**A**

**FIELD BASED PROJECT REPORT**

**on**

|  |
| --- |
| **BONE FRACTURE DETECTION AND CLASSIFICATION USING DEEP LEARNING APPROACH** |

**BACHELOR OF TECHNOLOGY**

**IN**

**ARTIFICIAL INTELLIGENCE &MACHINE LEARNING**

**SUBMITED BY**

**[BATCH :- 13]**

**SEPOOR VENKATARAMANA : 227Y1A66C0**

**NIRATI VINAYAKA : 227Y1A66C1**

**UNDER THE GUIDENCE OF**

**OF**

**Mr.R.MALLIKARJUNA**

**(Assistant Professor)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AIML)**

**MARRI LAXMAN REDDY**

**INSTITUTE OF TECHNOLOGY AND MANAGEMENT**

**(AUTONOMOUS)**

**(Affiliated to JNTU-H, Approved by AICTE New Delhi and Accredited by NBA & NAAC With ‘A’ Grade)**

**JULY 2024**

****

|  |  |
| --- | --- |
| **CERTIFICATE** | |
| This is to certify that the project report titled “**Bone fracture detection and classification”** is being submitted by S**.Venkataramana (227Y1A66C0) & N.Vinayaka (227Y1A66C1)** in II B.Tech II Semester C**omputer Science & Engineering(AIML)** is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree. | |
|  |  |
| **Internal Guide** | **HOD** |
|  |  |
| **Principal** | **External Examiner** |

****

**DECLARATION**

|  |  |
| --- | --- |
| We hereby declare that the Field based Project Report entitled, “**Bone fracture detection and classification”** submitted for the B.Tech degree is entirely my work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree. | |
|  | |
| **Date: 21/06/2024** | |
|  | |
|  |  |
| **S.VENKATARAMANA (227Y1A66C0)** | **N.VINAYAKA**  **( 227Y1A66C1)** |

****

**ACKNOWLEDGEMENT**

|  |
| --- |
| We are happy to express our deep sense of gratitude to the principal of the college  **Dr.R.MURALI PRASADD,**Professor,Department of Computer Science and Engineering(AIML)**, Marri Laxman Reddy Institute of Technology & Management,** for having provided me with adequate facilities to pursue my project.  We would like to thank **Dr.B. RAVI PRASAD,** Assoc. Professor and Head, Department ofC**omputer Science & Engineering(AIML),** Marri Laxman Reddy Institute of Technology & Management, for having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.  We are very grateful to our project guide **Mr.R.MALLIKARJUNA ,** Assistant. Professor, Department of C**omputer Science & Engineering(AIML)** , Marri Laxman Reddy Institute of Technology & Management, for his extensive patience and guidance throughout my project work.  We sincerely thank our seniors and all the teaching and non-teaching staff of the Department of Computer Science for their timely suggestions, healthy criticism and motivation during the course of this work.  We would also like to thank our classmates for always being there whenever we needed help or moral support. With great respect and obedience, we thank our parents and brother who were the backbone behind our deeds.  Finally, we express our immense gratitude with pleasure to the other individuals who have either directly or indirectly contributed to our need at right time for the development and success of this work. |

****

**CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S NO.** |  | **TITLE** | **PAGE NO.** |
|  | ABSTRACT | | **7** |
|  | LIST OF FIGURES | | **8** |
|  | SYMBOLS & ABBREVIATIONS | | 9 |
| **1** | **OVERVIEW** | | 10 |
|  | 1.1 | Introduction | 10 |
|  | 1.2 | Literature Survey | 11 |
| **2** | **METHODOLOGY** | | 14 |
|  | 2.1 | Data collection and preprocessing | 17 |
|  | 2.2 Model and Analysis | | 18 |
|  | 2.3 | System Testing | 20 |
|  | 2.4 | Problem statement | 22 |
|  | 2.5 Proposed System | | 22 |
|  | 2.6 Dataset | | 23 |
|  | 2.7 CNN(Convolution Nueral Network) | | 24 |
|  | 2.8 RestNet50 Model | | 27 |
|  | 2.9 Algorithm | | 28 |
| **3** | **SOFTWARE DEVOLOPEMENT** | | **29** |
|  | 3.1 | Why choose PythonI | 29 |
|  | 3.2 | Modules | 35 |
| **4** | **CODE IMPLEMENTATION** | | 37 |
|  | 4.1 | Code | 37 |
|  | 4.2 | Output | 52 |
|  | 4.3 | Accuracy | 55 |
| **5** | **CONCLUSION** | | **56** |
| **6** | **REFERENCES** | | **57** |

**ABSTRACT**

Identification of faults via computer-based techniques is a growing trend in all fields these days. Two main characteristics of Bone Fracture Detection are fast identification and high precision which is described by highly sensitive device by incorporating advanced techniques and effective resource usage. The effect of undue external stress above the limits of what the bone may tolerate is a crack in a bone or bone fracture. Canny Edge detection is an image processing technique that identifies bone fracture by utilizing automatic fracture detection efficiently and overcomes the question of noise reduction. There are many methodologies accessible in today's world for edge detection, such as Sobel, Canny, Log, Prewitt, and Robert.

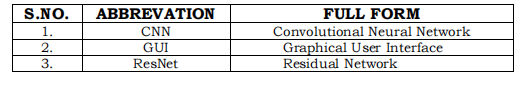
These processes, though, are hampered by crucial limitations such as a lack of capacity to conduct multi resolution research, culminating in the failure to identify small information during the analysis. The other major drawback of the techniques is that they operate well with high resolution and high-quality pictures, but because of their intrinsic lack of ability to differentiate between edges and noise elements, they do not work well with blurry images. The approach being suggested uses the CNN algorithm to solve these issues. The findings of the simulations carried out suggest that the approach proposed is a far more effective system for conducting edge detection on aggregate scales. The suggested system has also shown to be sufficiently resilient to retrieve the required details and do the necessary analysis on key portions of the images and manage noise in a much better way than the edge detectors currently usable.

**Keywords:** Machine Learning, Image processing using X-ray images, Canny Edge Detection, SVM algorithm

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG. NO** | **FIG. NAME** | **PAGE NO.** |
| 2.1 | Residual Networks (ResNet50) | 15 |
| 2.2 | Block Diagram | 15 |
| 2.3 | Structure of the network | 25 |
| 3.1 | search python.org | 33 |
| 3.2 | Go to downloads and select windows | 34 |
| 3.3 | Download Windows installer(64-bit) | 34 |
| 3.4 | Now select python.exe to path & install the IDLE3.2 | 35 |
| 4.1 | Bone Fracture Detection(processing image) | 53 |
| 4.2 | Normal Elbow | 53 |
| 4.3 | Fractured Hand | 54 |
| 4.4 | Accuracy | 55 |
| 4.5 | Model Accuracy | 55 |

**SYMBOLS & ABBREVIATIONS**



**CHAPTER 1**

**OVERVIEW**

**1.1 INTRODUCTION**

In this chapter, brief introduction to the different types of methodologies to

detect the bone fracture are explained.

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 PyQt or 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 ResNet-50 model, which has been improved

on a dataset of photos of broken and unfractured bones, will be applied to

the submitted image. The model will provide a categorization result, which

the GUI interface will present along with the uploaded picture. A message

will notify the user if a fracture is found.

Bone fractures are mostly caused by the automobile accident or bad fall. The

bone fractured risk is high in aged people due to the weaker bone [3]. The

fracture bone heals by giving proper treatment to the patient. The doctor

uses x-ray or MRI (Magnetic Resonance Imaging) image to diagnose the

fractured bone.

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[9]. 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 are explained in the chapter 2.8.

There are 2 processes during this analysis. First, the bone strip was removed from the X-ray images using a non-linear anisotropic diffusion technique. Second, Hough's modified transformation was created with automatic detection of the height and also of the extent and direction of victimization based on the gradient of the calculation line parameter. The system has the flexibility to produce an extremely correct designation of fractures in the bones of hands by using the X-ray images. Few discriminatory photo options were also used when the noise was eliminated and improved. The performance of the system has accuracy greater than eighty-six. The characteristics of homogeneity, contrast, energy, and correlation have been calculated step by step with GLCM to classify the broken bone and also the violated bone. We tend to has also shown an associated degree accuracy obtained by the system with eighty-six percent. However, we tend to report together that the performance of such a technique can also be improved further through the use of multiple GLCM functions, which can be performed in the future to classify the bone in numerous degrees of fracture specifically.

**1.2 LITERATURE SURVEY**

In recent years, bone fracture diagnosis in medical image analysis has

become more and more dependent on deep learning models. Popular deep

learning model ResNet-50 has been used to this challenge.

Researchers employed a ResNet-50 model in a study that was published in

the Journal of Digital Imaging to identify bone fractures in X-ray images. The

model outperformed previous models like Alexey and VGG-16 by identifying

fractures with an accuracy of 96.9%.

A ResNet-50 model was employed in a different research that was published

in the Journal of Medical Systems to find fractures in wrist X-ray pictures.

The model has a 91.7% accuracy rate, an 84.1% sensitivity rate, and a 94.4%

specificity rate. The study came to the conclusion that the ResNet-50 model

could be helpful in identifying wrist fractures[2].

A third research examined several deep learning models for spotting bone

fractures in X-ray pictures, and it was published in the Journal of Digital

Imaging. With an accuracy of 98.8%, the researchers discovered that ResNet-

50 performed better than models like Inception-v3 and DenseNet-121.

There are a number of libraries that may be used to create a Python GUI for

bone fracture detection, including PyQt, Tkinter, and wxPython. The user

friendly interface offered by these libraries makes it easy to add photographs

and view the result.

