DETECTION OF BONE FRACTURE USING CNN AND MATLAB

Dr. Ch. Rambabu
Associate Professeor,
Department of ECE,
Engineering Seshadri Rao Gudlavalleru Engineering Pradesh College,Gudlavalleru,Andhra Pradesh

A.Hema Sri
Department of ECE,
Seshadri Rao Gudlavalleru College,Gudlavalleru,Andhra

B.T.L.Kalyani
Department of ECE,
Seshadri Rao Gudlavalleru Engineering College,Gudlavalleru,Andhra Pradesh

A.Dayakar
Department of ECE,
Seshadri Rao Gudlavalleru Engineering College,Gudlavalleru,Andhra

Abstract **- Bone fractures are a significant medical concern requiring accurate and timely diagnosis to ensure effective treatment. This paper presents a deep learning-based system for bone fracture detection and classification using Convolutional Neural Networks (CNNs). The system integrates a user-friendly Graphical User Interface (GUI) to facilitate the input of bone fracture images. The process begins with the selection of a bone fracture image, followed by image resizing for uniformity. A pre-processed dataset of bone fracture images is utilized for training the CNN model to detect and classify fractures into three categories: Mild, Moderate, and Severe. The deep learning model leverages advanced CNN architectures for feature extraction and classification, achieving high accuracy in predicting fracture severity. The GUI enables users to input images, run the detection process, and view classified outputs seamlessly. This automated system demonstrates the potential to assist healthcare professionals in diagnosing fractures quickly and accurately, reducing dependency on manual assessments and enhancing clinical decision-making. The achieved accuracy underscores the effectiveness of the proposed approach, making it a valuable tool in medical imaging applications and orthopaedic care.**

Keywords - Bone Fracture images Dataset, Deep Learning algorithm, Convolutional Neural Network, GUI and Accuracy.

I. INTRODUCTION

Bone fractures, which are never a healthy situation, are graded by degree into Mild, Moderate, and Severe levels. Mild fractures are small breaks or cracks in the bone with minimal disturbance, and they are treated conservatively with immobilization and rest. Moderate fractures need casts or surgery to heal, while Severe fractures are bone displacement, soft tissue injury, or complications of fractures depending on more than one bone and thus complex operations and physiotherapy.

New technologies in medical imaging and deep learning such as Convolutional Neural Networks (CNNs) have greatly improved the accuracy of fracture classification and diagnosis to facilitate medical professionals in making the proper treatment decisions at the right time and in improving the recovery rate of patients.

Fractures most commonly occur from trauma, osteoporosis, or overuse. Trauma results from falls or accidents, and the size of the bone and surrounding muscles determines how much force a bone can withstand. Overuse fractures in athletes, usually to the foot or lower leg, are caused by cumulative stress. Fractures from osteoporosis occur with little trauma because the condition makes bones extremely brittle and prone to breaking. Fractures of bone are treated using RICE (Rest, Ice, Compression, Elevation), casting or splinting, physical therapy, or surgery depending on the magnitude. Prevention for fractures of bones consists of wearing proper equipment, resting, applying the right exercise technique, and maintaining a nutritionally balanced diet including calcium and vitamin D in order to support bones being well-conditioned. Daily exercise, especially weight exercising, will give bones strength as well as hardness and decrease likelihood of fractures.

II. LITERATURE SURVEY

Vijaykumar et al. suggest a rapid and effective algorithm to eliminate Gaussian noise without losing edges in digital images. The algorithm estimates the noise corruption, followed by replacing the center pixel with the mean of surrounding pixels by a threshold. It is more efficient in terms of computational complexity than common filters such as mean, Wiener, and bilateral filters. Experimental results indicate better performance for noise removal and edge preservation with lower computational complexity, rendering it simple to execute in hardware. [1]

Al-Khaffaf et al. introduce a noise removal algorithm for engineering drawings, with special concern for preserving fine details. The approach analyzes the neighbourhood of thin lines prior to deciding whether to remove or keep them. The algorithm is tested on scanned images with 15% salt-and-pepper noise corruption. Experiments demonstrate that the algorithm performs better in preserving quality, in terms of PSNR and MSE.[2]

"A fusion-classification approach to automatic detection of fractures from tibia X-ray images" by Mahndran and BaBoo suggests a four-step approach for the system: preprocessing, segmentation, feature extraction, and bone detection. The system employs classifiers such as Feed Forward Back Propagation Neural Networks (BPNN), Support Vector Machines (SVM), and Naïve Bayes (NB) for fusion classification. Detection rate and speed of Classification are improved dramatically according to the results [3].

