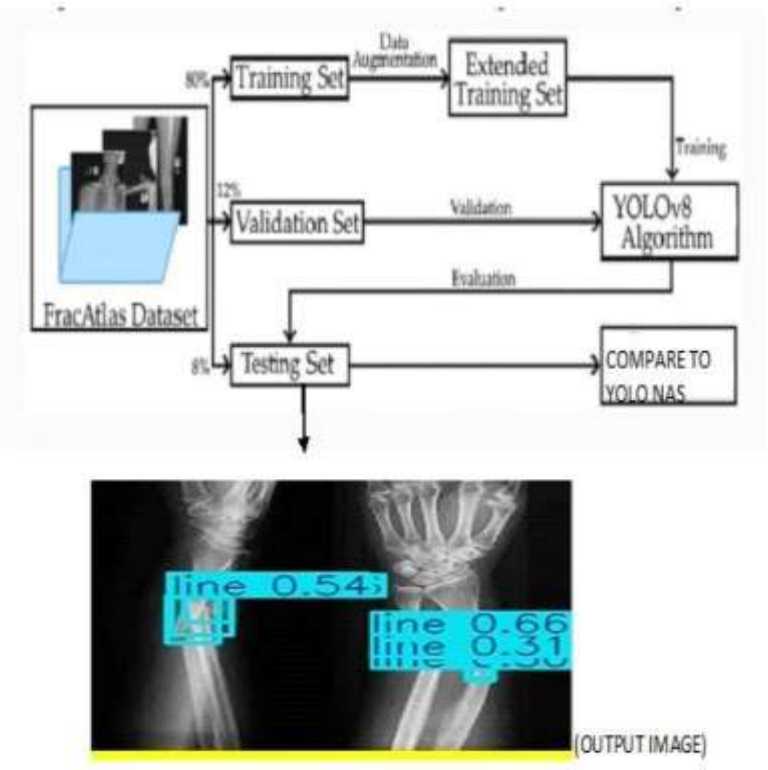


Fig 4.11 System Architecture

3.3 DESIGN METHODOLOGY

3.3.1 MODULES

1. Data Collection

Collect a dataset consisting of 1200 hand-bone X-ray images. These images should be annotated with fracture details, including fracture types like hairline fractures, displaced fractures, and compound fractures.

2. Preprocessing

Resize each X-ray image to a fixed dimension (e.g., 416x416 or 608x608 pixels) to match the input size required by YOLO V8.Normalize pixel values to the range [0, 1] by dividing the pixel values by 255.Apply data augmentation techniques like rotation, zoom, flipping, and cropping to improve the model's robustness and reduce overfitting..

3. Model Selection

YOLO V8 due to its high accuracy, speed, and real-time detection capability, making it suitable for medical image analysis.Integrate YOLO V8 with Convolutional Neural Networks (CNNs) and attention mechanisms to detect subtle and small fractures in X-ray images.

4. Model Training

Train the YOLO V8 model using the annotated X-ray dataset. The model learns to predict the bounding boxes, fracture types, and confidence scores in a single forward pass.Optimize hyperparameters such as learning rate, batch size, and number of epochs for optimal performance.Utilize pre-trained weights from similar tasks to enhance training speed and improve model performance.

5. Model Evaluation

Evaluate the model's performance using metrics like accuracy, precision, recall, F1-score, and Intersection over Union (IoU).Use K-fold cross-validation to assess the model’s robustness and ability to generalize across unseen data.

6. Model Optimization

Based on evaluation results, fine-tune the model by adjusting hyperparameters and training configurations to improve detection accuracy.Compare YOLO V8's performance with YOLO NAS to demonstrate improved fracture detection capabilities, especially for subtle and small fractures.

7. Prediction or Detection

For each X-ray image, the model predicts the location (bounding boxes), class labels (fracture types), and confidence scores.Apply Non-Maximum Suppression (NMS) to remove redundant bounding boxes and retain only the most accurate detections.

8. Deployment and Integration

Deploy the trained model in a real-time detection system, accessible through a web interface for medical professionals.Integrate the model into clinical workflows to assist radiologists in diagnosing fractures efficiently, offering real-time predictions that can help inform treatment decisions.

3.3.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system do and not how it should be implemented.

• PROCESSOR	:	DUAL CORE 2 DUOS.
• RAM	:	4GB DD RAM
• HARD DISK	:	250 GB

3.3.3 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

• Operating System	:	Windows 7/8/10
• Platform	:	Spyder3
• Programming Language	:	Python
• Front End	:	Spyder3

3.3.4 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behaviour, Firstly, the system is the first that achieves the standard notion of semantic security for data confidentiality in attribute-based deduplication systems by resorting to the hybrid cloud architecture.

3.3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

Usability

The system is designed with completely automated process hence there is no or less user intervention.

Reliability

The system is more reliable because of the qualities that are inherited from the chosen platform python. The code built by using python is more reliable.

Performance

This system is developing in the high-level languages and using the advanced back-end technologies it will give response to the end user on client system with in very less time.

Supportability

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is built into the system.

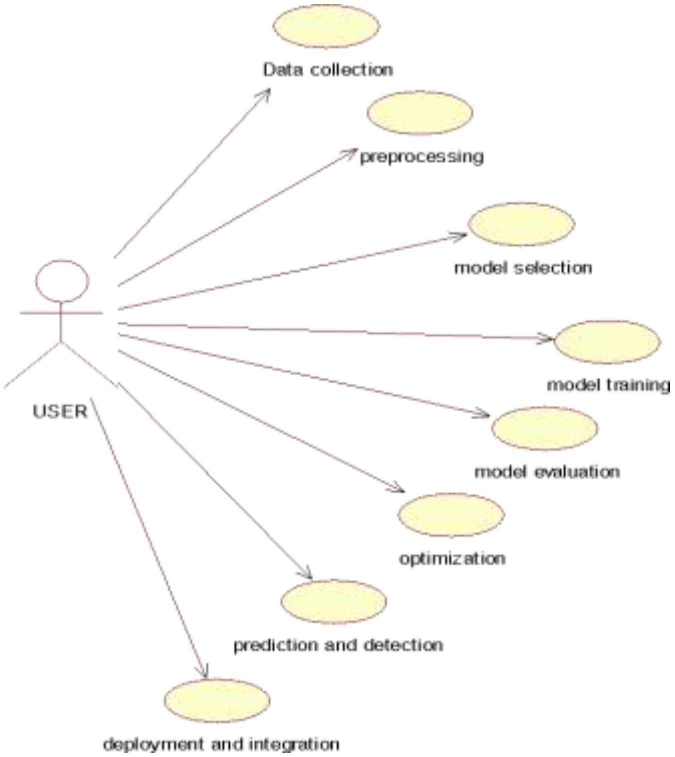
Implementation

The system is implemented in web environment using Jupyter notebook software. The server is used as the intelligence server and windows 10 professional is used as the platform. Interface the user interface is based on Jupyter notebook provides server system.

3.4 COMPONENTS DESIGN

UML DIAGRAMS

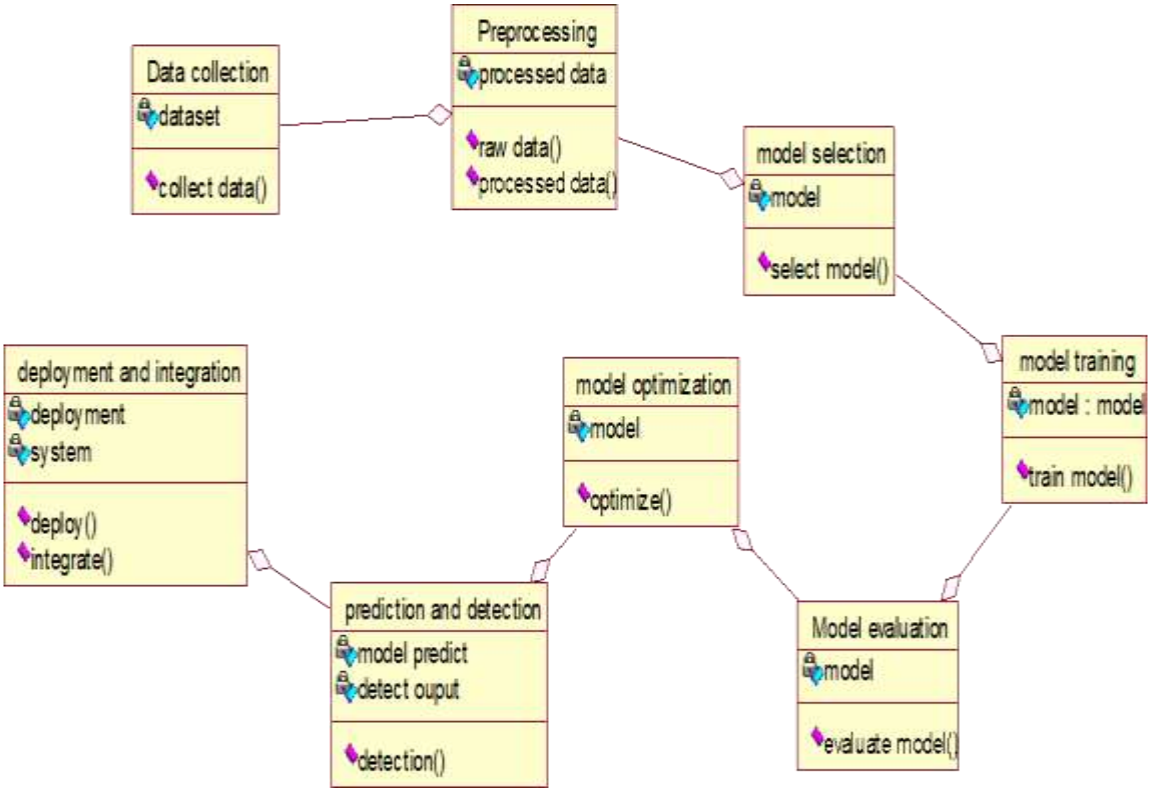
3.4.1 USE CASE DIAGRAM



EXPLANATION:

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. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

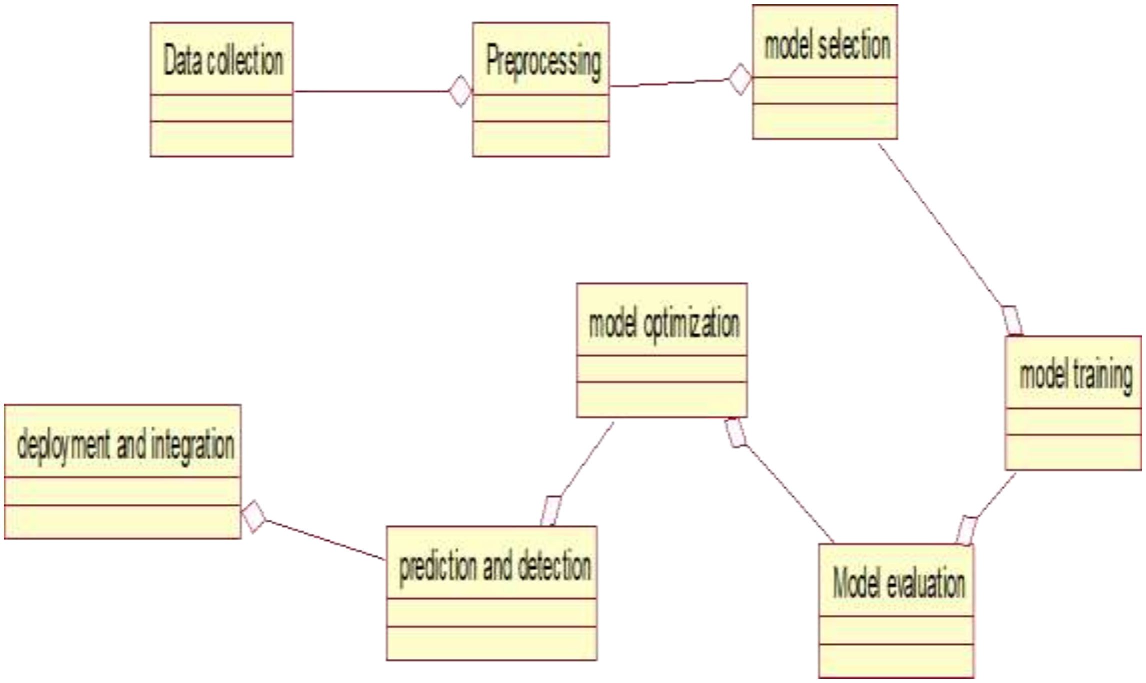
3.4 CLASS DIAGRAM



EXPLANATION

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

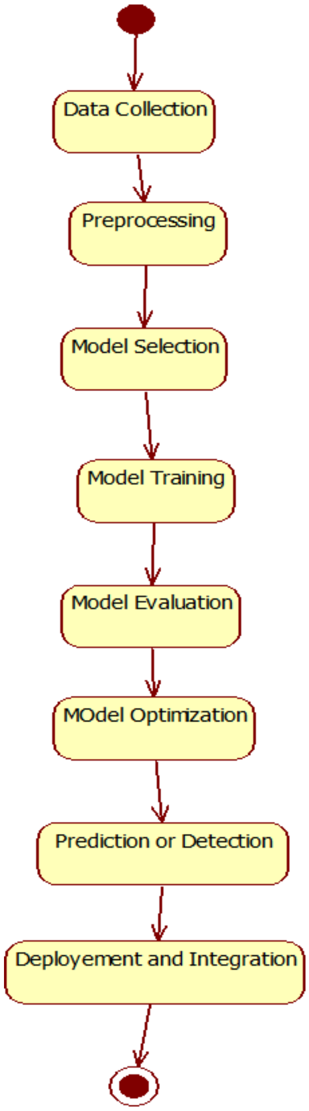
4.2.3 OBJECT DIAGRAM



EXPLANATION:

In the above digram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

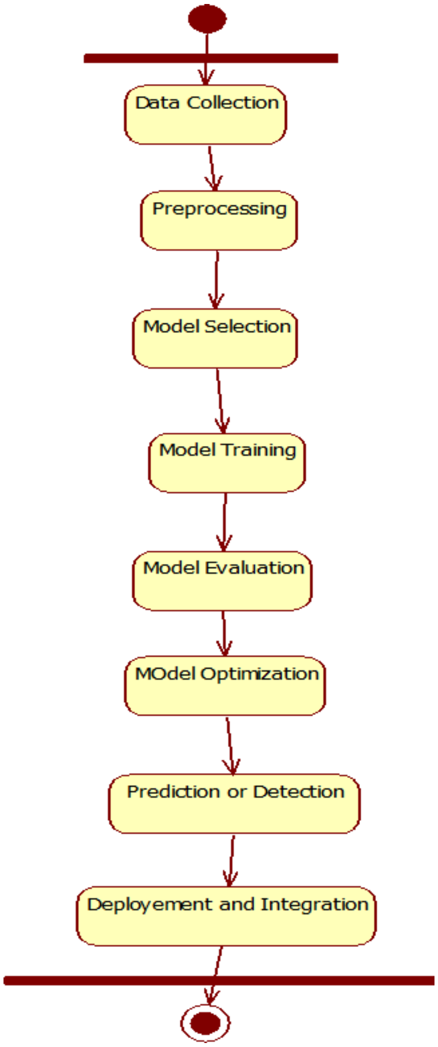
4.2.4 STATE DIAGRAM



EXPLANATION:

State diagram is a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

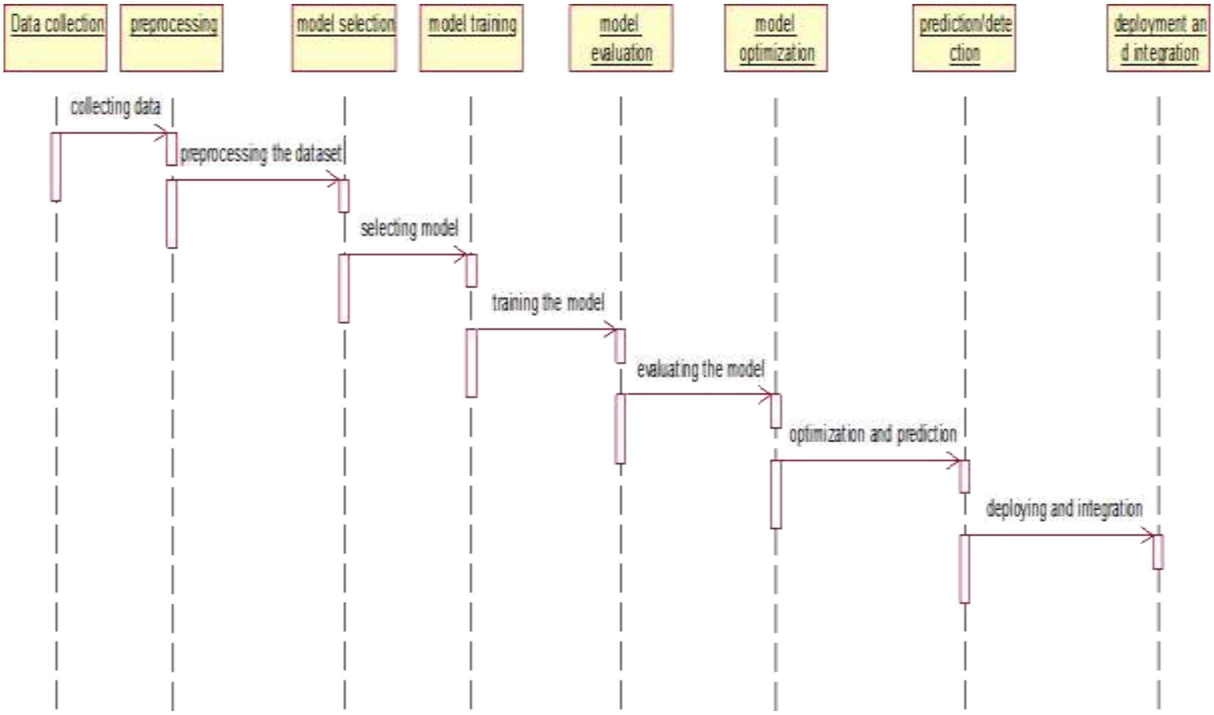
4.2.5 ACTIVITY DIAGRAM



EXPLANATION:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling 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.