In , a deep neural network model has been developed to classify the fracture and healthy bone. The deep learning model gets over fitted on the small data set. Therefore, data augmentation techniques have been used to increase the size of the data set. The three experiments have been performed to evaluate the performance of the model using softmax and Adam optimizer. The classification accuracy of the proposed model is 92.44% for the healthy and the fractured bone using 5 fold cross validation. The accuracy on 10% and 20% of the test data is more than 95% and 93% respectively. The proposed model performs much better than of the 84.7% and 86% of the .In ,these processes, though, are hampered by crucial limitations such as a lack of capacity to conduct multi resolution research, culminating in the failure to identify small information during the analysis. The other major drawback of the techniques is that they operate well with high resolution and high-quality pictures, but because of their intrinsic lack of ability to differentiate.

In [3], to overcome these problems, we proposed a transfer learning, Faster R-CNN deep learning model for fracture detection and classification with Region Proposal Network (RPN). Also, we retrained the top layer of the model by using inception v2 (version2) network architecture upon 50 x-ray images. This model was trained in 40k steps and its training stopped when loss remains only 0.0005. We evaluated the proposed model concerning detection and classification. We classify bone fracture x-ray images into two classes fracture and non-fracture also locate the location of fractures with a rectangle box. The overall accuracy has achieved from this method is 94% with respect to classification and detection. Our study shows that the proposed method is simple and efficient, which is worthwhile for dynamic detection, classification of fracture, now doctors and radiologists interact with more and more patient and overcome the workload. Furthermore, this approach improves the results, the run time performance and detection quality as compared to state-of-the-art techniques. In [4], bone fractures are the major and common issues faced by many people. These fractures often occur during accidents. To predict these fractures doctors are using x-rays. Sometimes it is difficult to predict whether it is fractured or not through the x-rays manually. These x-rays show a clear picture of the damage but the main issue is that some physicians are overlooking the small fractures which may cause a lot of damage in the future to that particular person. Model which analyses and classifies the images of hand, leg, chest, fingers and wrist fractures in a clear way. There are many other techniques to detect these fractures and this project is molded by using some artificial intelligence applications using machine learning and deep learning techniques. This project investigates specifically various models dependent on Convolutional Neural Networks which helps us to provide a better solution as it is a step-by-step process of image analyzing algorithm to predict whether the bone is fractured or normal. By comparing 3 types of CNN models which are ConvNet/CNN, VGG16 & R-CNN with the same image dataset, R-CNN gave the best accuracy. In [5], to overcome these problems, we proposed a transfer learning, Faster R-CNN deep learning model for fracture detection and classification with Region Proposal Network (RPN). Also, we retrained the top layer of the model by using inception v2 (version2) network architecture upon 50 x-ray images. This model was trained in 40k steps and its training stopped when loss remains only 0.0005. We evaluated the proposed model concerning detection and classification

The overall accuracy has achieved from this method is 94% with respect to classification and detection. Our study shows that the proposed method is simple and efficient, which is worthwhile for dynamic detection, classification of fracture, now doctors and radiologists interact with more and more patient and overcome the workload. Furthermore, this approach improves the results, the run time performance and detection quality as compared to state-of-the-art techniques. In [6], this paper gives a technique to identify bone fracture using machine learning algorithms, by which workload for orthopedics can be reduced. The significant use of machine learning in this era of big medical data would help gather information from the available x-ray images rather than spending hours in the radiology departments. This paper presents imaging technologies used to identify bone fracture in the human body and give quick results once the x-ray has been taken. In [7], the paper proposes a novel method, which is called boring survey based fracture detection (BSFD), to automatically detect FFP in 3D CT images. FFP appears as a tiny crack of the bone that is hard to find by surface observation. Thus, BSFD surveys the FFP by boring from the pelvic surface. It first segments the pelvic bone region from CT images. At every surface point, it bores a small quadrangular prism whose center is the point of interest. The prism is composed of voxels with CT values and is feed to a 3D convolutional neural network which predicts a probability of fracture. The proposed method was evaluated by using 110 elderly subjects with pelvic fractures. The AUC was 0.84 for training subjects and 0.77 for evaluation subjects. In addition, 3D visualization of fracture probability superimposed on the pelvic bone surface is provided for qualitative evaluation and supporting FFP diagnosis by physicians. The boring survey approach will be effective to detect FFP because the bored volume extracts the unique characteristics of tiny fractures in CT images. In [8], the feasibility study proposes a low-cost and portable bone fracture detection method and device to help this under-served segment of patients. Drawing on previously published work regarding the automated detection of mechanical fractures using induced vibrations in an industrial setting, this paper presents a technique to replicate and improve upon manual detection techniques using a tuning fork and stethoscope by using digital signal processing and machine learning techniques. In order to make fracture detection more accessible, the prototype device presented does not require any specialized skills to operate, maintains portability, is automated, and has the potential to be manufactured inexpensively. Fractures are detected by inducing vibrations in the bone and measuring the resulting signal to detect structural defects. Using animal bones with synthetic soft tissues to replicate the dampening effects of muscle and connective tissue, machine learning models were trained and tested, achieving 93.6% accuracy. The proposed technique may also prove effective in-vivo although further testing is required.

**CHAPTER 2**

**METHODOLOGY**

In this chapter,we are know about the methodologies for the bone detection

are briefly explained.

The methodology for building a bone fracture detection system using ResNet-

50 and Python GUI involves several steps:

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.

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

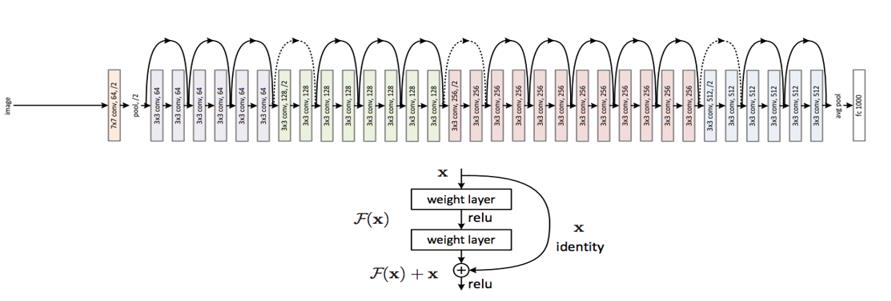
performance on this task and its architecture is shown in figure 2.1.

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.



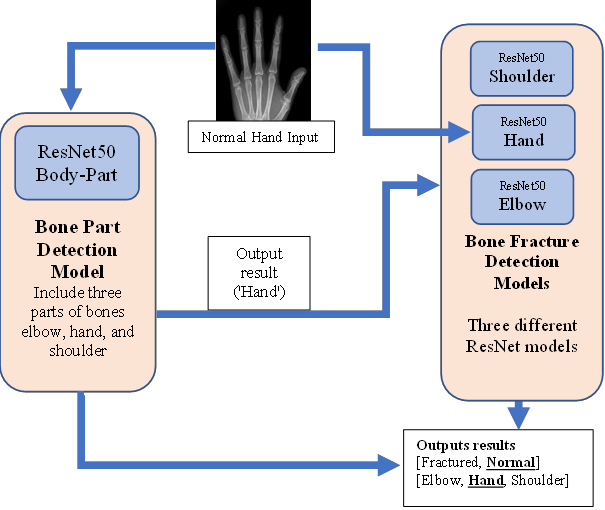
**Figure 2.1:** Residual Networks(ResNet50)

ResNet50 is a variant of ResNet50 model which has 48 Convolution layers

along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating

3points operations. It is a widely used ResNet model and we have

explored **ResNet50 architecture** in depth.



**Figure 2.2:** Block Diagram

In the above Figure 2.2 shows the block diagram of ResNet50 and here

shows how the ResNet50 works and gives the result.

**Step 1: Preprocessing**

These stages consist of the procedures that enhance the features of an input X-ray image so that he result image improves the performance of the subsequent stages of the proposed system.

**Step 2: Edge Detection**

It is based on analyzing the changes in the intensity in the image. However, the quality of edge detection is highly dependent on lighting conditions, the presence of objects of similar intensities, density of edges in the scene and noise. There are different algorithms for edge detections such as Canny, Laplacian and Sobel. In our experiments, the best results were obtained by using a modified version of the Canny edge detection algorithm in which the contrast is enhanced using a histogram equalization step. This finding is in accordance with the Nadernejad et al. result shows the results of using different edge detection algorithms.

**Step 3: Segmentation**

Image segmentation is the fundamental step to analyses image and extract data from them. It is an operation of partitioning an image into a collection of connected sets of pixels. The main purpose of interest in an image which helps in an image which helps in annotation of the object scene. There are three main approaches of image segmentation which are region approach, boundary approach, and edge approach. **Step 4: Image classifier**

In this step different classifier is used like SVM (Support Vector Machine), K-Nearest Neighbor (KNN), Back Propagation Neural Network (BPNN), Nave Byes(NB).

**Step 5: Fracture detection**

The last stage of this system is fracture detection it I performed by the procedures. First, the useful features. Here is an explanation of the performance of the system: extracted from the image. And then, these features are used to detect fracture or non-fracture image

1. First user must input an image to be processed; the image will then be carried filtering to remove noise that exists in the image.

2. The next step will performed after image filtering process, the image will during Canny Edge method, it will give results more visible lines on an X.

3. The system then check and combines the results of early detection canny with the Original image, then user can clearly see the shape of the bone and this combine will be processed by the system.

4. To detect the location of the fracture in the image, the system use shape detection with image matching process expressed when the line has an end, and give the result in percentage if and only if image will match with fractured image i. e. input x-ray image.

5. If image will not matched then no fractured will be detected. 6. Then final step is stop.

**2.1 Data collection and preprocessing**

Collect a dataset of X-ray images with labeled fracture and non-fracture

images. This dataset can be obtained from publicly available sources like the

NIH Chest X-ray dataset or the MURA dataset.

Preprocess the images by resizing them to a fixed size, normalizing the pixel

values, and applying data augmentation techniques like random cropping and

flipping.

**2.1.1 Fine-tuning ResNet-50**

Load the pre-trained ResNet-50 model from a deep learning library like Keras

or PyTorch.