Mahndran and BaBoo present an ensemble system for fracture detection from X-ray images based on fusion-based classifiers. Contrast, homogeneity, energy, and entropy are features extracted for classification. Several classifiers, namely BPNN, SVM, and NB, are evaluated with different combinations to facilitate fracture detection optimization. The experimental outcomes show that fusion classifiers, especially SVM and BPNN, give the optimal performance [4].

Rashmi et al. review some of the edge detection methods applied in digital image processing such as Prewitt, Sobel, and Canny operators. They emphasize Canny edge detection as being the most efficient technique, especially for noisy images. The Canny detector is adaptive and gives clear edges with fewer false alarms than other detectors. It is used extensively because it is accurate to segment images and has the capability to detect true edges consistently [5].

Hao et al. concentrate on the carpal bones' automatic separation in X-ray images of hands for detecting fractures. Crack detection is utilized by applying image processing methods such as enhancement, segmentation, and feature extraction. Segmentation uses the Canny edge detection, which yields better results. The system proves accuracy and efficiency while detecting ankle bone fractures from an X-ray image[6].

III. EXISTING METHOD

Utilizing Histogram of Oriented Gradients (HOG) for the classification of bone fractures is a traditional method used in image processing. HOG works by looking at the textures and patterns within bone images. It segments the image into tiny regions, computes the direction of the edges within each region, and builds a feature that aids in fracture recognition. These features are then labeled by a classifier, for example, Support Vector Machine (SVM), to identify the fracture type. While HOG is good, it performs poorly with intricate fracture patterns and lacks generalization compared to newer methods like Convolutional Neural Networks (CNNs), which learn automatically from data. Support Vector Machines (SVMs) are forms of machine learning algorithms that divide data into classes. For bone fractures, SVM graphs a line (which is called a hyperplane) to separate different fractures along significant points of the data. SVM can be very time-consuming and require much computing capacity, especially when utilizing a large amount of data. SVM too is sensitive to noisy data and this might affect its accuracy. While HOG and SVM are useful, they fail to pick up all information in fracture images as efficiently as CNNs, which are generally more accurate for this type of task.

IV. PROPOSED MODEL

The proposed bone fracture detection system consists of Convolutional Neural Networks (CNNs) with a Graphical User Interface (GUI) for simplicity of use. The users input images of bone fractures ,which are preprocessed and subsequently classified into three classes: Mild, Moderate, and Severe by the trained CNN. The system offers real-time output and accuracy rates in order to assess performance, offering a simple and cost-effective system for automated fracture detection at hospitals. Figure1 shows the block diagram of proposed model.

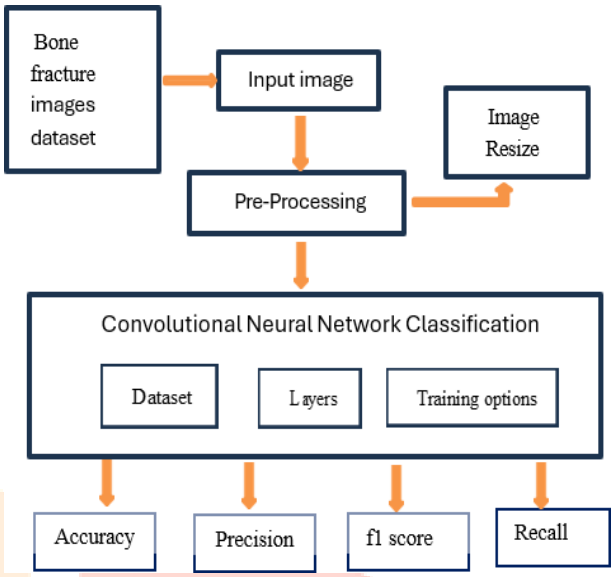


Figure 1: Block diagram of proposed model

Image Resize in MATLAB:

Image resize in MATLAB scales an image to a size standard for processing. An image is resized using the imresize command in MATLAB, which provides varying interpolation methods (e.g., nearest-neighbor, bilinear, and bicubic) for quality control. This is done for processing such as machine learning, medical images, and object detection, in which all the images should have the same dimensions.

GUI in MATLAB:

GUI in MATLAB enables interactive handling of image processing operations utilizing graphical controls such as buttons and sliders. GUI performs tasks such as the loading, scaling, and conversion of images to different formats, filtering, and CNN- based classification. GUI simplifies intricate operations rendering it feasible to process images along with checking models interactively.

1. Initialize

Gui Code function initializes the GUI in a manner where just a single copy of it gets executed and controls its life cycle.

2. OpeningFunction

The Gui_Code_OpeningFcn initializes the GUI, initializes default output, and sets up the GUI for interactive use by a user.