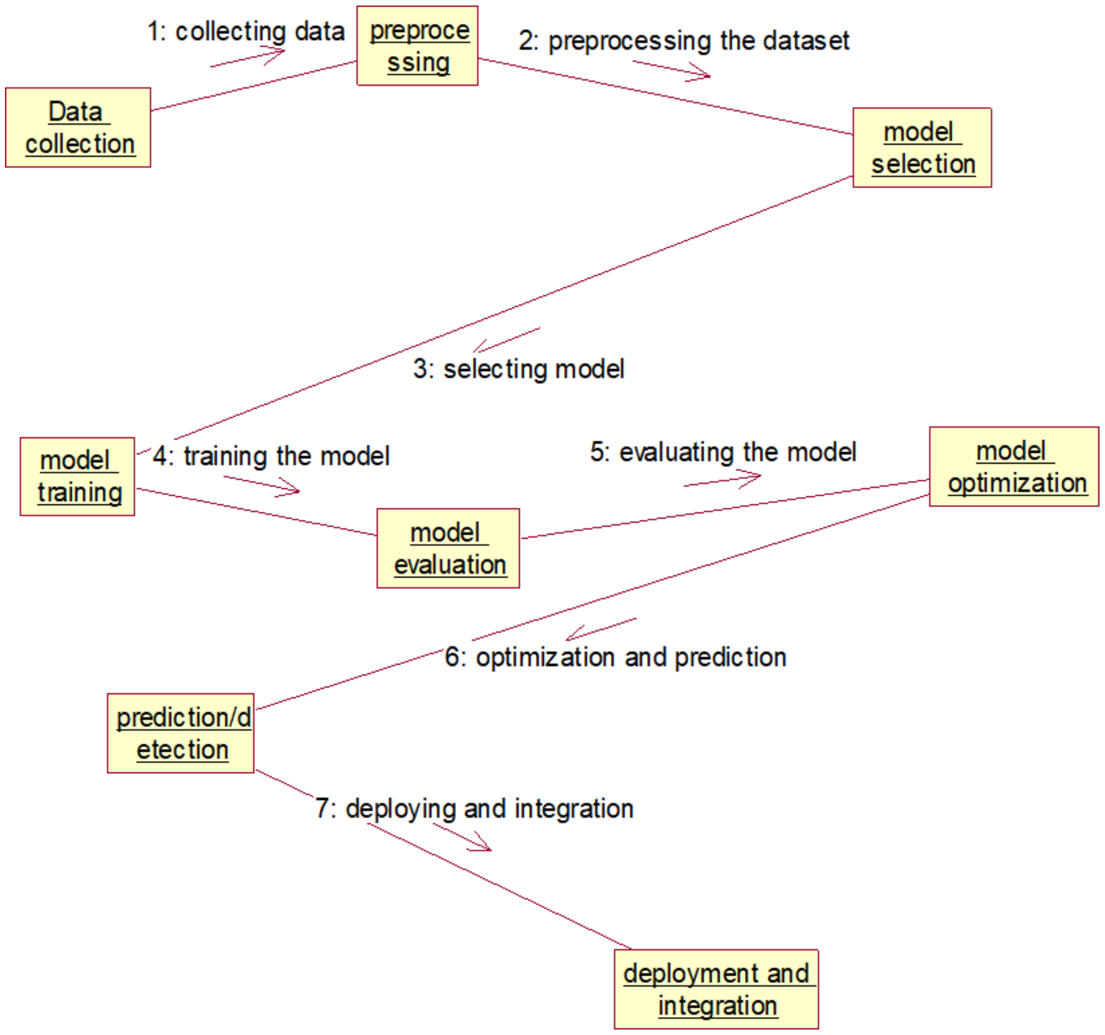
4.2.6 SEQUENCE DIAGRAM



EXPLANATION:

A sequence diagram in Unified Modelling 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 object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

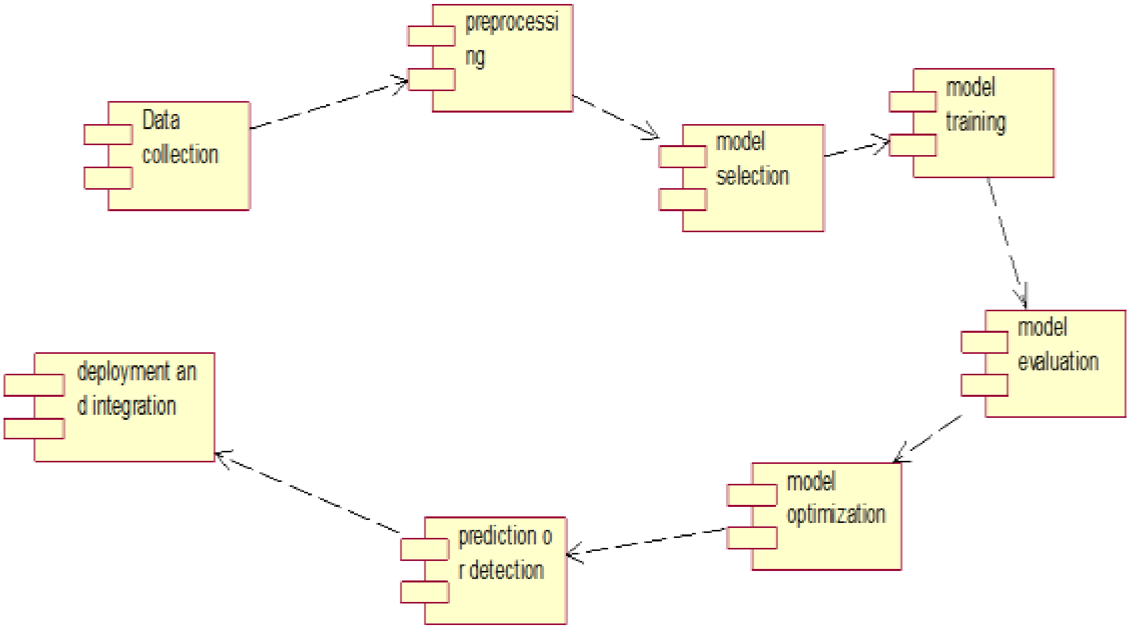
4.2.7 COLLABORATION DIAGRAM



EXPLANATION:

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language (UML). The concept is more than a decade old although it has been refined as modelling paradigms have evolved.