Replace the last fully connected layer of the model with a new layer that has

the same number of output units as the number of classes in the bone

fracture dataset (i.e., two for fracture and non-fracture).

Freeze the weights of all the layers except the new fully connected layer.

Train the model on the bone fracture dataset, using a transfer learning

approach. Use a suitable optimizer like Adam and a suitable loss function like

binary cross-entropy.

**2.1.2 Training and evaluation**

Split the bone fracture dataset into training, validation, and testing sets.

Train the fine-tuned ResNet-50 model on the training set, and evaluate it on

the validation set.

Fine-tune the model based on the performance on the validation set, using

techniques like early stopping to prevent overfitting.

Test the final model on the testing set, and evaluate its performance using

metrics like accuracy, precision, recall, and F1 score.

**2.1.3 Building the GUI**

Choose a GUI library like PyQt or Tkinter.

5Build 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[1].

Display the results on the GUI interface, including the uploaded image, the

classification result, and an alert message if a fracture is detected.

Integration:Integrate the fine-tuned ResNet-50 model with the GUI interface.

When a user uploads an X-ray image, preprocess the image and pass it

through the model.

Display the results on the GUI interface, including the uploaded image, the

classification result, and an alert message if a fracture is detected.

**2.2 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.

The given image is resampled to 256 X 256 by using the colour plane extraction in vision Assistant software. Resampling of the image is a technique used in order to convert the original image to an image with different width and/or height. Let G(x, y) be the original image and N(x, y) be the predefined resolution of the sampled image. Then the corresponding relation is given as:

**N(x, y) = Height\*width=256\*256 pixels**.

1. The images that undergo resampling yield images having fixed geometrical dimensions with the resolution being the same. Histogram: Histogram Analysis is then done on the resampled image. Histogram is the graphical representation of pixel intensity (taken on x-Axis) versus number of pixels (taken on y-Axis). Let us consider 8-bit grayscale image. For this image 256 different intensities are possible and the histogram of this image will display 256 numbers graphically representing the distribution of pixels among these values only. Laplacian-Guassian filter: To find regions of rapid change in images derivative filters like laplacian filters are used. Before applying this filter it is important to smooth the image as laplacian filter is sensitive to noise. This is done by using Guassian filter. Hence forth the image undergoes a two-step process called as Laplacian of Guassian operation and the mathematical equation is given as

In morphological image processing erode function is one of the basic operations that can be applied to both binary and as well as gray scale images. Gray -scale erosion of a point is defined as the minimum number of points in its neighbourhood that define the structuring element. If a(x) denotes an image, b(x) is the grayscale structuring element, the grayscale erosion of a by b is mathematically represented as

**2.2.1 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.

**2.2.2 Model evaluation**

Split the dataset into training and validation sets.

6Train the model on the training set and evaluate its performance on the

validation set.

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

**2.2.3 Testing**

Test the final model on a separate test dataset to evaluate its performance.

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.

**2.3 System test**

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.

7Compare the predicted fracture status with the actual fracture status for each

image in the test dataset.

Calculate the accuracy, precision, recall, and F1-score of the model on the

test dataset.

For example, we can use the following code snippet to perform the system

test:

from tensorflow.keras.models import load\_model

from sklearn.metrics import confusion\_matrix, accuracy\_score,

precision\_score, recall\_score, f1\_score

# Load the trained model

model = load\_model('bone\_fracture\_detection\_model.h5')

# Load the test dataset

test\_data = ...

# Preprocess the test images

test\_images = preprocess\_images(test\_data)

# Predict the fracture status of each test image

predictions = model. Predict(test\_images)

# Convert the predicted probabilities to class labels

predicted\_labels = np.argmax(predictions, axis=1)

# Extract the actual labels from the test dataset

actual\_labels = test\_data['fracture\_status'].values

# Calculate the performance metrics

accuracy = accuracy\_score(actual\_labels, predicted\_labels)

precision = precision\_score(actual\_labels, predicted\_labels)

recall = recall\_score(actual\_labels, predicted\_labels)

8f1 = f1\_score(actual\_labels, predicted\_labels)

confusion\_mat = confusion\_matrix(actual\_labels, predicted\_labels)

That requires timely diagnosis and treatment to avoid complications and

ensure proper healing. However, the diagnosis of bone fractures is often

challenging, requiring experienced radiologists to examine X-ray images and

make a diagnosis. The process can be time-consuming and prone to errors,

particularly in cases where the fractures.

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.

**2.4 Problem statement**

The problem statement for bone fracture detection using the ResNet-50

algorithm in Python is to develop a deep learning model that can accurately

classify X-ray images of bones as either fractured or non-fractured.

Bone fractures are a common medical condition are subtle or located in

complex areas of the bone.

The development of an accurate and automated bone fracture detection

system can help improve the speed and accuracy of fracture diagnosis,

enabling medical professionals to make faster and more informed treatment

decisions. The ResNet-50 algorithm is a powerful deep learning algorithm that

has shown promising results in various computer vision tasks, including

image classification. By leveraging the power of ResNet-50 and training it on

a large dataset of bone X-ray images, we can develop a robust bone fracture

detection model that can accurately diagnose bone fractures and improve

patient outcomes.

**2.5 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:

**Data collection**: Collect a large dataset of bone X-ray images that includes

both fractured and non-fractured images.

**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.**

**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].

**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.

**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.

**2.6 Dataset**

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:

10| \*\*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.

**2.7 CNN (CONVOLUTION NEURAL NETWORK)**

There are different types of Neural networks, but CNN is widely used for the

fractured images. The main advantage of CNN when compared with other

networks is that it automatically detects the important features without any

human supervision. It is very efficient in computational.In this paper, CNN

(convolution neural network) is used, which is used to divide into two

segments i.e., normal and abnormal, if it is the normal case it doesn’t go to

the segmentation stage and shows not affected and the process gets stopped.

If it is the abnormal case it goes to further classification. Neural networks can

be described in layman’s term as an intelligent assembly line where the input

data enters and the network figures out the needed adjustments and provides

a reasonably accurate conclusion based on the data as the output[4]. The

operations performed in the network are organized into a multilayered feed

forward network.

The main layers are listed below.

They are

1.Initial Input layer

2. Hidden layer 1

3. Pattern layer / Hidden layer 2

4. Output layer

Each input layer will pass through the series of convolution layer. Here

hidden layer is dataset. CNN has two inputs one is hidden layer and other is

input layer which compares and gives the image whether it is normal or

11abnormal. If abnormal it calculates stages depending upon the area. The

structure of the network is shown in the Figure 2.3.

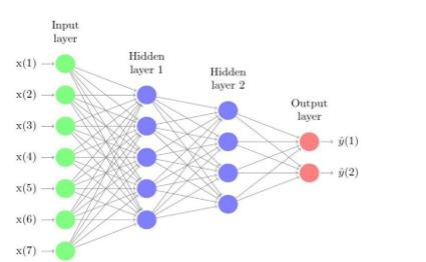
A convolutional neural network (CNN or convnet) is a subset of machine

learning. It is one of the various types of artificial neural networks which are

used for different applications and data types[5]. A CNN is a kind of network

architecture for deep learning algorithms and is specifically used for image

recognition and tasks that involve the processing of pixel data.



**Figure 2.3:** Structure of the network

There are other types of neural networks in deep learning, but for identifying

and recognizing objects, CNNs are the network architecture of choice. This

makes them highly suitable for computer vision (CV) tasks and for

applications where object recognition is vital, such as self-driving cars and

facial recognition.

**2.7.1 How convolutional neural networks work**

A CNN can have multiple layers, each of which learns to detect the different

features of an input image. A filter or kernel is applied to each image to

produce an output that gets progressively better and more detailed after

each layer. In the lower layers, the filters can start as simple features.

12At each successive layer, the filters increase in complexity to check and

identify features that uniquely represent the input object. Thus, the output

of each convolved image -- the partially recognized image after each layer --

becomes the input for the next layer. In the last layer, which is an FC layer,

the CNN recognizes the image or the object it represents.

With convolution, the input image goes through a set of these filters. As

each filter activates certain features from the image, it does its work and

passes on its output to the filter in the next layer. Each layer learns to

identify different features and the operations end up being repeated for

dozens, hundreds or even thousands of layers. Finally, all the image data

progressing through the CNN's multiple layers allow the CNN to identify

the entire object.

**2.7.2 Applications of convolutional neural networks**

Convolutional neural networks are already used in a variety of CV and image

recognition applications. Unlike simple image recognition applications, CV

enables computing systems to also extract meaningful information from

visual inputs (e.g., digital images) and then take appropriate action based on

this information[6].

The most common applications of CV and CNNs are used in fields such as the

following:

**Healthcare:** CNNs can examine thousands of visual reports to detect any

anomalous conditions in patients, such as the presence of malignant cancer

cells.

**Automotive:** CNN technology is powering research into autonomous vehicles

and self-driving cars.

**Social media:** Social media platforms use CNNs to identify people in a user's

photograph and help the user tag their friends.

**Retail:** E-commerce platforms that incorporate visual search allow brands to

recommend items that are likely to appeal to a shopper.

13**Facial recognition for law enforcement:** Generative adversarial networks

(GANs) are used to produce new images that can then be used to train deep

learning models for facial recognition

**Audio processing for virtual assistants:** CNNs in virtual assistants learn

and detect user-spoken keywords and process the input to guide their actions

and respond to the user.

**2.8 RestNet50 Model**

ResNet stands for Residual Network and is a specific type of convolutional

neural network (CNN) introduced in the 2015 paper “Deep Residual Learning

for Image Recognition” by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and

Sun Jian. CNNs are commonly used to power computer vision applications.

ResNet-50 is a 50-layer convolutional neural network (48 convolutional

layers, one MaxPool layer, and one average pool layer). Residual neural

networks are a type of artificial neural network (ANN) that forms networks by

stacking residual blocks.

