LIST OF FIGURES

LIST OF ABBREVATIONS

S.NO.	ABBREVATION	FULL FORM
1.	CNN	Convolutional Neural Network
2.	GUI	Graphical User Interface
3.	ResNet	Residual Network

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.

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.

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.

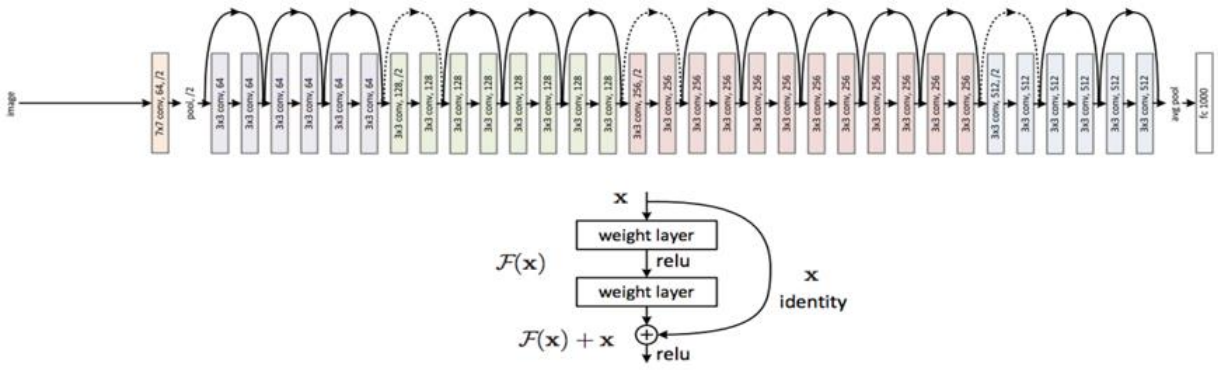

Figure 2.1: Residual Networks(ResNet50)

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

points operations. It is a widely used ResNet model and we have explored **ResNet50 architecture** in depth.