3. OutputFunction:

The Gui_Code_OutputFcn gets back the handle of the GUI main output.

4. ButtonCallbacks:
Every button will perform some associated action such as opening, resizes, gray-scale conversion, filters application, segmenting and classification Images with CNN. Each button performs certain operations such as opening, resizing, gray scale conversion, filtering, segmenting, and classifying images using a CNN.
5. EditBoxCallbacks:
Functions control edit box operations for text display or input.
6. ToggleButtonCallback:
Controls the toggle button state.

Convolutional Neural Networks (CNNs):

Convolutional Neural Networks (CNNs) are among deep learning's building blocks that are revolutionizing application in a multitude of fields, particularly computer vision, with their record- breaking ability to recognize and learn challenging patterns in image data. MATLAB Convolutional Neural Networks have caused a lot of excitement with their record-breaking ability to process image-based complex data. Fundamentally, a CNN captures the organization of the visual cortex in possessing many layers that learn successively and obtain hierarchical features from raw pixel inputs. Hierarchical feature extraction allows CNNs to automatically detect prominent information such as edges, texture, and shapes and thus are very potent at image classification, object detection, and image segmentation tasks. It is relatively simple to train and develop CNNs using the MATLAB development environment with the assistance of the deep learning toolbox to provide pre-defined layers, training parameters, and visualization functionality. MATLAB flexibility also facilitates preprocessing, augmentation, and addition of custom architectures and makes it straightforward for researchers and practitioners to personalize CNNs for different purposes. This overview delves into the applicability of Convolutional Neural Networks in MATLAB and how they push computer vision possibility horizons. MATLAB and deep learning combining in more ways, CNNs push horizons from medical diagnosis in imaging to autonomous vehicles, showing their wide- ranging effect across today's technology frontiers. The design of the network may vary depending on the type and the number of layers utilized. The type and the number of layers utilized are determined by the application or data in question. For example, classification networks may utilize a classification layer and a soft max layer, while regression networks must utilize a regression layer as the network's output layer. You can utilize a straightforward network with a variety of convolutional layers to learn from a small set of black-and- white image data. When it comes to the more complex data with millions of color images, though, you might find yourself using a more sophisticated network with numerous convolutional and fully connected layers.

Image Input Layer:

The MATLAB Image Input Layer is the most important factor in successfully feeding image data to different deep learning operations. Since it is the input layer of the neural network architecture, the layer facilitates image data input of different dimensions and types. The layer provides provision for preprocessing and normalization requirements of model training so that the network can learn key features efficiently. Furthermore, the Image Input Layer allows for simplicity of use when using augmented datasets, which also encourages model resistance. Simple to use and supporting most neural network frameworks, this layer allows it to be extremely easy to integrate image data into MATLAB-based deep learning operations.

Convolutional Layer:

A 2-D convolutional layer applies sliding convolutional filters to 2-D input. Create a 2-D convolutional layer using convolution 2-D layer. The convolutional layer consists of various component.

Filters and Stride:

A convolutional layer consists of neurons that connect to subregions of the input images or the outputs of the previous layer. The layer learns the features localized by these regions while scanning through an image. When creating a layer using the convolution2dLayer function, you can specify the size of these regions using the filter Size input argument. For each region, the train Network function computes a dot product of the weights and the input, and then adds a bias term. A set of weights that is applied to a region in the image is called a filter. The filter moves along the input image vertically and horizontally, repeating the same computation for each region. In other words, the filter convolves the input. The image shows a 3- by-3 filter scanning through the input. The lower map represents the input and upper map represents the output. The step size with which the filter moves is called a stride. We can specify the step size with the Stride name-value pair argument. The local regions that the neurons connect to can overlap depending on the filter Size and 'Stride' values.
This figure 2 shows a 3-by-3 filter scanning through the input with a stride of 2. The lower map represents the input and the upper map represents the output.

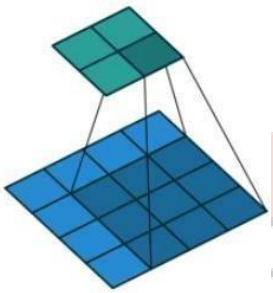


Figure 2: Filters and Stride

The number of weights in a filter is h * w * c, where h is the height, and w is the width of the filter, respectively, and c is the number of channels in the input. For example, if the input is a colour image, the number of colour channels is 3. The number of filters determines the number of channels in the output of a convolutional layer. Specify the number of filters using the Num filter argument with the convolution2dLayer function.