**2.8.1 Why is ResNet so popular?**

This model was immensely successful, as can be ascertained from the fact

that its ensemble won the top position at the ILSVRC 2015 classification

competition with an error of only 3.57%. Additionally, it also came first in the

ImageNet detection, ImageNet localization, COCO detection, and COCO

segmentation in the ILSVRC & COCO competitions of 2015.

ResNet-50 Architecture While the Resnet50 architecture is based on the above

model, there is one major difference. In this case, the building block was

modified into a bottleneck design due to concerns over the time taken to train

the layers. This used a stack of 3 layers instead of the earlier 2[10]. Therefore,

each of the 2-layer blocks in Resnet34 was replaced with a 3-layer bottleneck

block, forming the Resnet 50 architecture. This has much higher accuracy

than the 34-layer ResNet model. The 50-layer ResNet achieves a performance

of 3.8 bn FLOPS

14ResNet50 With Keras Keras is a deep learning API that is popular due to the

simplicity of building models using it. Keras comes with several pre-trained

models, including Resnet50, that anyone can use for their experiments.

Therefore, building a residual network in Keras for computer vision tasks like

image classification is relatively simple. You only need to follow a few simple

steps

What is Deep Residual Learning used for? ResNet was created with the aim of

tackling this exact problem. Deep residual nets make use of residual blocks

to improve the accuracy of the models. The concept of “skip connections,”

which lies at the core of the residual blocks, is the strength of this type of

neural network.

**2.9 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 72%

training, 18% validation and 10% test. The algorithm starts with data

augmentation 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. 15

**CHAPTER 3**

**SOFTWARE DEVELOPMENT**

In this chapter, the bone fracture detection using IDLE software is

explained.

**3.1 Why choose Python**

If you’re going to write programs, there are literally dozens of commonly used

languages to choose from. Why choose Python? Here are some of the features

that make Python an appealing choice.

**3.1.1 Python is Popular**

Python has been growing in popularity over the last few years. The 2018 Stack

Overflow Developer Survey ranked Python as the 7th most popular and the

number one most wanted technology of the year. World-class software

development countries around the globe use Python every single day.

According to research by Dice Python is also one of the hottest skills to have

and the most popular programming language in the world based on the

Popularity of Programming Language Index.

Due to the popularity and widespread use of Python as a programming

language, Python developers are sought after and paid well. If you’d like to dig

deeper into Python salary statistics and job opportunities, you can do so here.

**3.1.2 Python is interpreted**

Many languages are compiled, meaning the source code you create needs to

be translated into machine code, the language of your computer’s processor,

before it can be run. Programs written in an interpreted language are passed

straight to an interpreter that runs them directly.

This makes for a quicker development cycle because you just type in your

code and run it, without the intermediate compilation step.

One potential downside to interpreted languages is execution speed. Programs

that are compiled into the native language of the computer processor tend to

16run more quickly than interpreted programs. For some applications that are

particularly computationally intensive, like graphics processing or intense

number crunching, this can be limiting[8].

In practice, however, for most programs, the difference in execution speed is

measured in milliseconds, or seconds at most, and not appreciably noticeable

to a human user. The expediency of coding in an interpreted language is

typically worth it for most applications.

**3.1.3 Python is Free**

The Python interpreter is developed under an OSI-approved open-source

license, making it free to install, use, and distribute, even for commercial

purposes.

A version of the interpreter is available for virtually any platform there is,

including all flavors of Unix, Windows, macOS, smart phones and tablets, and

probably anything else you ever heard of. A version even exists for the half

dozen people remaining who use OS/2.

**3.1.4 Python is Portable**

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. (This is true of any interpreted

language, not just Python.)

**3.1.5 Python is Simple**

As programming languages go, Python is relatively uncluttered, and the

developers have deliberately kept it that way.

A rough estimate of the complexity of a language can be gleaned from the

number of keywords or reserved words in the language. These are words that

are reserved for special meaning by the compiler or interpreter because they

designate specific built-in functionality of the language.

Python 3 has 33 keywords, and Python 2 has 31. By contrast, C++ has 62,

Java has 53, and Visual Basic has more than 120, though these latter

17examples probably vary somewhat by implementation or dialect.

Python code has a simple and clean structure that is easy to learn and easy

to read. In fact, as you will see, the language definition enforces code structure

that is easy to read.

But It’s Not That Simple For all its syntactical simplicity, Python supports

most constructs that would be expected in a very high-level language,

including complex dynamic data types, structured and functional

programming, and object-oriented programming.

Additionally, a very extensive library of classes and functions is available that

provides capability well beyond what is built into the language, such as

database manipulation or GUI programming.

Python accomplishes what many programming languages don’t: the language

itself is simply designed, but it is very versatile in terms of what you can

accomplish with it.

**Conclusion**

This section gave an overview of the **Python** programming language,

including:

• A brief history of the development of Python

• Some reasons why you might select Python as your language of choice

Python is a great option, whether you are a beginning programmer looking to

learn the basics, an experienced programmer designing a large application, or

anywhere in between. The basics of Python are easily grasped, and yet its

capabilities are vast. Proceed to the next section to learn how to acquire and

install Python on your computer.

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.

Python is an interpreted language. Interpreted languages do not need to be

compiled to run. A program called an interpreter runs Python code on almost

18any kind of computer. This means that a programmer can change the code

and quickly see the results. This also means Python is slower than a compiled

language like C, because it is not running machine code directly.

Python is a good programming language for beginners. It is a high-level

language, which means a programmer can focus on what to do instead of how

to do it. Writing programs in Python takes less time than in some other

languages.

Python drew inspiration from other programming languages like C, Java, Perl,

and Lisp.

Python has a very easy-to-read syntax. Some of Python's syntax comes from

C, because that is the language that Python was written in. But Python uses

whitespace to delimit code: spaces or tabs are used to organize code into

groups. This is different from C. In C, there is a semicolon at the end of each

line and curly braces ({}) are used to group code. Using whitespace to delimit

code makes Python a very easy-to-read language.

**3.1.6 Python use [change / change source]**

Python is used by hundreds of thousands of programmers and is used in

many

places. Sometimes only Python code is used for a program, but most of the

time it is used to do simple jobs while another programming language is used

to do more complicated tasks[7].

Its standard library is made up of many functions that come with Python

when it is installed. On the Internet there are many other libraries available

that make it possible for the Python language to do more things. These

libraries make it a powerful language; it can do many different things.

Some things that Python is often used for are:

• Web development

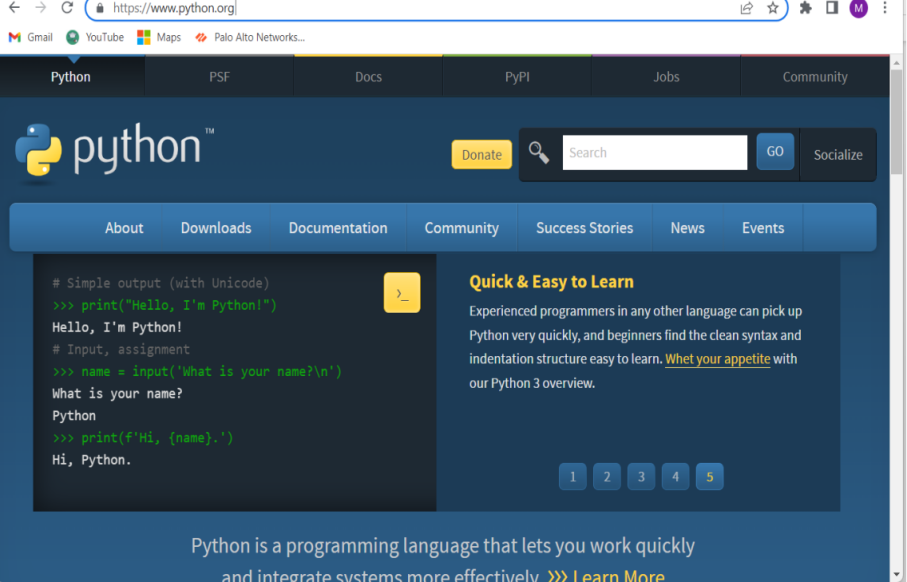
• Scientific programming

• Desktop GUIs

19• Desktop applications

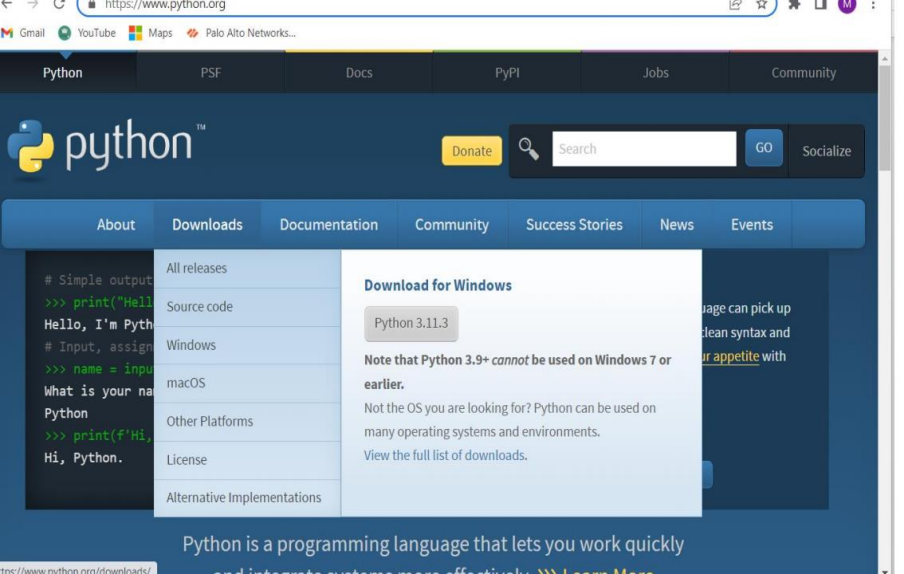
**3.1.7 Python installation**

**Step 1:** Search python.org



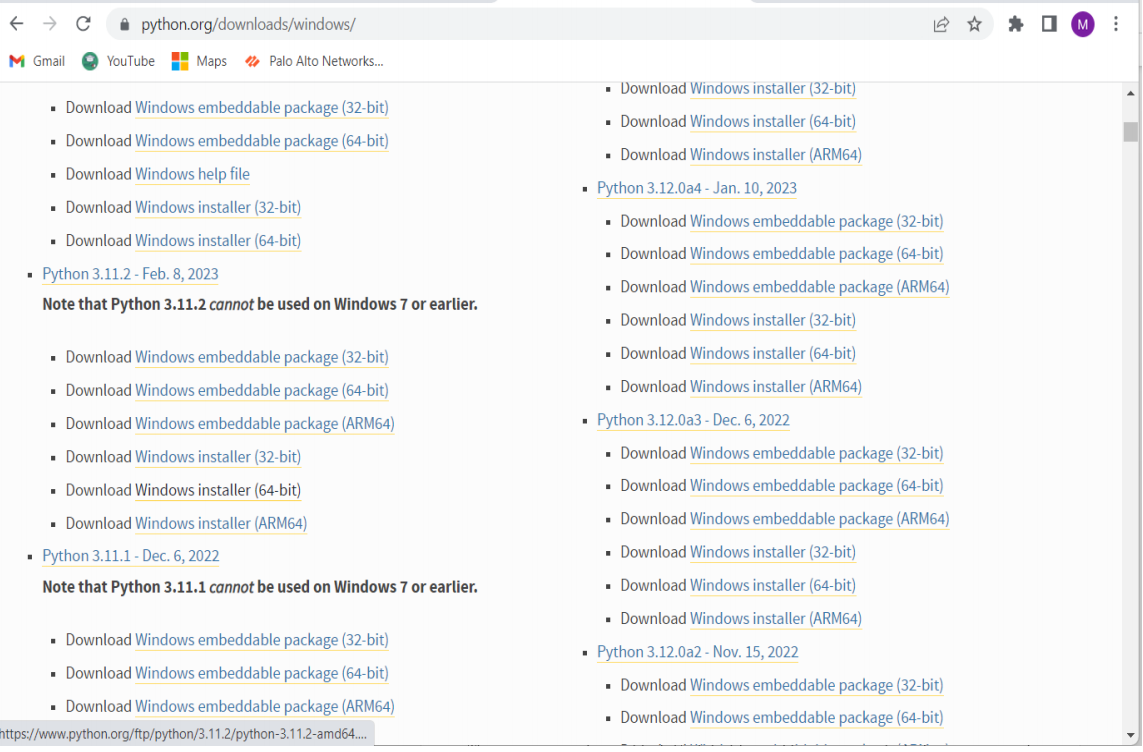
**Figure 3.1:** search python.org

**Step 2:** Go to downloads and select window



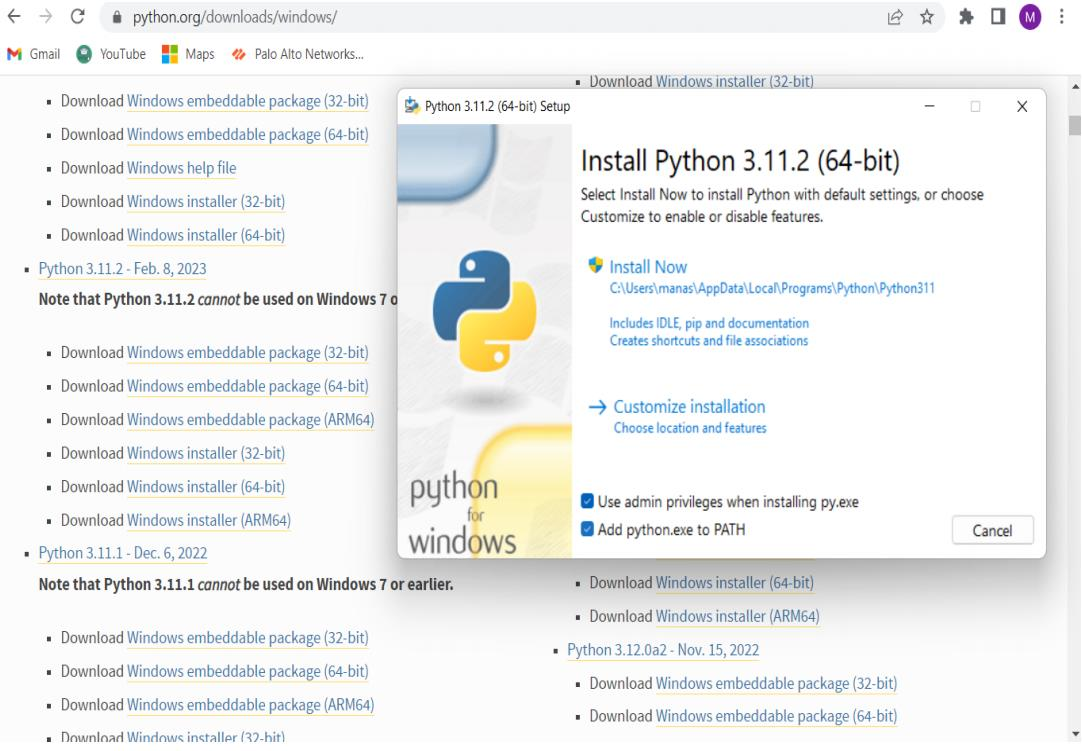
**Figure 3.2:** Go to downloads and select window

**Step 3:** Download Windows installer(64-bit)



**Figure 3.3:** Download Windows installer(64-bit)

**Step 4:** Now select python.exe to path and install the IDLE



**Figure 3.4:** Now select python.exe to path and install the IDLE**3.2**

**MODULES**

\* customtkinter~=5.0.3

\* PyAutoGUI~=0.9.53

\* PyGetWindow~=0.0.9

\* Pillow~=8.4.0

\* numpy~=1.19.5

\* tensorflow~=2.6.2

\* keras~=2.6.0

\* pandas~=1.1.5

\* matplotlib~=3.3.4

\* scikit-learn~=0.24.2

\* colorama~=0.4

**CustomTkinter:** CustomTkinter is a python UI-library based on

Tkinter, which provides new, modern and fully customizable

widgets. They are created and used like normal Tkinter widgets

and can also be used in combination with normal Tkinter elements.

The widgets and the window colors either adapt to the system

appearance or the manually set mode ('light', 'dark'), and all

CustomTkinter widgets and windows support HighDPI scaling

(Windows, macOS). With CustomTkinter you'll get a consistent and

modern look across all desktop platforms (Windows, macOS,

Linux).

**PyAutoGUI:** PyAutoGUI is a cross-platform GUI automation

Python modulefor human beings. Used to programmatically

control the mouse & keyboard. pip install pyautogui.

**PyGetWindow:** A simple, cross-platform module for

obtaining GUIinformation on and controlling application's

windows.

2223

**Pillow:** It allows you to pull some statistics data out of image using

histogram method, which later can be used for statistical analysis

and automatic contrast enhancement.

**Numpy:** It is used to perform a wide variety of mathematical

operations on arrays. It adds powerful data structures to Python

that guarantee efficient calculations with arrays and matrices and

it supplies an enormous library of high-level mathematical

functions that operate on these arrays and matrices.

**Tensorflow:** It helps you implement best practices for data automation,

model tracking, performance monitoring, and model retraining.

**Keras:** creating deep models which can be productized on

smartphones.Keras is also used for distributed training of deep

learning models.

**Pandas:** It is a Python library used for working with data sets. It

has functionsfor analyzing, cleaning, exploring, and

manipulating data. The name "Pandas" has a reference to both

"Panel Data", and "Python Data Analysis" and was created by Wes

McKinney in 2008.

**Matplotlib:** It is a cross-platform, data visualization and graphical

plotting library for Python and its numerical extension NumPy.

**CHAPTER 4**

**CODE IMPLEMENTATION**

In this chapter, the source code for implementing the bone fracture

detection using CNN-RESNET50 is elaborated.

**4.1 Code**

**4.1.1 Main gui.py**

import os

from tkinter import filedialog

import customtkinter as ctk

import pyautogui

import pygetwindow

from PIL import ImageTk, Image

from predictions import predict

# global variables

project\_folder = os.path.dirname(os.path.abspath(\_\_file\_\_))

folder\_path = project\_folder + '/images/'

filename = ""

class App(ctk.CTk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Bone Fracture Detection")

self.geometry(f"{500}x{740}")

self.head\_frame = ctk.CTkFrame(master=self)

self.head\_frame. Pack(pady=20, padx=60, fill="both", expand=True)

self.main\_frame = ctk.CTkFrame(master=self)