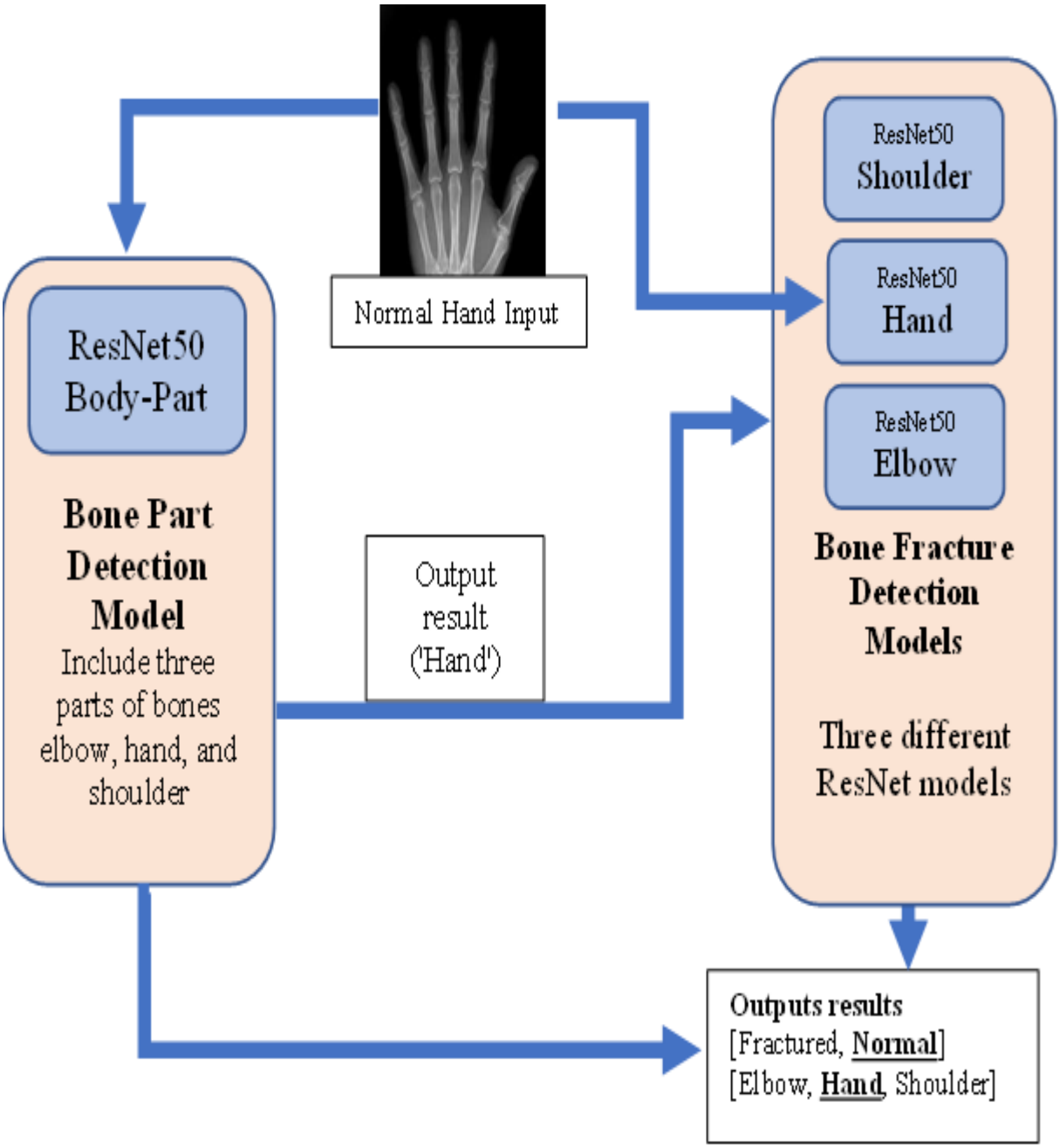

Figure 2.2: Block Diagram

In the above Figure 2.2 shows the block diagram of ResNet50 and here shows how the ResNet50 works and gives the result.

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.

Build a GUI interface that allows users to upload an X-ray image, initiate the detection process, and display the results.

Use a suitable image processing library like OpenCV or Pillow to preprocess the uploaded image and pass it through the fine-tuned ResNet-50 model[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.

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.

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

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

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.

Compare 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)
```

```
f1 = 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:

| **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

abnormal. 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.

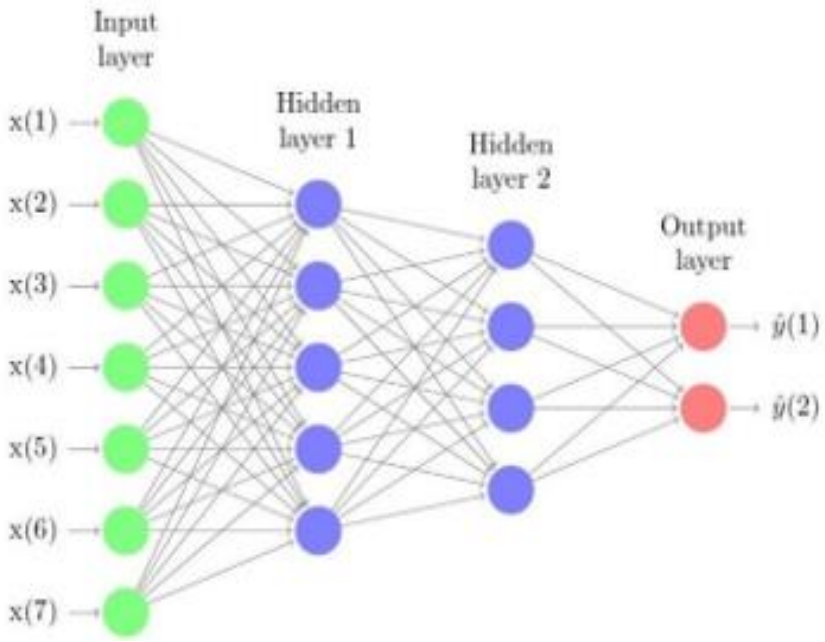

Figure 2.3: Structure of the network

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.

At 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.

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

ResNet50 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.