Dilated Convolution:

A dilated convolution is a convolution in which the filters are expanded by spaces inserted between the elements of the filter. Specify the dilation factor using the 'Dilation Factor' property. Use dilated convolutions to increase the receptive field (the area of the input which the layer can see) of the layer without increasing the number of parameters or computation. The layer expands the filters by inserting zeros between each filter element. The dilation factor determines the step size for sampling the input or equivalently the up-sampling factor of the filter. It corresponds to an effective filter size of (Filter Size – 1). * Dilation Factor + 1.

For example, a 3-by-3 filter with the dilation factor [2 2] is equivalent to a 5-bv-5 filter with zeros between the elements. This figur ated by a factor of two scanning throu represents the input and the upper map

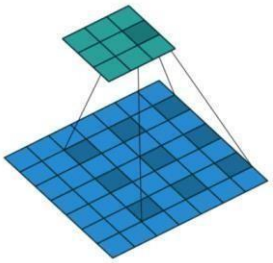


Figure 3: Dilated Convolution

Feature Maps:

As a filter moves along the input, it uses the same set of weights and the same bias for the convolution, forming a feature map. Each feature map is the result of a convolution using a different set of weights and a different bias. Hence, the number of feature maps is equal to the number of filters. The total number of parameters in a convolutional layer is ((h*w*c + 1) *Number of Filters), where 1 is the bias.

Padding:

We can also apply padding to input image borders vertically and horizontally using the 'Padding' name-value pair argument. Padding is values appended to the borders of the input to increase its size. By adjusting the padding, you can control the output size of the layer. This figure 4 shows a 3-by-3 filter scanning through the input with padding of size 1. The lower map represents the input and the upper map represents the output.

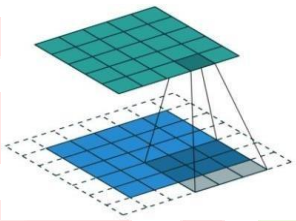


Figure 4: Padding

Output size:

The output height and width of a convolutional is

(Input Size – ((Filter Size – 1) *Dilation Factor + 1) + 2*Padding)/Stride + 1.

This value must be an integer for the whole image to be fully covered. If the combination of these options does not lead the image to be fully covered, the software by default ignores the remaining part of the image along the right and bottom edges in the convolution.

Number of Neurons:

The product of the output height and width gives the total number of neurons in a feature map, say Map Size. The total number of neurons (output size) in a convolutional layer is Map Size*Number of Filters.

Usually, the results from these neurons pass through some form of nonlinearity, such as rectified linear units (ReLU).

Number of Layers:

A convolutional neural network can consist of one or multiple convolutional layers. The number of convolutional layers depends on the amount and complexity of the data.

Batch Normalization Layer:

Create a batch normalization layer using batch Normalization Layer. A batch normalization layer normalizes a mini-batch of data across all observations for each channel independently. To speed up training of the convolutional neural network and reduce the sensitivity to network initialization, use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers. The layer first normalizes the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation. Then, the layer shifts the input by a learnable offset β and scales it by a learnable scale factor γ. β and γ are themselves learnable parameters that are updated during network training.

Batch normalization layers normalize the activations and gradients propagating through a neural network, making network training an easier optimization problem. To take full advantage of this fact, you can try increasing the learning rate. Since the optimization problem is easier, the parameter updates can be larger and the network can learn faster. You can also try reducing the L2 and dropout regularization. With batch normalization layers, the activations of a specific image during training depend on which images happen to appear in the same mini-batch. To take full advantage of this regularizing effect, try shuffling the training data before every training epoch. To specify how often to shuffle the data during training, use the 'Shuffle' name-value pair argument of training Options.

ReLU Layer:

Create a ReLU layer using reluLayer. A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero. Convolutional and batch normalization layers are usually followed by a nonlinear activation function such as a rectified linear unit (ReLU), specified by a ReLU layer. A ReLU layer performs a threshold operation to each element, where

any input value less than zero is set to zero, that is,

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

The ReLU layer does not change the size of its input. There are other nonlinear activation layers that perform different operations and can improve the network accuracy for some applications. For a list of activation layers, see Activation Layers.

Cross Channel Normalization (Local Response Normalization)Layer:

Create a cross-channel normalization layer using cross Channe Normalization Layer. A channel-wise local response (cross-channel) normalization layer carries out channel-wise normalization. This layer performs a channel-wise local response Normalization. It usually follow the ReLU activation layer.
This layer replaces each element with a normalized value it obtains using the elements from a certain number of neighboring channels (elements in the normalization window).
That is, for each element x in the input, train Network computes a normalized value x ′ Using

$$x' = \frac{x}{\left(K + \frac{\alpha * ss}{windowChannelSize}\right)^{\beta}},$$

where K, α, and β are the hyperparameters in the normalization, and ss is the sum of squares of the elements in the normalization window. You must specify the size of the normalization window using the window Channel Size argument of the cross Channel Normalization Layer function. You can also specify the hyperparameters using the Alpha, Beta, and K name-value pair arguments. The previous normalization formula is slightly different than what is presented. You can obtain the equivalent formula by multiplying the alpha value by the windowChannelSize.