24self.main\_frame.pack(pady=20, padx=60, fill="both", expand=True)

self.head\_label = ctk.CTkLabel(master=self.head\_frame, text="Bone Fracture

Detection",

font=(ctk.CTkFont("Roboto"), 28))

self.head\_label.pack(pady=20, padx=10, anchor="nw", side="left")

img1 = ctk.CTkImage(Image.open(folder\_path + "info.png"))

self.img\_label = ctk.CTkButton(master=self.head\_frame, text="", image=img1,

command=self.open\_image\_window,

width=40, height=40)

self.img\_label.pack(pady=10, padx=10, anchor="nw", side="right")

self.info\_label = ctk.CTkLabel(master=self.main\_frame,

text="Bone fracture detection system, upload an x-ray image for fracture

detection.",

wraplength=300, font=(ctk.CTkFont("Roboto"), 18))

self.info\_label.pack(pady=10, padx=10)

self.upload\_btn = ctk.CTkButton(master=self.main\_frame, text="Upload

Image", command=self.upload\_image)

self.upload\_btn.pack(pady=0, padx=1)

self.frame2 = ctk.CTkFrame(master=self.main\_frame, fg\_color="transparent",

width=256, height=256)

self.frame2.pack(pady=10, padx=1)

img = Image.open(folder\_path + "Question\_Mark.jpg")

img\_resized = img.resize((int(256 / img.height \* img.width), 256)) # new width

& height

img = ImageTk.PhotoImage(img\_resized)

self.img\_label = ctk.CTkLabel(master=self.frame2, text="", image=img)

self.img\_label.pack(pady=1, padx=10)

self.predict\_btn = ctk.CTkButton(master=self.main\_frame, text="Predict",

25command=self.predict\_gui)

self.predict\_btn.pack(pady=0, padx=1)

self.result\_frame = ctk.CTkFrame(master=self.main\_frame,

fg\_color="transparent", width=200, height=100)

self.result\_frame.pack(pady=5, padx=5)

self.loader\_label = ctk.CTkLabel(master=self.main\_frame, width=100,

height=100, text="")

self.loader\_label.pack(pady=3, padx=3)

self.res1\_label = ctk.CTkLabel(master=self.result\_frame, text="")

self.res1\_label.pack(pady=5, padx=20)

self.res2\_label = ctk.CTkLabel(master=self.result\_frame, text="")

self.res2\_label.pack(pady=5, padx=20)

self.save\_btn = ctk.CTkButton(master=self.result\_frame, text="Save Result",

command=self.save\_result)

self.save\_label = ctk.CTkLabel(master=self.result\_frame, text="")

def upload\_image(self):

global filename

f\_types = [("All Files", "\*.\*")]

filename = filedialog.askopenfilename(filetypes=f\_types,

initialdir=project\_folder+'/test/Wrist/')

self.save\_label.configure(text="")

self.res2\_label.configure(text="")

self.res1\_label.configure(text="")

self.img\_label.configure(self.frame2, text="", image="")

img = Image.open(filename)

img\_resized = img.resize((int(256 / img.height \* img.width), 256)) # new width

& height

26img = ImageTk.PhotoImage(img\_resized)

self.img\_label.configure(self.frame2, image=img, text="")

self.img\_label.image = img

self.save\_btn.pack\_forget()

self.save\_label.pack\_forget()

def predict\_gui(self):

global filename

bone\_type\_result = predict(filename)

result = predict(filename, bone\_type\_result)

print(result)

if result == 'fractured':

self.res2\_label.configure(text\_color="RED", text="Result: Fractured",

font=(ctk.CTkFont("Roboto"), 24))

else:

self.res2\_label.configure(text\_color="GREEN", text="Result: Normal",

font=(ctk.CTkFont("Roboto"), 24))

bone\_type\_result = predict(filename, "Parts")

self.res1\_label.configure(text="Type: " + bone\_type\_result,

font=(ctk.CTkFont("Roboto"), 24))

print(bone\_type\_result)

self.save\_btn.pack(pady=10, padx=1)

self.save\_label.pack(pady=5, padx=20)

def save\_result(self):

tempdir = filedialog.asksaveasfilename(parent=self, initialdir=project\_folder +

'/PredictResults/',

title='Please select a directory and filename', defaultextension=".png")

screenshots\_dir = tempdir

27window = pygetwindow.getWindowsWithTitle('Bone Fracture Detection')[0]

left, top = window.topleft

right, bottom = window.bottomright

pyautogui.screenshot(screenshots\_dir)

im = Image.open(screenshots\_dir)

im = im.crop((left + 10, top + 35, right - 10, bottom - 10))

im.save(screenshots\_dir)

self.save\_label.configure(text\_color="WHITE", text="Saved!",

font=(ctk.CTkFont("Roboto"), 16))

def open\_image\_window(self):

im = Image.open(folder\_path + "rules.jpeg")

im = im.resize((700, 700))

im.show()

if \_\_name\_\_ == "\_\_main\_\_":

app = App()

app.mainloop()

**4.1.2 Prediction test.py**

import os

from colorama import Fore

from predictions import predict

# load images to predict from paths

# .... / elbow1.jpg

# Hand fractured -- elbow2.png

# / / \ .....

# test - Elbow ------

# \ \ / elbow1.png

28# Shoulder normal -- elbow2.jpg

# .... \

#

def load\_path(path):

dataset = []

for body in os.listdir(path):

body\_part = body

path\_p = path + '/' + str(body)

for lab in os.listdir(path\_p):

label = lab

path\_l = path\_p + '/' + str(lab)

for img in os.listdir(path\_l):

img\_path = path\_l + '/' + str(img)

dataset.append(

{

'body\_part': body\_part,

'label': label,

'image\_path': img\_path,

'image\_name': img

}

)

return dataset

categories\_parts = ["Elbow", "Hand", "Shoulder"]

categories\_fracture = ['fractured', 'normal']

def reportPredict(dataset):

29total\_count = 0

part\_count = 0

status\_count = 0

print(Fore.YELLOW +

'{0: <28}'.format('Name') +

'{0: <14}'.format('Part') +

'{0: <20}'.format('Predicted Part') +

'{0: <20}'.format('Status') +

'{0: <20}'.format('Predicted Status'))

for img in dataset:

body\_part\_predict = predict(img['image\_path'])

fracture\_predict = predict(img['image\_path'], body\_part\_predict)

if img['body\_part'] == body\_part\_predict:

part\_count = part\_count + 1

if img['label'] == fracture\_predict:

status\_count = status\_count + 1

color = Fore.GREEN

else:

color = Fore.RED

print(color +

'{0: <28}'.format(img['image\_name']) +

'{0: <14}'.format(img['body\_part']) +

'{0: <20}'.format(body\_part\_predict) +

'{0: <20}'.format((img['label'])) +

'{0: <20}'.format(fracture\_predict))

30print(Fore.BLUE + '\npart acc: ' + str("%.2f" % (part\_count / len(dataset) \*

100)) + '%')

print(Fore.BLUE + 'status acc: ' + str("%.2f" % (status\_count / len(dataset)

\* 100)) + '%')

return

THIS\_FOLDER = os.path.dirname(os.path.abspath(\_\_file\_\_))

test\_dir = THIS\_FOLDER + '/test/'

reportPredict(load\_path(test\_dir))

**4.1.3 Prediction.py**

import numpy as np

import tensorflow as tf

from keras.preprocessing import image

# load the models when import "predictions.py"

model\_elbow\_frac =

tf.keras.models.load\_model("weights/ResNet50\_Elbow\_frac.h5")

model\_hand\_frac =

tf.keras.models.load\_model("weights/ResNet50\_Hand\_frac.h5")

model\_shoulder\_frac =

tf.keras.models.load\_model("weights/ResNet50\_Shoulder\_frac.h5")

model\_parts =

tf.keras.models.load\_model("weights/ResNet50\_BodyParts.h5")

# categories for each result by index

# 0-Elbow 1-Hand 2-Shoulder

categories\_parts = ["Elbow", "Hand", "Shoulder"]

# 0-fractured 1-normal

categories\_fracture = ['fractured', 'normal']

# get image and model name, the default model is "Parts"

# Parts - bone type predict model of 3 classes

3132

# otherwise - fracture predict for each part

def predict(img, model="Parts"):

size = 224

if model == 'Parts':

chosen\_model = model\_parts

else:

if model == 'Elbow':

chosen\_model = model\_elbow\_frac

elif model == 'Hand':

chosen\_model = model\_hand\_frac

elif model == 'Shoulder':

chosen\_model = model\_shoulder\_frac

# load image with 224px224p (the training model image size, rgb)

temp\_img = image.load\_img(img, target\_size=(size, size))

x = image.img\_to\_array(temp\_img)

x = np.expand\_dims(x, axis=0)

images = np.vstack([x])

prediction = np.argmax(chosen\_model.predict(images), axis=1)

# chose the category and get the string prediction

if model == 'Parts':

prediction\_str = categories\_parts[prediction.item()]

else:

prediction\_str = categories\_fracture[prediction.item()]

return prediction\_str

**4.1.4 Training set.py**

import numpy as npimport pandas as pd

import os.path

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

from tensorflow.keras.optimizers import Adam

# load images to build and train the model

# .... / img1.jpg

# test Hand patient0000 positive -- img2.png

# / / \ .....

# Dataset - Elbow ------ patient0001

# \ train \ / img1.png

# Shoulder patient0002 negative -- img2.jpg

# .... \

#

def load\_path(path, part):

"""

load X-ray dataset

"""

dataset = []

for folder in os.listdir(path):

folder = path + '/' + str(folder)

if os.path.isdir(folder):

for body in os.listdir(folder):

if body == part:

33body\_part = body

path\_p = folder + '/' + str(body)

for id\_p in os.listdir(path\_p):

patient\_id = id\_p

path\_id = path\_p + '/' + str(id\_p)

for lab in os.listdir(path\_id):

if lab.split('\_')[-1] == 'positive':

label = 'fractured'

elif lab.split('\_')[-1] == 'negative':

label = 'normal'

path\_l = path\_id + '/' + str(lab)

for img in os.listdir(path\_l):

img\_path = path\_l + '/' + str(img)

dataset.append(

{

'body\_part': body\_part,

'patient\_id': patient\_id,

'label': label,

'image\_path': img\_path

}

)

return dataset

# this function get part and know what kind of part to train, save model and

save plots

def trainPart(part):

# categories = ['fractured', 'normal']

THIS\_FOLDER = os.path.dirname(os.path.abspath(\_\_file\_\_))

34image\_dir = THIS\_FOLDER + '/Dataset/'

data = load\_path(image\_dir, part)

labels = []

filepaths = []

# add labels for dataframe for each category 0-fractured, 1- normal

for row in data:

labels.append(row['label'])

filepaths.append(row['image\_path']

filepaths = pd.Series(filepaths, name='Filepath').astype(str)

labels = pd.Series(labels, name='Label')

images = pd.concat([filepaths, labels], axis=1)

# split all dataset 10% test, 90% train (after that the 90% train will split to

20% validation and 80% train

train\_df, test\_df = train\_test\_split(images, train\_size=0.9, shuffle=True,

random\_state=1)

# each generator to process and convert the filepaths into image arrays,

# and the labels into one-hot encoded labels.

# The resulting generators can then be used to train and evaluate a deep

learning model.