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

run 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

examples 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

any 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

- Desktop applications

3.1.7 Python installation

Step 1: Search python.org

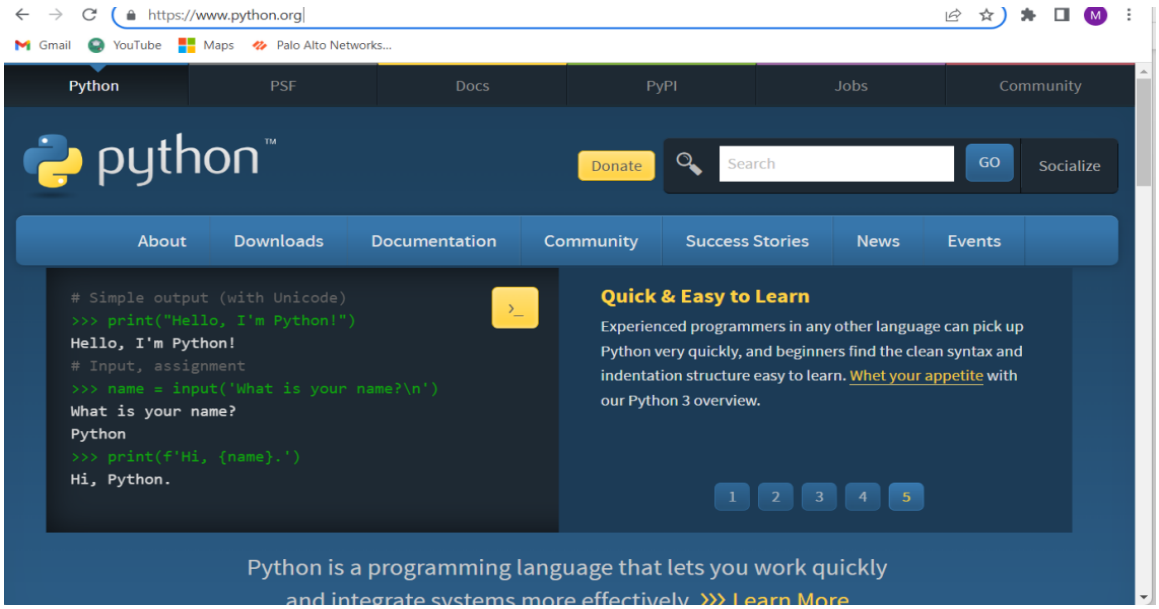

Figure 3.1: search python.org

Step 2: Go to downloads and select window

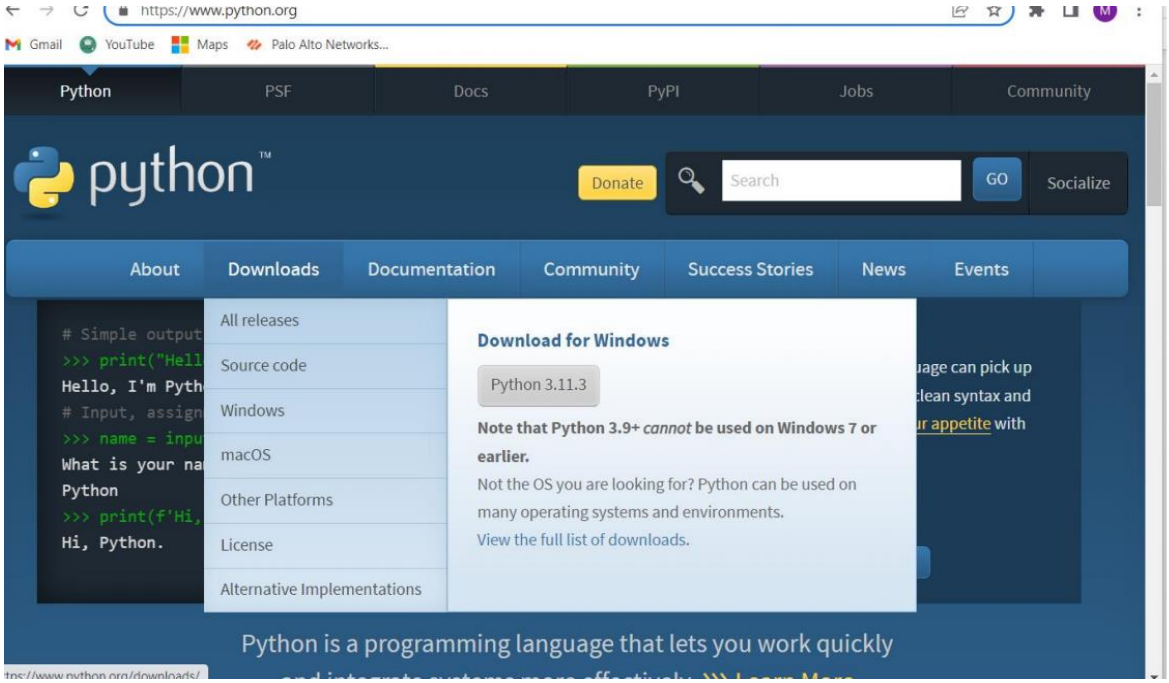

Figure 3.2: Go to downloads and select window

Step 3: Download Windows installer(64-bit)