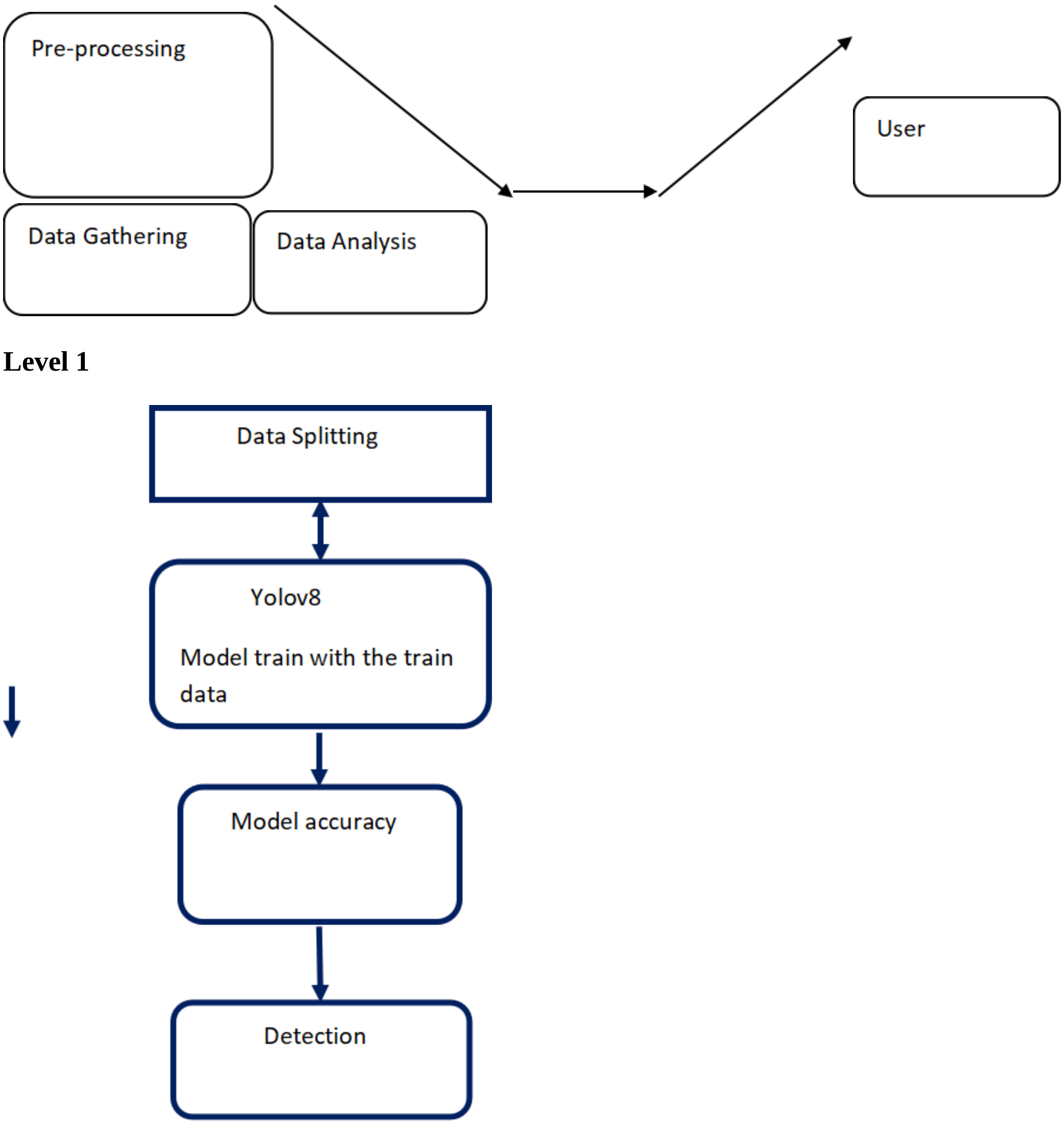
4.2.8 COMPONENT DIAGRAM



EXPLANATION

In the Unified Modelling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

4.2.9 DATA FLOW DIAGRAM



Level 0

Level 1

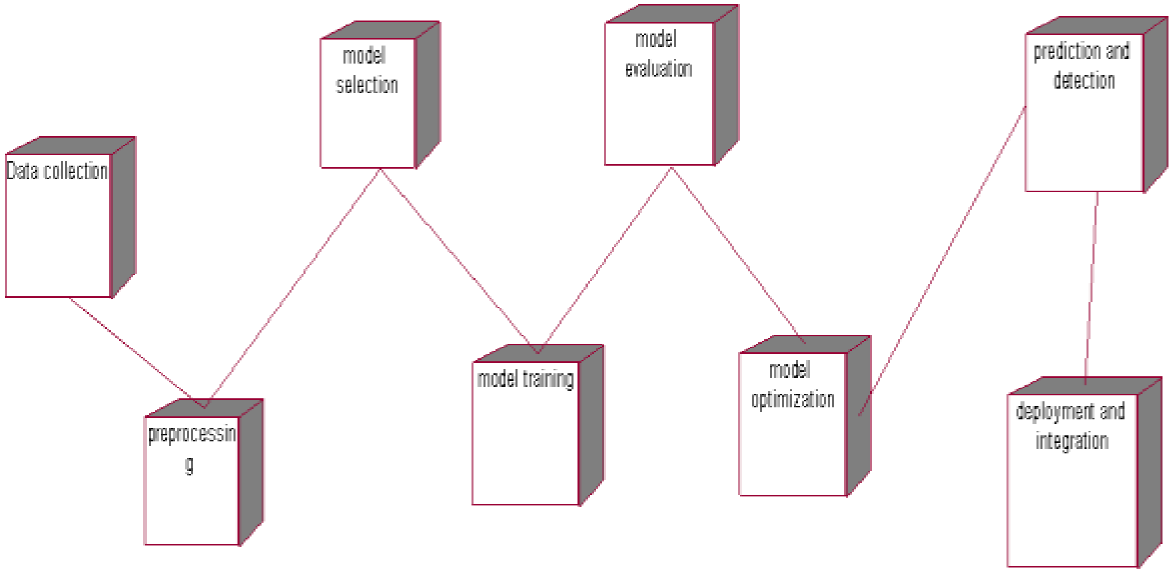
Fig 4.9: Data Flow Diagrams

EXPLANATION:

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

4.2.10 DEPLOYMENT DIAGRAM



EXPLANATION:

Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.

3.5 TOOLS AND TECHNOLOGIES USED

3.5.1 EXISTING TECHNIQUE:

The existing abstract describes the use of advanced technologies for detecting hand bone fractures through X-ray images using a hybrid model. The model combines deep learning algorithms, including YOLO NAS (You Only Look Once - Neural Architecture Search), EfficientDet, and DETR3 (DEtection TRansformer), all of which are known for their excellent object detection capabilities. YOLO NAS optimizes the neural network architecture for better performance, while EfficientDet focuses on computational efficiency without compromising accuracy, enabling fast detection of fractures. DETR3, an improved version of the DETR model, uses transformer-based techniques to enhance object detection, making it suitable for complex medical image analysis.

3.5.2 PROPOSED TECHNIQUE OR ALGORITHM USED

- In the context of the project described in the abstract, the reference to V8 likely pertains to a version or iteration of the proposed model, such as a refined or enhanced model architecture. While the exact details of V8 are not explicitly provided in the abstract, it is reasonable to assume that V8 refers to a more advanced version (Version 8) of the hybrid model that combines deep learning algorithms for detecting hand fractures.
- This version likely incorporates improvements or optimizations based on the feedback from earlier versions of the model, such as enhancements in accuracy, efficiency, and the ability to detect small or hairline fractures more effectively. It could also involve a more sophisticated use of the YOLO NAS, EfficientDet, and DETR3 algorithms, or even the integration of additional techniques such as data augmentation, model fine-tuning, or new preprocessing steps to improve detection performance in medical imaging.

3.5.3 DEVELOPMENT TOOLS

Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages**.**

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Importance of Python

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive −** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented −** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language −** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- **Easy-to-learn −** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain −** Python's source code is fairly easy-to-maintain**.**

- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Libraries used in python

- numpy - mainly useful for its N-dimensional array objects.
- pandas - Python data analysis library, including structures such as dataframes.
- matplotlib - 2D plotting library producing publication quality figures.

- scikit-learn - the machine learning algorithms used for data analysis and data mining tasks.

Figure: NumPy, Pandas, Matplotlib, Scikit-learn

3.6 IMPLEMENTATION

Coding:

```
 # import os

# from flask import Flask, render_template, request, redirect, url_for, send_from_directory, flash

# from ultralytics import YOLO

# import os

# import cv2

# import numpy as np

# from werkzeug.utils import secure_filename

# # Initialize the Flask app

# app = Flask(_name_)

# # Folder for uploaded images and results

# UPLOAD_FOLDER = 'uploads'

# RESULTS_FOLDER = 'results'

# # Create directories if they don't exist

# os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# os.makedirs(RESULTS_FOLDER, exist_ok=True)
#  app.config['UPLOAD_FOLDER']  =  UPLOAD_FOLDER

# app.config['RESULTS_FOLDER'] = RESULTS_FOLDER

# # Load the YOLO model

# model =
YOLO('D:/SHIVAA/CODES/YOLO/yolo.v1i.yolov8/runs/detect/train9/weights/best.pt')  # Path
to your trained YOLO weights

# app.secret_key = 'supersecretkey'

# # In-memory user database (For demonstration purposes only)

# users = {}

# @app.route('/')

# def home():

#     return render_template('home.html')

# @app.route('/index')

# def index():

#     return render_template('index.html')

# @app.route('/register', methods=['GET', 'POST'])

# def register():

#     if request.method == 'POST':

#       username  =  request.form['username']

#       password  =  request.form['password']

#      if username in users:

#            flash("Username already exists!", "danger")

#            return redirect(url_for('register'))

#        users[username] = password

#        flash("Registration successful! Please login.", "success")
#         return redirect(url_for('login'))
#     return render_template('register.html')
# @app.route('/login', methods=['GET', 'POST'])
# def login():
#     if request.method == 'POST':
#         username = request.form['username']
#         password = request.form['password']
#         if username in users and users[username] == password:
#             flash("Login successful!", "success")
#             return redirect(url_for('index'))
#         else:
#             flash("Invalid username or password!", "danger")
#             return redirect(url_for('login'))
#     return render_template('login.html')
# @app.route('/upload', methods=['POST'])
# def upload_file():
#     if 'file' not in request.files:
#         return redirect(url_for('home'))
#     file = request.files['file']
#     if file.filename == '':
#         return redirect(url_for('home'))
#     if file:
#         filename = secure_filename(file.filename)
#         file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
#         file.save(file_path)

#         if filename.lower().endswith(('.mp4', '.avi', '.mov', '.mkv')):  # Video formats

#             result_video_path = process_video(file_path, filename)

#             return render_template('result.html',

#                                    original_file=filename,

#                                    result_video=result_video_path)

#         else:  # Image file

#             results = model(file_path)

#             result_img_path = os.path.join(app.config['RESULTS_FOLDER'],
f"result_{filename}")

#             annotated_frame = results[0].plot()

#             cv2.imwrite(result_img_path, annotated_frame)

#             return render_template('result.html',

#                                    original_file=filename,

#                                    result_file=f"result_{filename}")

# @app.route('/uploads/<filename>')

# def uploaded_file(filename):

#     return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

# @app.route('/results/<filename>')

# def result_file(filename):

#     return send_from_directory(app.config['RESULTS_FOLDER'], filename)

# @app.route('/results/videos/<filename>')

# def result_video(filename):

#     video_path = os.path.join(app.config['RESULTS_FOLDER'], filename)
#     return send_from_directory(app.config['RESULTS_FOLDER'], filename,
mimetype='video/mp4')

# def process_video(input_path, filename):

#     cap = cv2.VideoCapture(input_path)

#     fourcc = cv2.VideoWriter_fourcc(*'avc1')  # Use H.264 codec for better compatibility

#     output_path = os.path.join(app.config['RESULTS_FOLDER'], f"result_{filename}")

#     fps = int(cap.get(cv2.CAP_PROP_FPS))

#     width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

#     height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

#     out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

#     while cap.isOpened():

#         ret, frame = cap.read()

#         if not ret:

#             break

#         # YOLO processing

#         results = model(frame)

#         annotated_frame = results[0].plot()  # Annotate frame

#         out.write(annotated_frame)

#     cap.release()

#     out.release()

#     return f"result_{filename}"  # Return relative path

# @app.route('/performance')

# def performance():

#     return render_template('performance.html')

# @app.route('/chart')
# def chart():

#     return render_template('chart.html')

# if _name_ == '_main_':

#     app.run(debug=True)

from flask import Flask, render_template, request, redirect, url_for, send_from_directory, flash

from ultralytics import YOLO

import os

import subprocess

from werkzeug.utils import secure_filename

import cv2

import numpy as np

from werkzeug.utils import secure_filename

# Initialize the Flask app

app = Flask(_name_)

# Folder for uploaded images and results

UPLOAD_FOLDER = 'uploads'

RESULTS_FOLDER = 'results'


# Create directories if they don't exist

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

os.makedirs(RESULTS_FOLDER, exist_ok=True)


app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

app.config['RESULTS_FOLDER'] = RESULTS_FOLDER
```

```
# Load the YOLO model
model = YOLO('C:/Users/91917/OneDrive/Desktop/MAJOR/CODE/Bone_Fracture/best.pt')  # Path to your trained YOLO weights

app.secret_key = 'supersecretkey'

# In-memory user database (For demonstration purposes only)
users = {}

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/index')
def index():
    return render_template('index.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        if username in users:
            flash("Username already exists!", "danger")
            return redirect(url_for('register'))

        users[username] = password
        flash("Registration successful! Please login.", "success")
        return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        if username in users and users[username] == password:
            flash("Login successful!", "success")
            return redirect(url_for('index'))
        else:
            flash("Invalid username or password!", "danger")
            return redirect(url_for('login'))

    return render_template('login.html')
```