Max and Average Pooling Layers:

A 2-D max pooling layer performs down sampling by dividing the input into rectangular pooling regions, then computing the maximum of each region. Create a max pooling layer using maxPooling2dLayer. A 2-D average pooling layer performs down sampling by dividing the input into rectangular pooling regions, then computing the average of each region. Create an average pooling layer using averagePooling2dLayer. Pooling layers follow the convolutional layers for down-sampling, hence, reducing the number of connections to the following layers. They do not perform any learning themselves, but reduce the number of parameters to be learned in the following layers. They also help reduce overfitting. A max pooling layer returns the maximum values of rectangular regions of its input. The size of the rectangular regions is determined by the poolSize argument of max Polling Layer. For example, if poolSize is [2 3], then the layer returns the maximum value in regions of height 2 and width 3.
An average pooling layer outputs the average values of rectangular regions of its input. The size of the rectangular regions is determined by the poolSize argument of averagePoolingLayer. For example, if poolSize is [2 3], then the layer returns the average value of regions of height 2 and width 3.
Pooling layers scan through the input horizontally and vertically in step sizes you can specify using the 'Stride' name-value pair argument. If the pool size is smaller than or equal to the stride, then the pooling regions do not overlap.
For nonoverlapping regions (Pool Size and Stride are equal), if the input to the pooling layer is n-by-n, and the pooling region size is h-by- h, then the pooling layer down-samples the regions by h. That is, the output of a max or average pooling layer for one channel of a convolutional layer is n/h-by-n/h. For overlapping regions, the output of a pooling layer is (Input Size – Pool Size + 2*Padding)/Stride + 1.

Fully Connected Layer:

Create a fully connected layer using fully Connected Layer. A fully connected layer multiplies the input by a weight matrix and then adds a bias vector. The convolutional (and down-sampling) layers are followed by one or more fully connected layers. As the name suggests, all neurons in a fully connected layer connect to all the neurons in the previous layer. This layer combines all of the features (local information) learned by the previous layers across the image to identify the larger patterns. For classification problems, the last fully connected layer combines the features to classify the images. This is the reason that the output Size argument of the last fully connected layer of the network is equal to the number of classes of the data set. For regression problems, the output size must be equal to the number of response variables. You can also adjust the learning rate and the regularization parameters for this layer using the related name-value pair arguments when creating the fully connected layer. If you choose not to adjust them, then train Network uses the global training parameters defined by the training Options function. For details on global and layer training options.

Output Layers:

Softmax and Classification Layers:

A softmax layer applies a softmax function to the input. Create a softmax layer using SoftMax Layer. A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes. Create a classification layer using classification Layer.
For classification problems, a softmax layer and then a classification layer usually follows the final fully connected layer. The output unit activation function is the softmax function:

$$y_r(x) = \frac{\exp(a_r(x))}{\sum_{j=1}^{k} \exp(a_j(x))},$$

Were,

$$0 \le y_r \le 1 \text{ and } \sum_{j=1}^{k} y_j = 1.$$

The softmax function is the output unit activation function after the last fully connected layer for multi-class classification problems:

$$P(c_r|x,\theta) = \frac{P(x,\theta|c_r)P(c_r)}{\sum_{j=1}^{k} P(x,\theta|c_j)P(c_j)} = \frac{\exp(a_r(x,\theta))}{\sum_{j=1}^{k} \exp(a_j(x,\theta))},$$

Where, the conditional probability of the sample given class r,and P(cr) is the class prior probability:

$$0 \le P(c_r|x,\theta) \le 1 \text{ and } \sum_{j=1}^{k} P(c_j|x,\theta) = 1.$$

where N is the number of samples, K is the number of classes, wi is the weight for class *i*, t_{ni} is the indicator that the *n*th sample belongs to the *ith* class, and y_{ni} is the output for sample n for class *i*, which in this case, is the value from the softmax function. In other words, y_{ni} is the probability that the network associates the nth input with class *i*.

CNN Classification:

Mild Fracture:

A mild fracture, also known as a hairline fracture or stress fracture, is a small crack in the bone. A mild fracture is most often brought on by repeated stress or trauma on the bone.