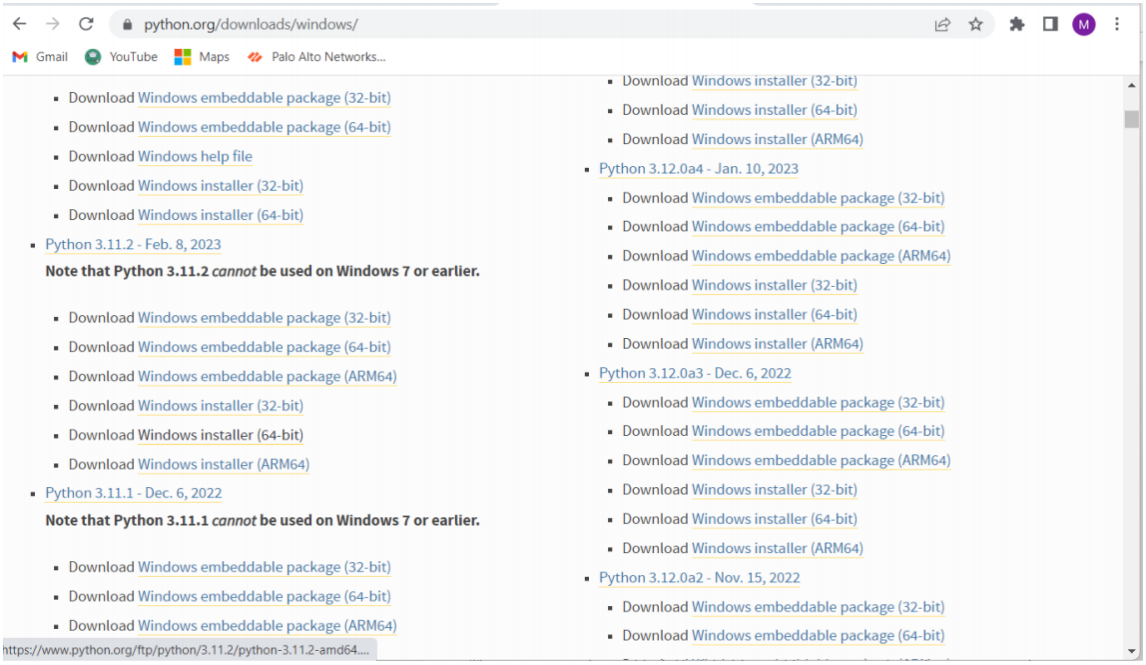

Figure 3.3: Download Windows installer(64-bit)

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 module for human beings. Used to programmatically control the mouse & keyboard. pip install pyautogui.

PyGetWindow: A simple, cross-platform module for obtaining GUI information on and controlling application's windows.

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 functions for 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)
```

```
self.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",
```

```
command=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
```

```
img = 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
```

```
window = 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
```

```
#               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):
```

```
total_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))
```

```
    print(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
```

```
# 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 np
```

```
import 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:
```

```
body_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__))
```

```
image_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)
```

```
train_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,
```

```
x_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 + "-------")
```

```
# 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()
```

```
my_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 .