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return redirect(url_for('home'))

    file = request.files['file']
    if file.filename == '':
        return redirect(url_for('home'))

    if file:
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        if filename.lower().endswith(('.mp4', '.avi', '.mov', '.mkv')):  # Video formats
            result_video_path = process_video(file_path, filename)
            return render_template('result.html',
                                   original_file=filename,
                                   result_video=result_video_path)

        else:  # Image file
            results = model(file_path)
            result_img_path = os.path.join(app.config['RESULTS_FOLDER'], f"result_{filename}")
            annotated_frame = results[0].plot()
            cv2.imwrite(result_img_path, annotated_frame)

            return render_template('result.html',

                                   original_file=filename,

                                   result_file=f"result_{filename}")


@app.route('/uploads/<filename>')

def uploaded_file(filename):

    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

@app.route('/results/<filename>')

def result_file(filename):

    return send_from_directory(app.config['RESULTS_FOLDER'], filename)

@app.route('/results/videos/<filename>')

def result_video(filename):

    return send_from_directory(app.config['RESULTS_FOLDER'], filename,
mimetype='video/mp4')

def process_video(input_path, filename):

    # OpenCV Video Capture

    cap = cv2.VideoCapture(input_path)

    output_path = os.path.join(app.config['RESULTS_FOLDER'], f"result_{filename}")


    # Get video properties

    fps = int(cap.get(cv2.CAP_PROP_FPS))

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    frame_size = (width, height)
```

```
# Temporary file for processed frames
temp_output = os.path.join(app.config['RESULTS_FOLDER'], "temp_video.mp4")
fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Temporary OpenCV video
out = cv2.VideoWriter(temp_output, fourcc, fps, frame_size)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # YOLO Processing: Annotate the frame with detections
    results = model(frame)
    annotated_frame = results[0].plot()
    out.write(annotated_frame)
cap.release()
out.release()
# FFmpeg Encoding for Final Output
ffmpeg_command = [
    "ffmpeg", "-y",  # Overwrite output if exists
    "-i", temp_output,   # Input temporary video
    "-c:v", "libx264",   # H.264 codec for encoding
    "-preset", "medium", # Encoding speed
    "-crf", "23",        # Quality (Lower is better)
    "-c:a", "aac",       # Audio codec
```

```
        "-strict", "experimental",
        output_path          # Final output video
    ]
    try:
        # Run FFmpeg to re-encode
        subprocess.run(ffmpeg_command, check=True)
    except subprocess.CalledProcessError as e:
        print(f"FFmpeg failed: {e}")
        raise