Characteristics of a mild fracture:

- Small, thin crack in the bone
- Little or no displacement of bone fragments
- Little pain and inflammation
- Little impairment of daily activity

-Typically treated with immobilization, pain management, and exercise therapy.

Examples of minimal fractures are:

-A small crack in the fibula (lower leg bone on the lateral side) due to running or jumping

- A hairline fracture of the wrist caused by falling on the hand.

Moderate Fracture:

Moderate fracture is a larger break in the bone that can be treated more intensely. A fracture can be caused by a multitude of reasons such as trauma, injury during sport or osteoporosis.

Characteristics of a moderate fracture:

- Wider crack or fracture in the bone
- Some displacement of the bone fragments, but still stable
- Slight swelling and pain

-Some interference with daily functioning, but able to do some tasks

-Surgical treatment may be required, including reduction and fixation, in addition to immobilization and physical therapy

Moderate fractures consist of:

-A humerus (upper arm bone) fracture due to a fall on an outstretched arm

-A tibia (shin bone) fracture due to a sports injury, such as a soccer or football injury.

Severe Fracture:

A severe fracture is a comminuted or complex fracture (multiple fragments) which requires immediate medical attention. The fracture may be caused by high-energy trauma, i.e., an auto accident or fall from a height.

Characteristics of a severe fracture:

- A comminuted or complex fracture (many fragments)

-A high degree of displacement of the bone fragments, making it unstable

-Severe pain and swelling

-Has a significant impact on everyday activities, as even small things are difficult to accomplish

-Typically must be treated surgically, i.e., ORIF, in order to align the bone and permit it to heal

Severe fractures are illustrated as follows:

-A femoral (thigh bone) fracture that is a complex fracture and resulted from an automobile accident.

-A pelvic comminuted fracture resulting from a fall from a height.

Accuracy:

"A accuracy" is also a common performance measure in MATLAB utilized to calculate the overall accuracy of a classification model, particularly on binary and multi-class classification models. Accuracy is just the number of correctly classified samples to the number of all the samples in the data set. To better comprehend Accuracy, familiarity with jargon terms for True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) will be helpful:

-True Positives (TP): Number of positive instances that have been correctly predicted by the classifier.

-False Positives (FP): Number of negative instances that have been incorrectly predicted as positive by the classifier.

-True Negatives (TN): Number of negative instances that have been correctly predicted by the classifier.

-False Negatives (FN): Number of positive instances that have been incorrectly predicted as negative by the classifier. Accuracy can be obtained by using the following formula:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

i.e., Accuracy is the proportion of correct classification of the samples as positive or negative to the number of available samples in the data set.

High Accuracy value indicates that classifier is good and classifying correctly majority of the samples. However, Accuracy may not be the best metric of model performance when working with datasets that are imbalanced and in which one of the classes is significantly larger than the other. For example, if in an imbalanced dataset one class is significantly larger than the other class, a classifier may have high Accuracy by simply predicting the majority class all the time even if it would fail to predict the minority class correctly.

Precision:

Precision refers to the precision of the positive predictions by the classifier. Precision indicates what percentage of the positive cases predicted were actually positive. Precision is defined as:

Precision = TP / (TP + FP)

-TP (True Positives): The number of true positive samples, i.e., the samples which are actually positive and classified correctly.

-FP (False Positives): The number of false positive samples, i.e., the samples which are not actually positive but incorrectly classified as positive.

A high Precision measure means that if the classifier labels a sample positive, it most likely is positive. Precision is particularly crucial in situations where there is a high cost of False Positives (e.g., spam filtering or disease diagnosis).

Recall:

Recall estimates the classifier's capacity to recognize all the true positive cases. It responds to the question: "Out of all the true positives, how many did the model identify correctly?" Recall is computed as: Recall = TP / (TP + FN)

-TP (True Positives): The number of samples that were correctly classified as positive.

-FN (False Negatives): The number of positive samples that were mistakenly labeled as negative.

High Recall value indicates that the classifier performs well in finding positive cases. Recall is important when it is expensive to miss positive cases, for example, disease detection or fraud detection.

F1 Score:

The F1 Score is the harmonic mean of Precision and Recall, which gives a single measure that averages both. It's especially useful when you need to achieve a balance between Precision and Recall, particularly in datasets with imbalanced classes. The formula is:

F1 Score = 2 × (Precision × Recall / Precision + Recall)

A high F1 Score means that the classifier achieves a good balance between Precision and Recall.

If Precision and Recall are very different in value, the F1 Score will be nearer to the lower value.