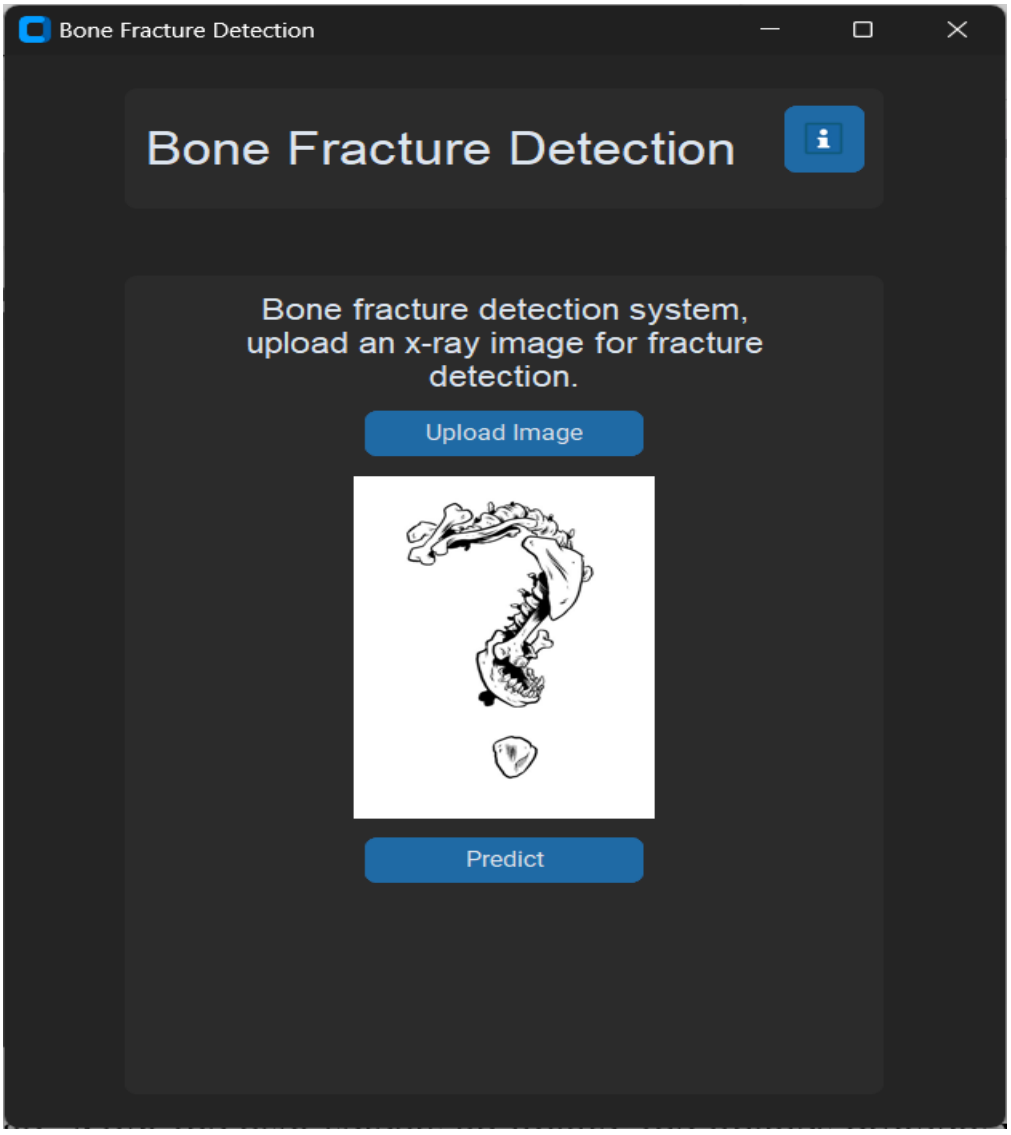

Figure 4.1: Processing image

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

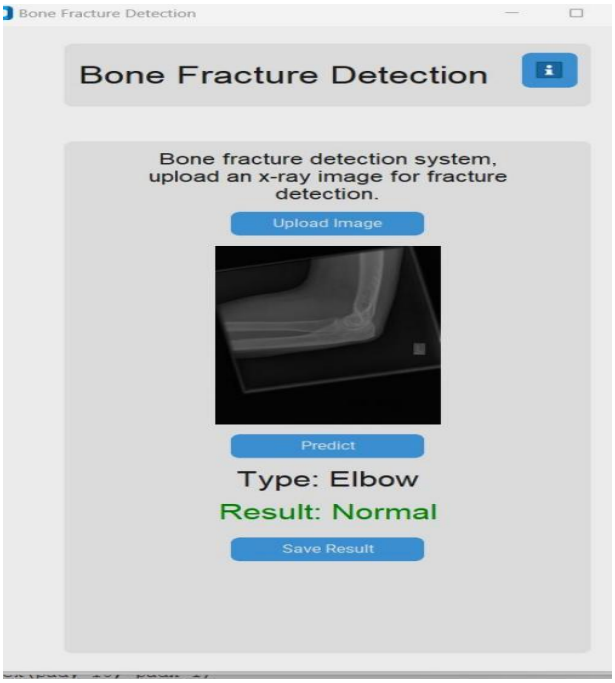

Figure 4.2: Normal Elbow

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

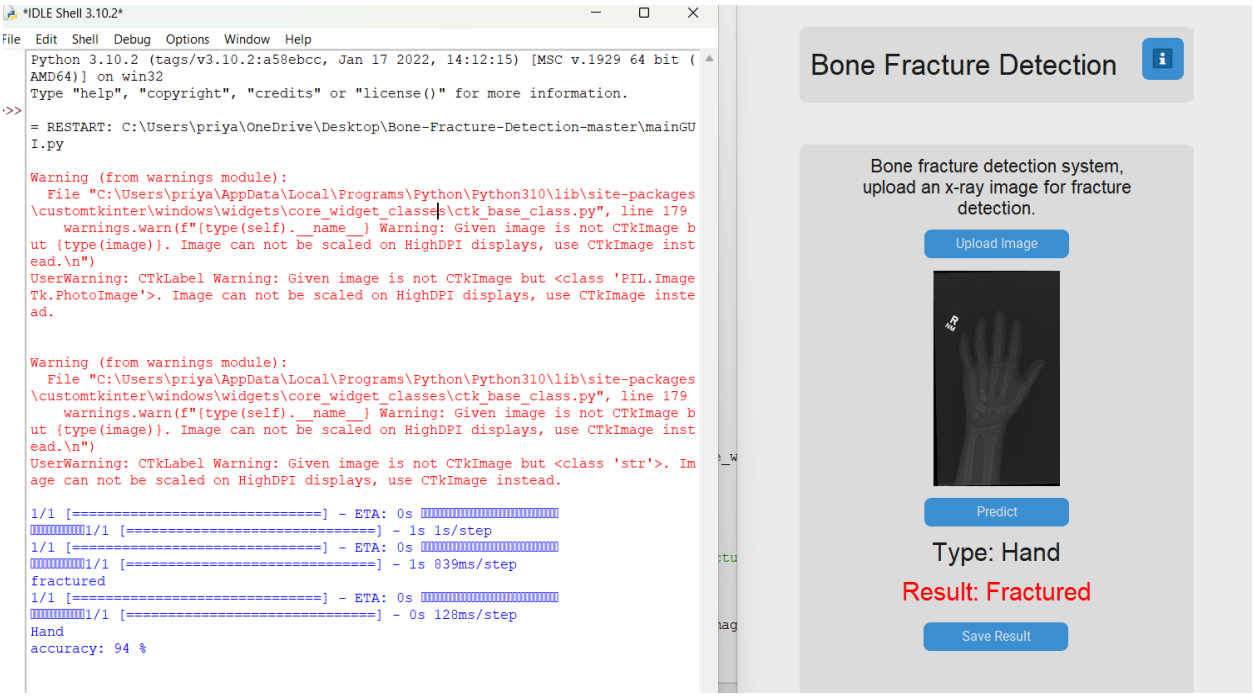