# now we have 10% test, 72% training and 18% validation

train\_generator =

tf.keras.preprocessing.image.ImageDataGenerator(horizontal\_flip=True,

preprocessing\_function=tf.keras.applications.resnet50.preprocess\_input,

validation\_split=0.2)

# use ResNet50 architecture

test\_generator = tf.keras.preprocessing.image.ImageDataGenerator(

preprocessing\_function=tf.keras.applications.resnet50.preprocess\_input)

35train\_images = train\_generator.flow\_from\_dataframe(

dataframe=train\_df,

x\_col='Filepath',

y\_col='Label',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='categorical',

batch\_size=64,

shuffle=True,

seed=42,

subset='training'

)

val\_images = train\_generator.flow\_from\_dataframe(

dataframe=train\_df,

x\_col='Filepath',

y\_col='Label',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='categorical',

batch\_size=64,

shuffle=True,

seed=42,

subset='validation'

)

test\_images = test\_generator.flow\_from\_dataframe(

dataframe=test\_df,

36x\_col='File path',

y\_col='Label',

target\_size=(224, 224),

color\_mode='rgb',

class\_mode='categorical',

batch\_size=32,

shuffle=False

)

# we use rgb 3 channels and 224x224 pixels images, use feature extracting ,

and average pooling

pretrained\_model = tf.keras.applications.resnet50.ResNet50(

input\_shape=(224, 224, 3),

include\_top=False,

weights='imagenet',

pooling='avg')

# for faster performance

pretrained\_model.trainable = False

inputs = pretrained\_model.input

x = tf.keras.layers.Dense(128, activation='relu')(pretrained\_model.output)

x = tf.keras.layers.Dense(50, activation='relu')(x)

# outputs Dense '2' because of 2 classes, fratured and normal

outputs = tf.keras.layers.Dense(2, activation='softmax')(x)

model = tf.keras.Model(inputs, outputs)

# print(model.summary())

print("-------Training " + part + "-------")

37# Adam optimizer with low learning rate for better accuracy

model.compile(optimizer=Adam(learning\_rate=0.0001),

loss='categorical\_crossentropy', metrics=['accuracy'])

# early stop when our model is over fit or vanishing gradient, with restore best

values

callbacks = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=3,

restore\_best\_weights=True)

history = model.fit(train\_images, validation\_data=val\_images, epochs=25,

callbacks=[callbacks])

# save model to this path

model.save(THIS\_FOLDER + "/weights/ResNet50\_" + part + "\_frac.h5")

results = model.evaluate(test\_images, verbose=0)

print(part + " Results:")

print(results)

print(f"Test Accuracy: {np.round(results[1] \* 100, 2)}%")

# create plots for accuracy and save it

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

# plt.show()

figAcc = plt.gcf()

38my\_file = os.path.join(THIS\_FOLDER, "./plots/FractureDetection/" + part +

"/\_Accuracy.jpeg")

figAcc.savefig(my\_file)

plt.clf()

# create plots for loss and save it

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

# plt.show()

figAcc = plt.gcf()

my\_file = os.path.join(THIS\_FOLDER, "./plots/FractureDetection/" + part +

"/\_Loss.jpeg")

figAcc.savefig(my\_file)

plt.clf()

# run the function and create model for each parts in the array

categories\_parts = ["Elbow", "Hand", "Shoulder"]

for category in categories\_parts:

trainPart(category

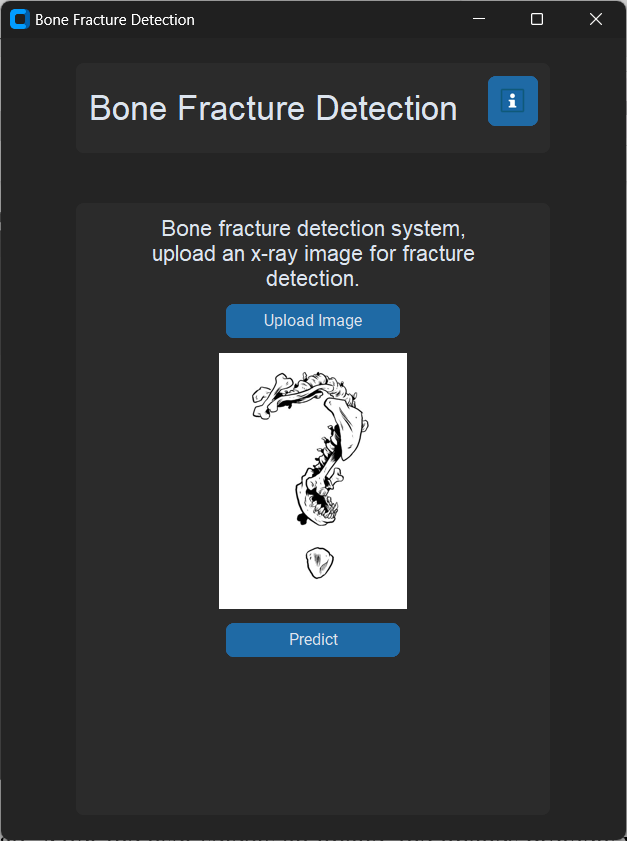
**4.2 Output**

Initialisation of GUI which contains an upload image button and predict

button and the name of the GUI is bone fracture detection.

Here we need to give an x-ray image of a patients report which are

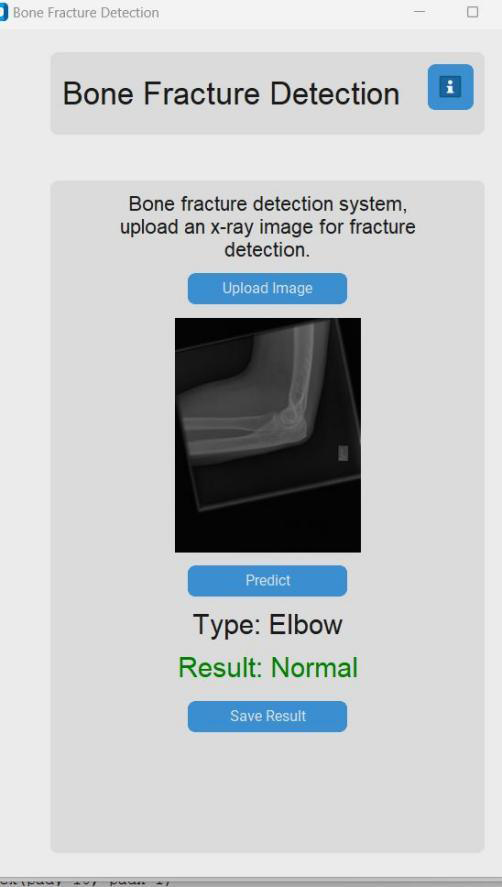
contains in a dataset which is previously defined and trained for it



.

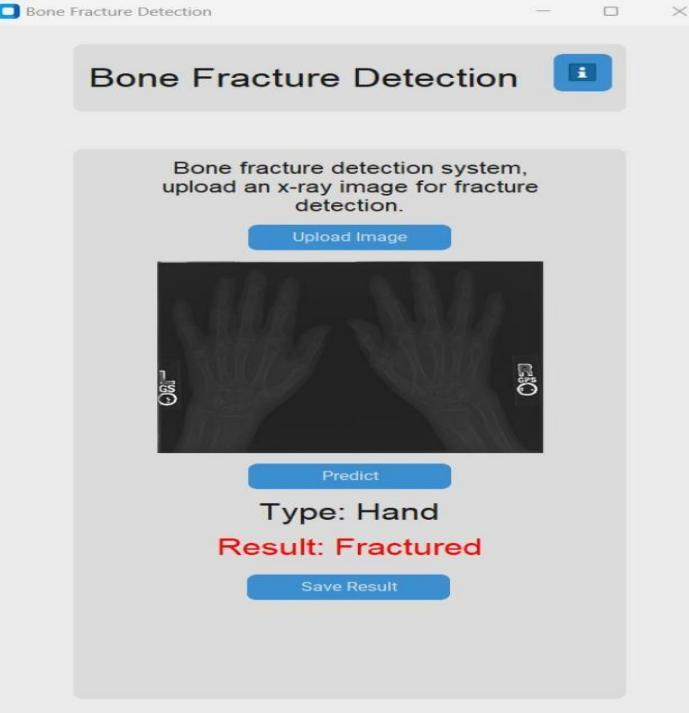
**Figure 4.1:** Processing image

In the above Figure 4.1 shows the GUI window when we run the code we get this page to upload