V. Results and Discussion

This paper is a comparative analysis of three categories of fractures. Different classification methods such as CNN algorithm and graphical user interface are employed for fracture type detection. For this purpose, we gathered a normal human's bone x- ray MATLAB is employed to implement various CNN algorithm and graphical user interface for detection of whether an image is mild, moderate, severe, and also provides the information about Accuracy, Precision, F1 score, Recall.

Figure 5:Input image

This figure 5 represent Process begins with capturing an input X-ray image that is preprocessed to resize for standardization.

Figure 6 :Resized image

This Figure 6 represent Preprocessed images are employed together with CNNs for training the model, doing feature extraction, and learning the fracture patterns.

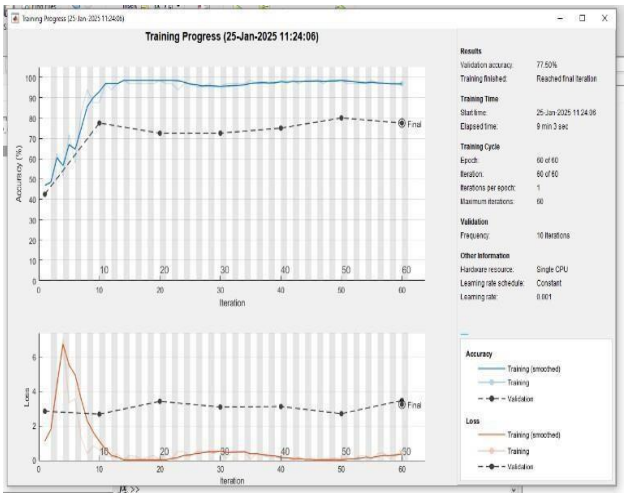
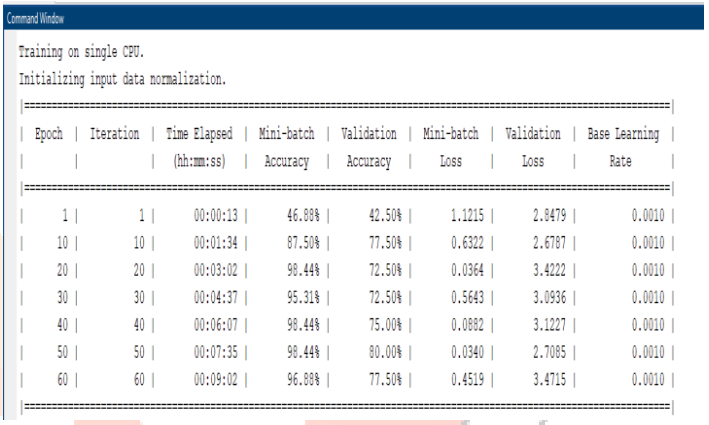


Figure 7 : Training Progress Image

This figure 7 represent trained model classifies the severity of fracture into categories such as mild, moderate, or severe.



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:13	46.88%	42.50%	1.1215	2.8479	0.0010
10	10	00:01:34	87.50%	77.50%	0.6322	2.6787	0.0010
20	20	00:03:02	98.44%	72.50%	0.0364	3.4222	0.0010
30	30	00:04:37	95.31%	72.50%	0.5643	3.0936	0.0010
40	40	00:06:07	98.44%	75.00%	0.0882	3.1227	0.0010
50	50	00:07:35	98.44%	80.00%	0.0340	2.7085	0.0010
60	60	00:09:02	96.88%	77.50%	0.4519	3.4715	0.0010

Figure 8 : Training Iterations Image

This figure 8 represent Training Iterations Image

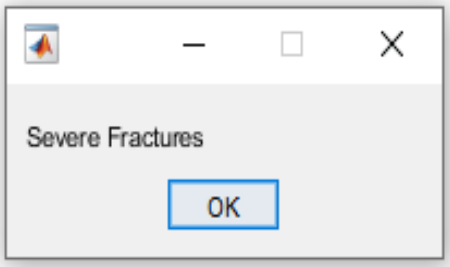


Figure 9 : Classification Result Image

This figure 9 represent performance metrics including accuracy, precision, recall, and F1- score are outputted in the MATLAB command line, indicating overall performance of the model.

Table1: Accuracy,Precision,Recall,f1Score

Command window
Accuracy : 85.48% Precision : 84.88% Recall : 85.11% f1Score : 84.82%

In addition, a confusion matrix is generated to see how the actual and predicted classes relate to each other, and it gives more insight into how well the classification is performing.

Table2: Confusion Matrix

47	**6**	**1**
4	**27**	**2**
	5	**32**

This paper represents a good plan for medical image analysis that will result in better diagnosis and help medical professionals make faster, better decisions.