Figure 4.4: Accuracy

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.

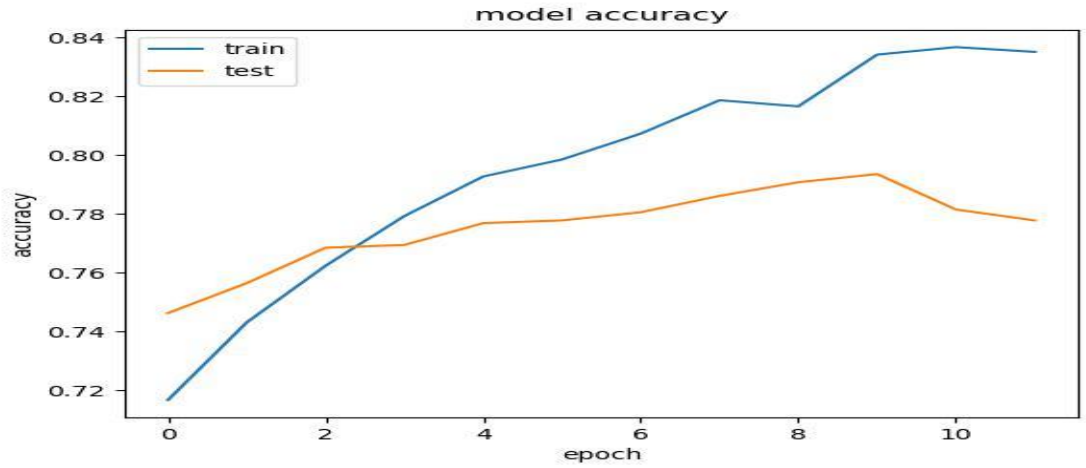

Figure 4.5: Model Accuracy

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.

CHAPTER 5

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].