**Figure 4.2:** Normal Elbow

A GUI page in which we upload a x-ray image and it detects that x-ray image

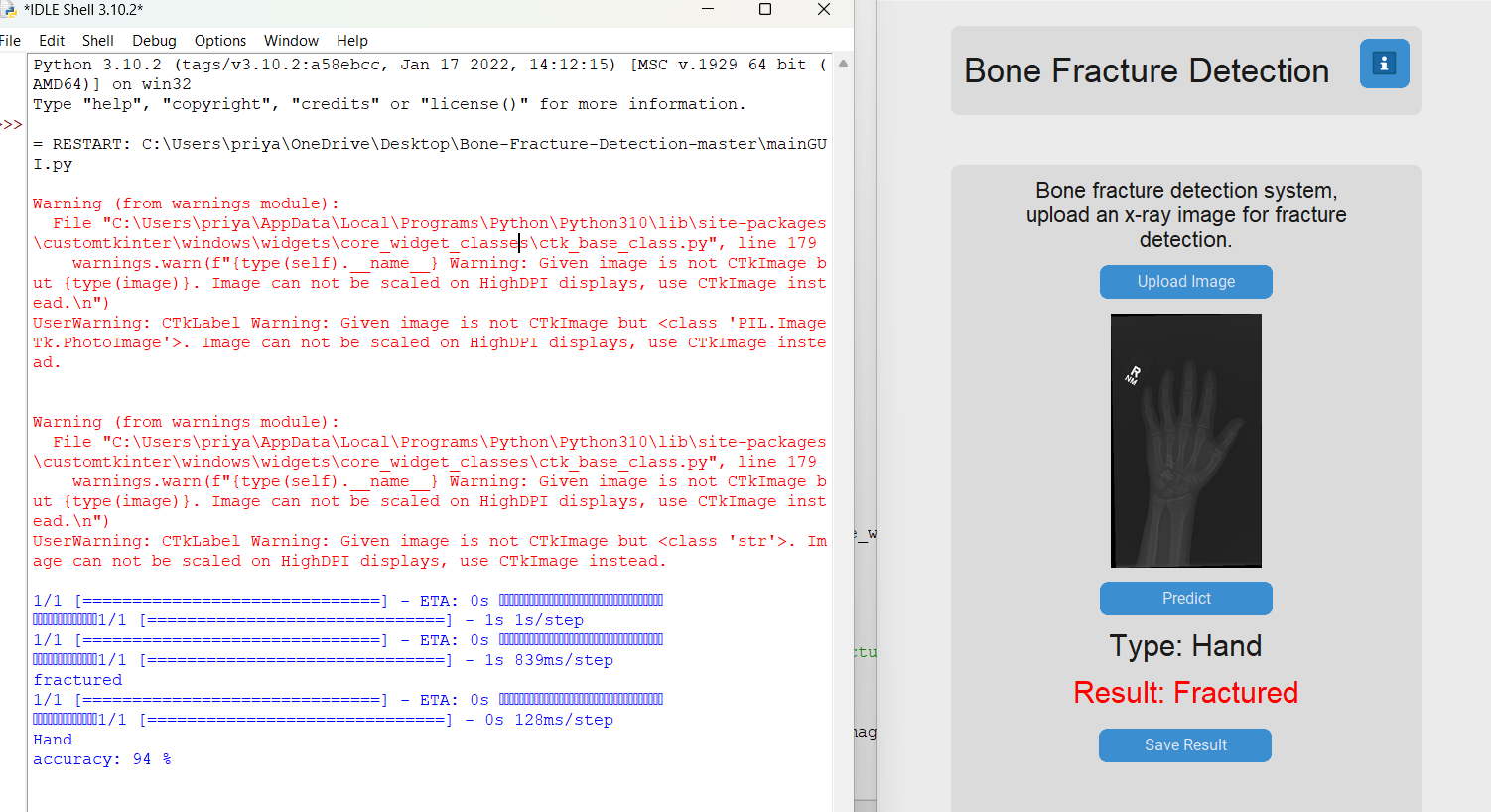
as a Type: Elbow and Result: Normal which shown in the Figure 4.2.



**Figure 4.3:** Fractured Hand

A GUI page in which we upload a x-ray image & it detects that x-ray image

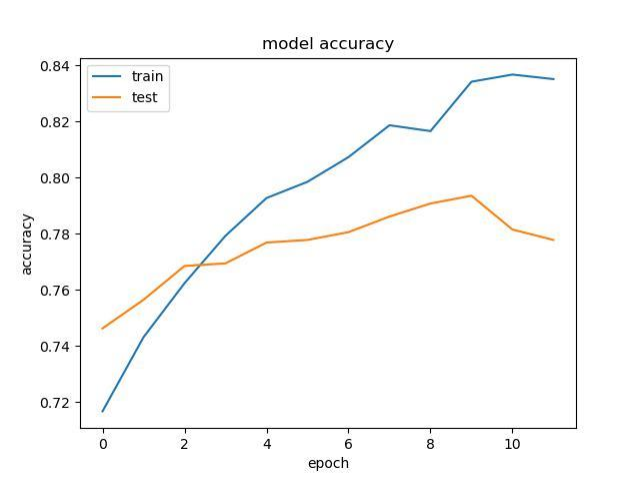
as a Type: Hand and Result: Fractured which shown in the Figure 4.3.

**4.3 ACCURACY**

**Figure 4.4:** Accuracy

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 and also give as accuracy: 94% and

which is shown in the Figure 4.4.

**Figure 4.5:** Model Accuracy

To improve the efficiency and accuracy of fracture, this is the plot for the

train and test. The orange line shows the test accuracy and the blue lines

shows the train accuracy which is shown in the Figure 4.5.43

**CONCLUSION**

In 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.

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.

**REFERENCES**

**1.** Shen, W., Zhou, M., Yang, F., Yang, C., & Tian, J. (2018). Multi-scale

Residual Network for Brain Tumor Segmentation. In Proceedings of

the IEEE conference on computer vision and pattern recognition

workshops (pp. 1175-1183).

**2.** Wang, Z., Yan, X., Smith-Miles, K., & Zhang, L. (2019). Bone fracture

detection using deep convolutional neural network and support vector

machine. IEEE Access, 7, 166032-166040.

**3.** Lakhani, P., & Sundaram, B. (2018). Deep learning at chest

radiography: automated classification of pulmonary tuberculosis by

using convolutional neural networks. Radiology, 284(2), 574-582.

**4.** Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H.,

Baxter, S. L., ... & Zhang, K. (2018). Identifying medical diagnoses

and treatable diseases by image-based deep learning. Cell, 172(5),

1122-1131.

**5.** Rajaraman, S., Antani, S. K., Poostchi, M., Silamut, K., Hossain, M.

A., Maude, R. J., ... & Jaeger, S. (2018). Pre-trained convolutional

neural networks as feature extractors toward improved malaria

parasite detection in thin blood smear images. Malaria journal, 17(1),

1-15.

**6.** Roy, S.; Whitehead, T.D.; Quirk, J.D.; Salter, A.; Ademuyiwa, F.O.; Li,

S.; An, H.; Shoghi, K.I. Optimal co-clinical radiomics: Sensitivity of

radiomic features to tumour volume, image noise and resolution in

co-clinical T1-weighted and T2-weighted magnetic resonance

imaging. eBioMedicine 2020, 59, 102963. [CrossRef] [PubMed].

**7.** Roy, S,; Bandyopadhyay, S.K. A new method of brain tissues

segmentation from MRI with accuracy estimation. Procedia Comput.

Sci. 2016, 85, 362-369.[CrossRef].

**8.** Hallas, P.; Ellingsen, T. Errors in fracture diagnoses in the emergency

department—Characteristics of patients and diurnal variation. BMC

Emerg. Med. 2006, 6, 4. [CrossRef].

**9.** Roy, S.; Whitehead, T.D.; Li, S.; Ademuyiwa, F.O.; Wahl, R.L.;

Dehdashti, F.; Shoghi, K.I. Co-clinical FDG-PET radiomic signature

in predicting response to neoadjuvant chemotherapy in triple

negative breast cancer. Eur. J. Nucl. Med. Mol. Imaging 2022, 49,

550–562. [CrossRef] [PubMED].

**10.** Krupinski, E.A.; Berbaum, K.S.; Caldwell, R.T.; Schartz, K.M.; Kim, J.

Long Radiology Workdays Reduce Detection and Accommodation

Accuracy. J. Am. Coll. Radiol. 2010, 7, 698–704. [CrossRef].

**11**. D. P. Yadav and S. Rathor, "Bone Fracture Detection and Classification using Deep Learning Approach," 2020 International Conference on Power Electronics & IoT Applications in Renewable Energy and its Control (PARC), 2020, pp. 282-285, doi: 10.1109/PARC49193.2020.236611.

**12**. Shinde Shunhangi 2, Bhor Sakshi 3, Pawale Kirti 4, P.R.Patil 5 1,2,3,4Student, SAE Pune, Maharashtra, India. 5Guide, SAE Pune, Maharashtra, India.

**13.** W. Abbas et al., "Lower Leg Bone Fracture Detection and Classification Using Faster RCNN for XRays Images," 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020, pp. 1-6, doi: 10.1109/ INMIC50486.2020.9318052.

**14**. 1Kallimpudi Bhaskara sai kiran, 2B Satyasaivani 1 PG Student, 2Associate Professor 1Department of computer science, 1GITAM (Deemed to be University), Visakhapatnam, Andhra Pradesh, India.

**15.** W. Abbas et al., "Lower Leg Bone Fracture Detection and Classification Using Faster RCNN for XRays Images," 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020, pp. 1-6, doi: 10.1109/ INMIC50486.2020.9318052.

**16.** S. Yadahalli, A. Parmar, P. Zambare and R. Sawant, "Bone Deformity Identification Using Machine Learning," 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 2021, pp. 1355-1358, doi: 10.1109/ ICICCS51141. 2021.9432272.

**17.** N. Yamamoto et al., "Boring Survey Based Fracture Detection (BSFD) for Fragility Fracture of the Pelvis in CT Images," 2021 International Conference on Machine Learning and Cybernetics (ICMLC), 2021, pp. 1-5, doi: 10.1109/ ICMLC54886. 2021. 9737242.

**18.** Boger, J. Chakalasiya, K. Christofferson, Y. Wang and J. Raiti, "Induced Acoustic Resonance for Noninvasive Bone Fracture Detection Using Digital Signal Processing and Machine Learning," 2020 IEEE Global Humanitarian Technology Conference (GHTC), 2020, pp. 1-4, doi: 10.1109/GHTC46280.2020.9342913.

**19.** Saranya and K. Kottilingam, "A Survey on Bone Fracture Identification Techniques using Quantitative and Learning Based Algorithms," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 241-248, doi: 10.1109/ICAIS50930.2021.9395817.

**20.** S. M. N. Fathima, R. Tamilselvi, M. P. Beham and A. Nagaraj, "A Deep Learning Approach on Segmentation of Bone for BMD Measurement from DEXA Scan Images," 2020 Sixth International Conference on Bio Signals, Images, and Instrumentation (ICBSII), 2020, pp. 1-5, doi: 10.1109/ ICBSII49132.2020.9167573.