GUI OUTPUT:

The system initiates by importing input X-ray image through Graphical User Interface (GUI), offering an user-friendly interface. Pre-processing input image is resizing so input sizes will be the same for the Convolutional Neural Network (CNN). The model of CNN learns to classify the severity of the fracture into mild, moderate, or severe based on features obtained from it. The “classification result” is displayed prominently along with important performance measures like accuracy, precision, recall, and f1-score on the command window for proper verification. A “confusion matrix” is also created to project the relationship of actual and estimated classes for profound understanding of how well the model is performing. The integration of the GUI improves user interaction, making the system accessible to medical professionals, enhancing diagnostic accuracy, and supporting more efficient decision-making. This figure 10 represents a GUI Image and This figure 11 represents the GUI Image – Output.

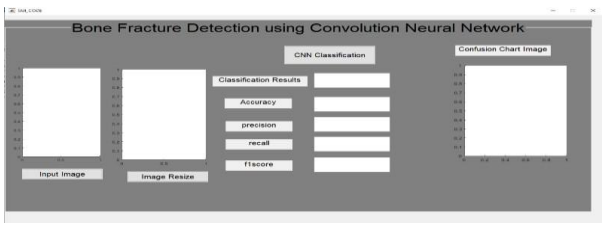


Figure 10 : GUI Image

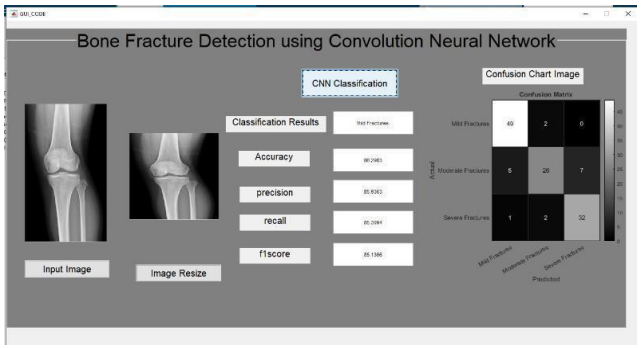


Figure 11 : GUI Image – Output

VI. CONCLUSION

This paper introduces an optimized and automated bone fracture detector based on Convolutional Neural Networks (CNN) on MATLAB. Through preprocessing of X-ray images, resizing, and data augmentation, the CNN model was trained to differentiate fractures into categories such as mild, moderate, and severe. Its structure consists of convolutional layers, batch normalization, dropout for regularization, and fully connected layers, with high accuracy without overfitting.

Performance assessment by accuracy, precision, recall, and F1-score validates the model's efficacy and stability. The confusion matrix indicates the right and wrongly classified cases, providing further insight into class-wise performance. An intuitive GUI was created to facilitate ease of use, enabling clinicians or users to upload X-ray images and obtain real-time classification results.

The model had up to 87% accuracy, and this indicates that CNNs are appropriate for classifying medical images. With additional advancements such as bigger datasets, transfer learning, or real-time use, the system can go a long way in assisting clinical diagnostics and facilitating early, correct treatment decisions.

VII. REFERENCES

[1] V. Vijaykumar, P. Vanathi, and P. Kanagasabapathy, "Fast and efficient algorithm to remove Gaussian noise in digital images," *IAENG Int. J. Comput. Sci.*, vol. 37, no. 1, 2010.

[2] H. Al-Khaffaf, A. Z. Talib, and R. A. Salam, "Removing salt-and-pepper noise from binary images of engineering drawings," in *Proc. 19th Int. Conf. Pattern Recognit. (ICPR)*, 2008, pp. 1–4.

[3] S. K. Mahndran and S. Santhosh BaBoo, "An enhanced tibia fracture detection tool using image processing and classification fusion techniques in X-ray images," *Int. J. Comput. Appl.*, vol. 11, no. 14, Aug. 2011. [Online]. Available: http://www.ijcaonline.org (Online ISSN: 0975-4172, Print ISSN: 0975-4350)

[4] S. K. Mahndran and S. Santhosh BaBoo, "An ensemble system for automatic fracture detection," *IACIT Int. J. Eng. Technol.*, vol. 4, no. 1, Feb. 2012.

[5] Rashmi, M. Kumar, and R. Saxena, "Algorithm and technique on various edge detection: A survey," *Int. J. Eng. Sci. Emerg. Technol.*, vol. 4, no. 3, Jun. 2013.

[6] S. Hao, Y. Han, J. Zhang, and Z. Ji, "Automatic isolation of carpal-bone in hand X-ray medical image," in *Proc. [Conference name not provided]*, 2013

IJCRT