    # Remove temporary file
    os.remove(temp_output)
    return f"result_{filename}"  # Return relative path
@app.route('/about')
def performance():
    return render_template('about.html')
if _name_ == '_main_':
    app.run(debug=True)
```

3.7 TESTING AND VALIDATION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

TYPES OF TESTS

3.7.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

3.7.2 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

3.7.3 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

3.7.4 Performance Test

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

3.7.5 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

3.7.6 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization:

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

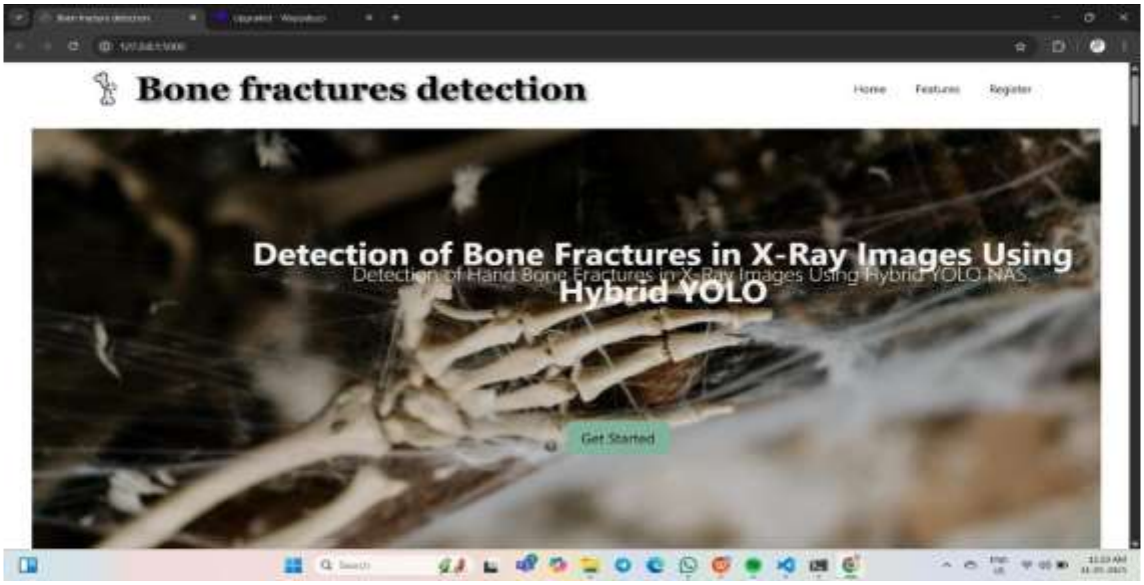
3.7.7 Build the test plan

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identity the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

3.7.8 Test Cases

SI.No	Test Scenario	User Action	Expected Result	Actual Result	Remarks
1	Image Upload	Upload a valid hand bone X-ray image	Image is accepted and passed to YOLOv8 model	Image successfully uploaded and detected	Pass
2	Invalid File Upload	Upload a non-image file (e.g., .pdf, .txt)	System shows error message for unsupported format	Error message displayed	Pass
3	Model Detection Accuracy	Upload X-ray with a known fracture	Correct fracture type is detected with bounding box	Detected fracture accurately	Pass
4	Real-Time Inference	Use live webcam stream for fracture detection	Model performs detection with minimal lag (<1 sec)	Detection processed in real time	Pass
5	Detection Performance (FPS)	Run continuous video input	System maintains >10 FPS detection speed	Maintained 12 FPS	Pass
6	User Registration	Register with valid username, email, and password	Account is created and success message displayed	User registered successfully	Pass
7	Duplicate Registr tion	Register with existing email	System should reject registration and show warning	Proper error shown for duplicate email	Pass
8	User Login	Login with valid credentials	User redirected to dashboard	Login successful	Pass
9	Invalid Login	Enter incorrect password	Error message shown for invalid credentials	Error alert displayed	Pass

3.8 SNAPSHOTS

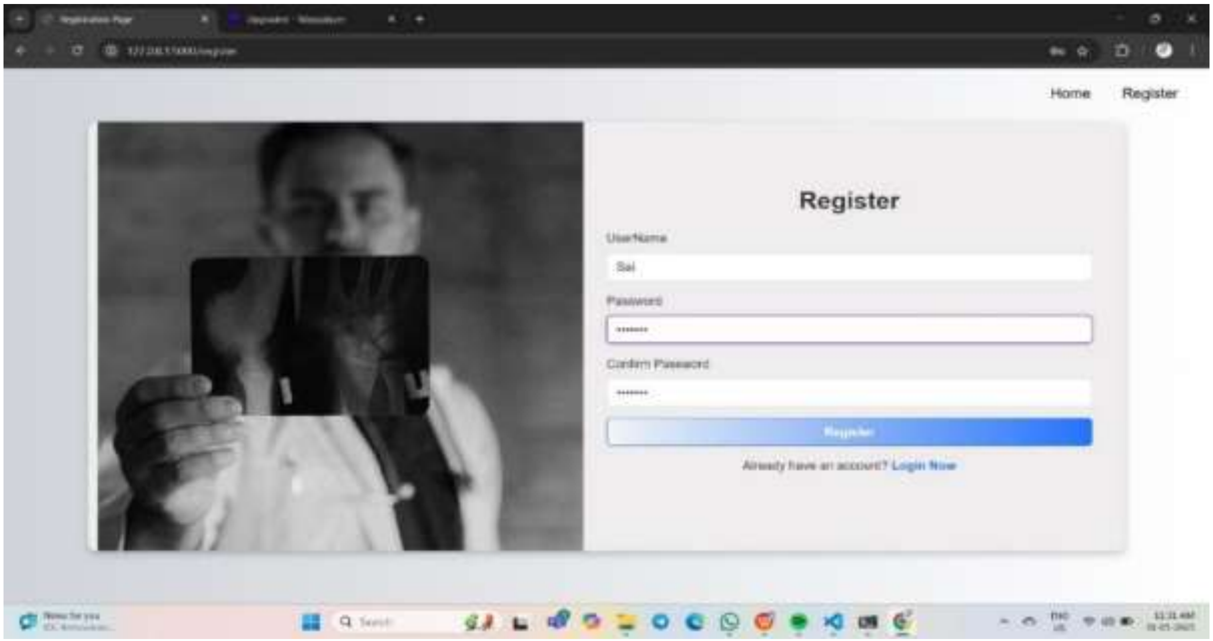


Bone fractures detection

Detection of Bone Fractures in X-Ray Images Using Hybrid YOLO

Detection of Hand Bone Fractures in X-Ray Images Using Hybrid YOLO NAS

Fig 3.8.1: Home page



Register

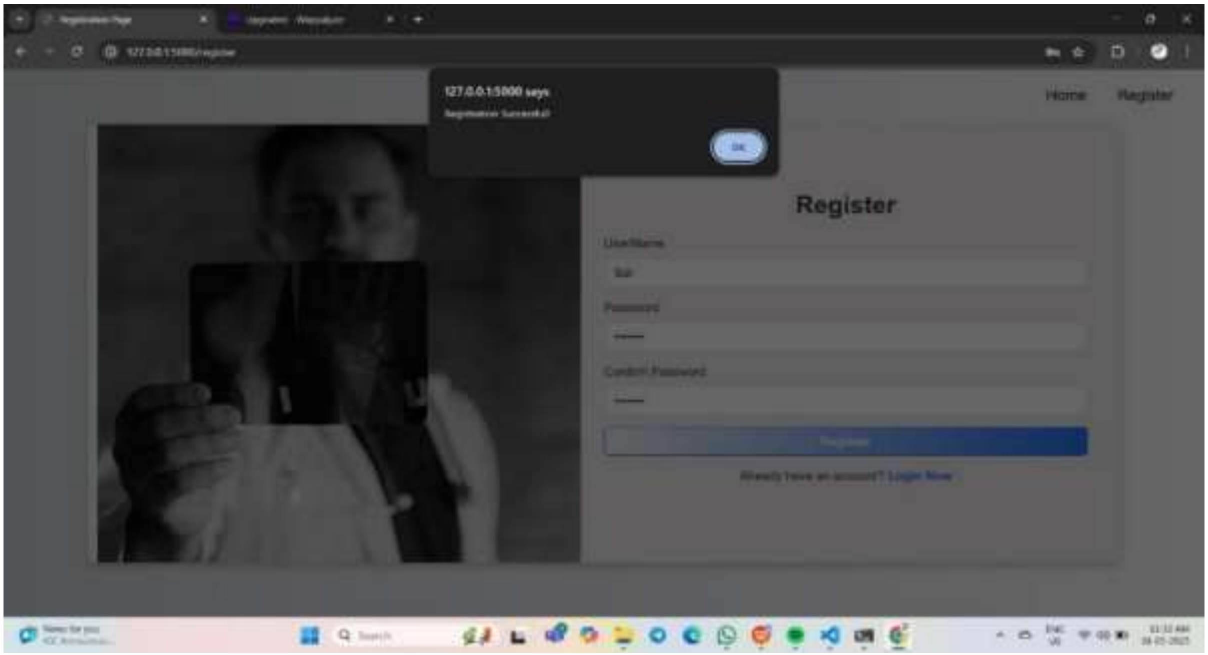
UserName: Sai

Password: ••••••••

Confirm Password: ••••••••

Already have an account? Login Now

Fig 3.8.2: Register page



127.0.0.1:5000 says

Registration Successful

OK

Register

UserName

Sai

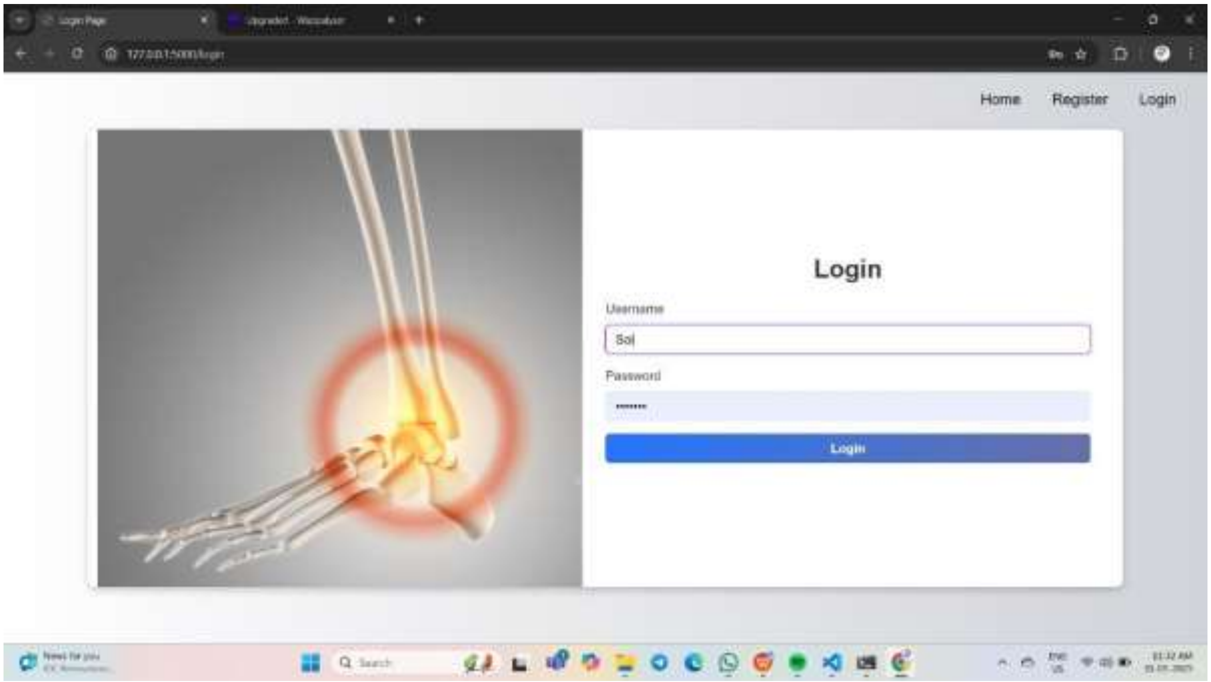
Password

••••••

Confirm Password

••••••

Fig 3.8.3: Register successful page



Login

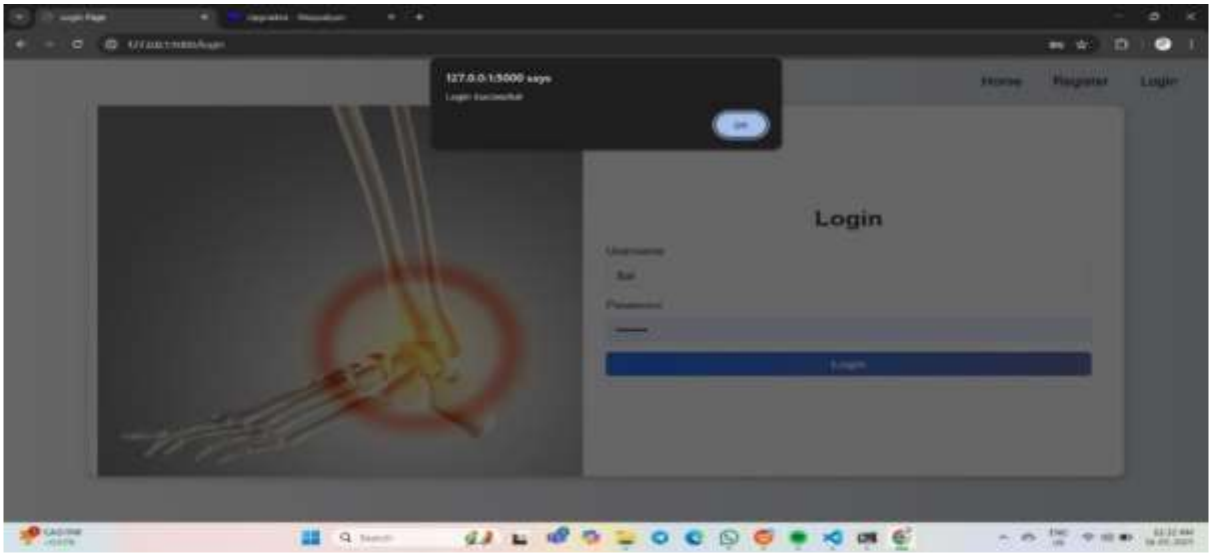
Username

Sai

Password

••••••••

Fig 3.8.4: Login page



127.0.0.1:5000 says

Login Successful

Fig 3.8.5:Login successful page

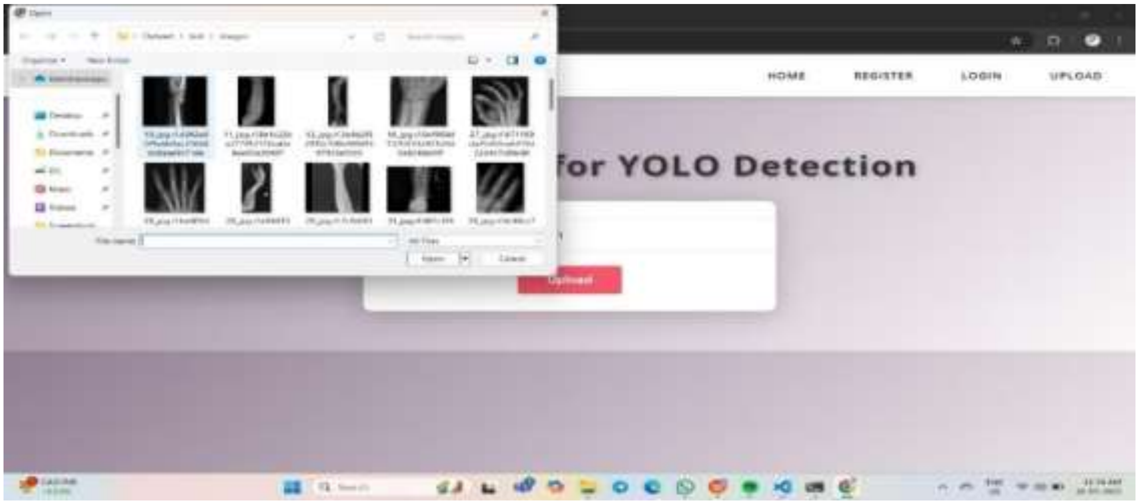


Fig 3.8.6: Image for YOLO detection page

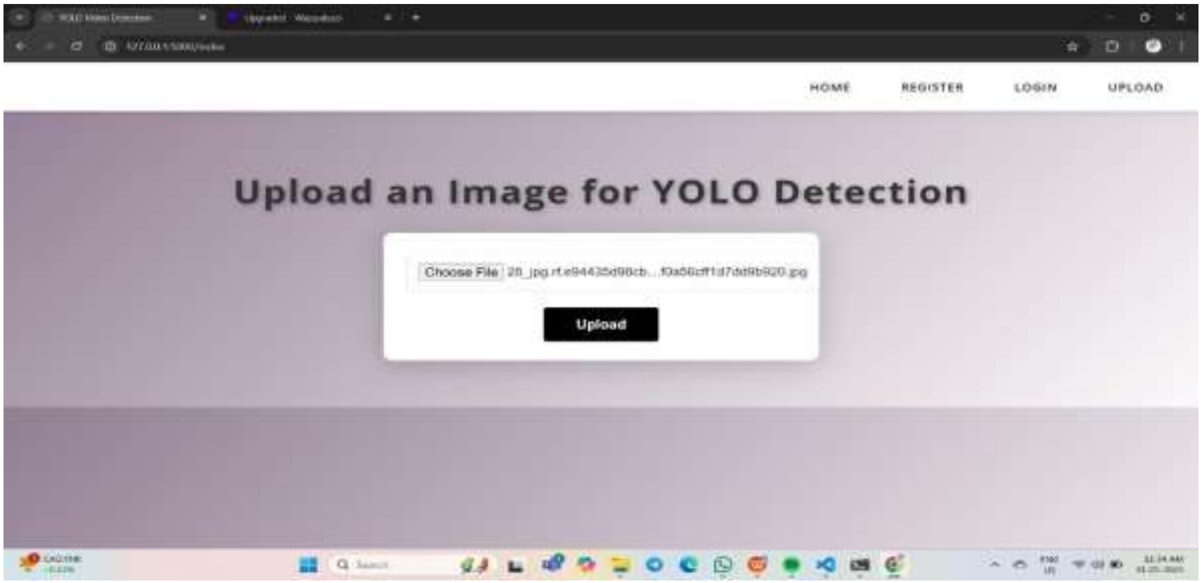

HOME
REGISTER
LOGIN
UPLOAD
Upload an Image for YOLO Detection
Choose File
28_jpg.rf.e94435d98cb....f0a56cff1d7dd9b920.jpg
Upload

Fig 3.8.7: Upload image for YOLO detection page

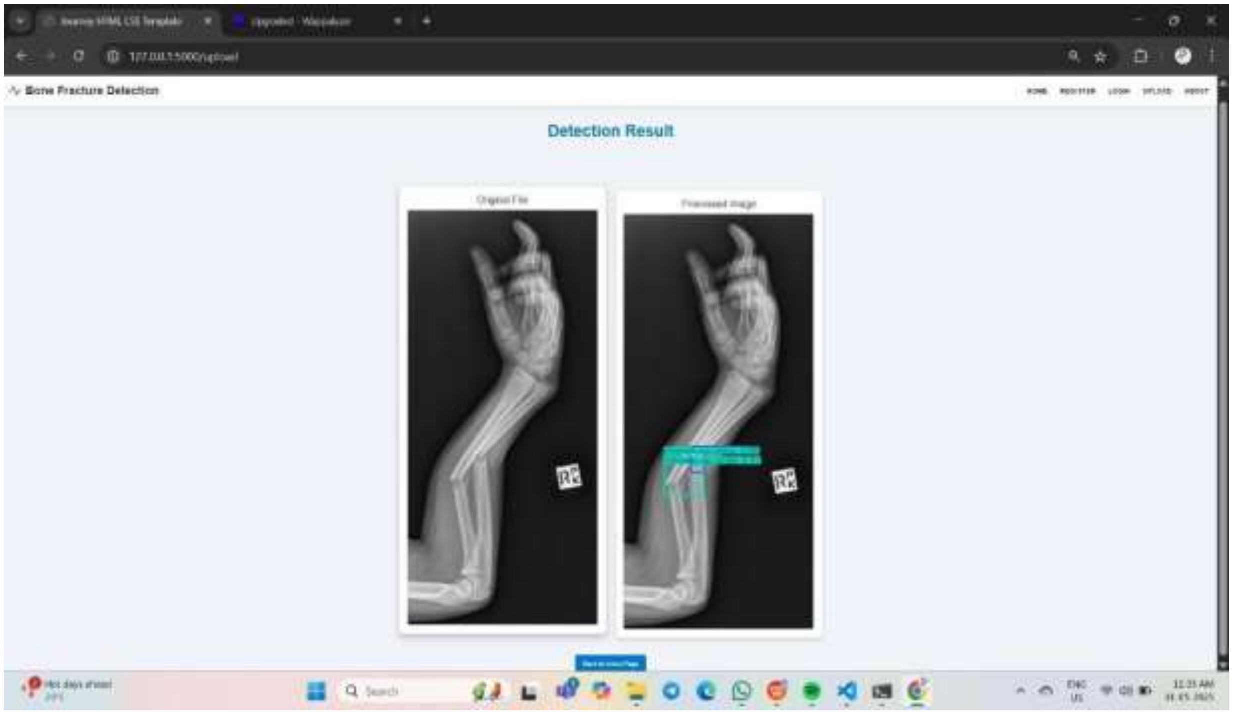


Fig 3.8.8:Detection Result page

CHAPTER 4

RESULTS & DISCUSSIONS

4.1 RESULT

The Hybrid YOLO V8 model has demonstrated high accuracy and efficiency in detecting hand bone fractures in X-ray images, significantly improving upon traditional detection methods. The model successfully identifies various fracture types—including subtle hairline and displaced fractures—with enhanced precision and processing speed, surpassing previous models like YOLO NAS.

Through the integration of a well-curated dataset of 1,200 hand-bone X-ray images, along with advanced preprocessing and data augmentation techniques, the model exhibits robust performance in clinical settings. The ability to provide real-time, reliable predictions enables radiologists to make faster and more informed decisions, ultimately benefiting patient care.

Additionally, the project lays the foundation for future enhancements, including dataset expansion, incorporation of 3D imaging, and integration of patient-specific data for personalized diagnostics. These advancements hold great potential for improving fracture detection accuracy and efficiency, contributing to better healthcare outcomes**.**

4.2 FUTURE ENHANCEMENTS:

The future of this project holds significant potential for further refinement and expansion. To improve the accuracy and reliability of the fracture detection system, the model could be trained on a larger and more diverse dataset, incorporating images from various demographics, age groups, and more types of fractures. This would ensure that the model generalizes well across different populations and fracture scenarios. Moreover, integrating 3D imaging techniques, such as CT scans or MRI, would provide a deeper understanding of complex fractures, enabling the model to detect fractures in multiple dimensions with higher precision. Additionally, incorporating transfer learning with pre-trained models from large datasets could improve the feature extraction process, helping the model detect even smaller and more subtle fractures that may be overlooked in conventional methods. Advanced post-processing techniques could be employed to further refine the model’s predictions, reducing false positives and negatives and enhancing the detection of complex fractures. Furthermore, integrating patient history and demographic data could improve diagnostic accuracy by personalizing predictions based on individual characteristics. Continuous learning mechanisms would also allow the model to adapt to new cases, keeping it up-to-date and improving its ability to detect emerging fracture types. Through these future enhancements, this project could evolve into an even more robust, accurate, and versatile tool for bone fracture detection, with wide-reaching applications in clinical settings worldwide.

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

In conclusion, this project successfully demonstrates the application of the Hybrid YOLO V8 model for detecting hand bone fractures in X-ray images, providing an efficient and accurate solution for real-time fracture detection. By leveraging the advanced capabilities of YOLO V8, the model can accurately identify various types of fractures, including subtle hairline and displaced fractures, which are often challenging to detect with traditional methods. The results show improved detection performance, with faster processing times and enhanced precision compared to previous models like YOLO NAS.The integration of a well-curated dataset of 1200 hand-bone X-ray images, along with advanced preprocessing and data augmentation techniques, has allowed the model to achieve a high level of accuracy and robustness. Furthermore, the model's ability to provide reliable predictions in a clinical setting presents a valuable tool for radiologists, enabling quicker and more informed decision-making.This project not only contributes to advancing medical image analysis but also lays the groundwork for future improvements, such as expanding the dataset, incorporating 3D imaging, and integrating patient-specific data for more personalized diagnosis. Ultimately, the successful deployment of this model can greatly enhance the speed, accuracy, and efficiency of fracture diagnosis, making it a vital tool for healthcare professionals and improving patient outcomes.

CHAPTER 6

REFERENCES

6.1 REFERENCES

1. **FormedHealth.org**, "How Do Hands Work," Inst. Quality Efficiency Health Care (IQWiG), Cologne, Germany, 2006. Accessed: Aug. 31, 2010. [Online]. Available:

2. **"Take Matters Into Your Own Hands: Understanding Hand Injuries in the Workplace,"** Occupational Health & Safety Online, Aug. 1, 2018. [Online]. Available:

3. A. Pinto, D. Berritto, A. Russo, F. Riccitiello, M. Caruso, M. P. Belfiore, V. R. Papapietro, M. Carotti, F. Pinto, A. Giovagnoni, L. Romano, and R. Grassi, "Traumatic fractures in adults: Missed diagnosis on plain radiographs in the emergency department," *Acta Bio Medica, Atenei Parmensis*, vol. 89, no. 1, pp. 111–123, 2018, doi: 10.23750/abm.v89i1-S.7015.

4. **Mayo Clinic**, "Broken Hand Symptoms and Causes," Accessed: Jun. 30, 2022. [Online]. Available:

5. R.-Y. Ju and W. Cai, "Fracture detection in pediatric wrist trauma X-ray images using YOLOv8 algorithm," 2023. arXiv:2304.05071.

6. K. D. Ahmed and R. Hawezi, "Detection of bone fracture based on machine learning techniques," *Meas., Sensors*, vol. 27, Jun. 2023, Art. no. 100723, doi: 10.1016/j.measen.2023.100723.

7. S. R. Karanam, Y. Srinivas, and S. Chakravarty, "A systematic review on approach and analysis of bone fracture classification," *Mater. Today, Proc.*, vol. 80, pp. 2557–2562, Jan. 2023, doi: 10.1016/j.matpr.2021.06.408.

8. L. Xue, W. Yan, P. Luo, X. Zhang, T. Chaikovska, K. Liu, W. Gao, and K. Yang, "Detection and localization of hand fractures based on GA_Faster R-CNN," *Alexandria Eng. J.*, vol. 60, no. 5, pp. 4555–4562, Oct. 2021, doi: 10.1016/j.aej.2021.03.005. [Online].

9. I. Kandel, M. Castelli, and A. Popovič, "Musculoskeletal images classification for detection of fractures using transfer learning," *J. Imag.*, vol. 6, no. 11, p. 127, Nov. 2020, doi: 10.3390/jimaging6110127.

10. K. C. Santos, C. A. Fernandes, and J. R. Costa, "Feasibility of bone fracture detection using microwave imaging," *IEEE Open J. Antennas Propag.*, vol. 3, pp. 836–847, 2022, doi: 10.1109/OJAP.2022.3194217.

11. A. Inui, Y. Mifune, H. Nishimoto, S. Mukohara, S. Fukuda, T. Kato, T. Furukawa, S. Tanaka, M. Kusunose, S. Takigami, Y. Ehara, and R. Kuroda, "Detection of elbow OCD in the ultrasound image by artificial intelligence using YOLOv8," *Appl. Sci.*, vol. 13, no. 13, p. 7623, Jun. 2023, doi: 10.3390/app13137623.

12. M. A. Kassem, S. M. Naguib, H. M. Hamza, M. M. Fouda, M. K. Saleh, and K. M. Hosny, "Explainable transfer learning-based deep learning model for pelvis fracture detection," *Int. J. Intell. Syst.*, vol. 2023, Jun. 2023, Art. no. 3281998, doi: 10.1155/2023/3281998.

13. B. Zhang, C. Jia, R. Wu, B. Lv, B. Li, F. Li, G. Du, Z. Sun, and X. Li, "Improving rib fracture detection accuracy and reading efficiency with deep learning-based detection software: A clinical evaluation," *Brit. J. Radiol.*, vol. 94, no. 1118, Feb. 2021, Art. no. 20200870, doi: 10.1259/bjr.20200870.

14. M. S. A. M. Bevers, C. E. Wyers, A. M. Daniels, E. A. Audenaert, S. M. J. van Kuijk, B. van Rietbergen, P. P. M. M. Geusens, S. Kaarsemaker, H. M. J. Janzing, P. F. W. Hannemann, M. Poeze, and J. P. van den Bergh, "Association between bone shape and the presence of a fracture in patients with a clinically suspected scaphoid fracture," *J. Biomech.*, vol. 128, Nov. 2021, Art. no. 110726, doi: 10.1016/j.jbiomech.2021.110726. [Online].

15. D. P. Yadav, A. Sharma, S. Athithan, A. Bhola, B. Sharma, and I. B. Dhaou, "Hybrid SFNet model for bone fracture detection and classification using ML/DL," *Sensors*, vol. 22, no. 15, p. 5823, Aug. 2022, doi: 10.3390/s22155823.

16. Y. Ma and Y. Luo, "Bone fracture detection through the two-stage system of crack-sensitive convolutional neural network," *Informat. Med. Unlocked*, vol. 22, Dec. 2021, Art. no. 100452, doi: 10.1016/j.imu.2020.100452. [Online].

17. E. Li and Q. Tan, "Role of ultrasound imaging to assess and diagnose various body fractures: Systemic review and meta-analysis," *J. Radiat. Res. Appl. Sci.*, vol. 15, no. 3, pp. 357–364, Sep. 2022, doi: 10.1016/j.jrras.2022.08.008. [Online]. Available:

18. Y. Rakesh and A. Akilandeswari, "Evaluation of Bernsen algorithm for bone fracture detection based on edge detection," *J. Surv. Fisheries Sci.*, vol. 10, no. 1, pp. 2060–2068, 2023.

19. D. Mishra and A. K. Mishra, "Evolution of artificial intelligence in bone fracture detection," *Int. J. Reliable Quality E-Healthcare*, vol. 11, no. 2, pp. 1–17, Jul. 2022, doi: 10.4018/ijrqeh.299958.

20. J. Terven and D. Cordova-Esparza, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," 2023. arXiv:2304.00501.

21. T. A. Sumi, N. Basnin, M. S. Hossain, K. Andersson, and M. S. Hoassain, "Classifying humerus fracture using X-ray images," in *Proc. 4th Ind. Revolution Beyond*, Lecture Notes in Electrical Engineering, vol. 980, Singapore, M. S. Hossain, S. P. Majumder, N. Siddique, and M. S. Hossain, Eds., Singapore: Springer, 2023, pp. 527–538, doi: 10.1007/978-981-19-8032-9_37.