Classification of X-rays With and Without Fractures Using Feature Extraction and a Convolutional Neural Network

Angela Felicia Marie M. Reyes
Department of Manufacturing Engineering and Management
De La Salle University
Manila, Philippines
angela_felicia_reyes@dlsu.edu.ph

***Abstract*—Bone fractures can occur in different regions of the body with varying severity and form depending on the type of injury. Some particular fractures, such as scapular and radial head elbow fractures, are difficult to spot, which could cause delays in diagnosis and, in effect, treatment. The goal of this study is to detect and classify different fractures in a variety of regions in the body making use of an extensive dataset of X-rays across all body regions. The data used is a compilation of three bone fracture datasets – Bone Break Classifier Dataset, bone_fracture, and fracture – made by Mohan Kumar, Abdelaziz Faramawy, and Harsha Arya respectively, with the datasets compiled into one named Bone Fracture Multi-Region X-ray Data by Madushani Rodrigo. These images were preprocessed in order to ensure the contrast between the bone and the fracture is visible. Masking was used to differenciate the bone from the background and fed into a convolutional neural network for classification. After training and validating, the network was able to classify the test set with 99.21% accuracy. This could help with the workflow of radiologists and reduce delays in diagnosis. This would be especially useful in emergency radiology for trauma patients.**

Keywords—x-ray, computer vision, convolutional neural network, classification, bone fracture

I. Introduction

A. Bone Fractures

Bone fractures typically occur after bodily trauma, damaging and breaking the bone. For proper treatment, the fracture should be addressed as soon as possible, making the quick diagnosis of such injuries to be influential in the outcome of the patient in question. Diagnosis is usually done through the use of medical imaging, the most common being X-rays. While diagnosis is usually straightforward, more subtle fractures are prone to misinterpretation or undetection, delaying treatment and possibly aggravating the injury further [1]. An example of the latter would be in nasal fractures among children. Nasal fractures are a common type of pediatric facial fracture, and corrective treatment must be within two weeks of the fracture occuring. Due to how pediatric bones tend to fuse, if not treated within that time frame the nasal bones will join together asymmetrically causing issues both functionally and aesthetically [2].

B. Computer Vision and Machine Learning

Computer vision and machine learning have been applied to orthopedics and medical imaging in multiple different ways, whether this be measuring bone density and correlating it to bone fracture risk for people with osteoporosis [3] or using computer vision to detect and classify bone fractures [4, 5, 6, 7].

II. Methodology

A. Data Collection

The database, titled "*Bone Fracture Multi-Region X-ray Data*" was originally downloaded from Kaggle and uploaded by Madushani Rodrigo [8]. This dataset is a compilation of three other bone fracture datasets, namely *Bone Break Classifier Dataset*, *bone_fracture*, and *fracture*. These were made by Mohan Kumar, Abdelaziz Faramawy, and Harsha Arya respectively. This was then uploaded into a personal Google Drive for easy importing into Google Colab. The dataset was already split into training, validating, and testing groups. Within each group, a folder was created for each class: fractured or not fractured.

B. Libraries Used

The libraries used in this project were os, NumPy, OpenCV, Keras [9] and its different sublibraries such as layers, SKLearn, Matplotlib, and types. These are for operating system and file manipulation, numerical operations, computer vision, image processing, machine learning, confusion matrices, plots and charts, and identifying data types. The Google Drive mounting functionality was also imported from the google.colab library.

C. Data Importing

The personal Google Drive was mounted to Colab and the dataset path of the folder with data was stored in the dataset_path variable. Due to the data being split prior, the data had to be stored and processed separately to prevent them from merging. The path of the training, testing, and validation sets were taken by appending the folder names (train, test, val) to the dataset_path variable. From there, the path files for the two class folders in each group were stored as well. Empty arrays were declared for the images themselves and their labels per group, with a 0 label for not fractured and a 1 for fractured. Using a for loop and going through the respective directories, each image was read. If the file type did not come up as a NoneType, the image would be resized using OpenCV [10] to 256x256 for uniformity. The resized image would then be appended to its array and its respective label would be appended to the label array as well. If the file type did output as a

NoneType, the image would simply be skipped. This process was repeated for all groups.

After all the images and labels were stored in their arrays, the length and shape of each were printed to ensure the number of images and labels were equal and to check the shape of each set.

D. Data Preprocessing

Preprocessing the data came in 8 steps: converting the image to grayscale, denoising with Gaussian blur, using OTSU thresholding to mask the region of interest (ROI), applying Contrast Limited Adaptive Histogram Equalization (CLAHE) to the mask, and using erosion on the mask to make the ROI more precise. The image masks and finalized processed images were saved in their own arrays. Again, this process was repeated per set.

A for loop for the pre-processing of the image to obtain the masks was constructed, going through every item in the appropriate array. The conversion to grayscale was done using the cvtColor function from the OpenCV library. This had to go through try and except blocks, since there were certain images that could not be passed through the function for different reasons. The alternative would be to comb through the images and find which ones were returning with an error, which would be tedious and unrealistic. Using the try and except blocks allowed the program to skip over these images and proceed. Once the image was in a grayscale color format, it was passed through a Gaussian blur function to smoothen and denoise it. The mask area of the ROI was extracted using an OTSU threshold and the image that falls under said mask was obtained using a bitwise_and function. This was then appended into an array to store all the masks.

The mask was then passed into another for loop to apply CLAHE to the image, boosting its contrast. CLAHE is a type of histogram equalization – a process that involves spreading out the histogram of values of a given image, making the low values lower and high values higher to increase differentiation and therefore contrast – that divides the image into tiles and applies equalization to each tile independently. By taking tiles of the image, each tile has increased contrast local to its immediate neighbors. If normal histogram equalization were used, the histogram would be global. Given that the backgrounds of the X-rays are typically much darker than the bones or tissue, the region of interest would end up washed out, making the bones and the tissue surrounding it indistinguishable from one another. The localization technique of CLAHE ensures that the bones and tissue remain separated in value [12987312 sources]. The image was then eroded to get the mask closer to the bones themselves. Afterward, the processed images were stored in an array. The first 10 denoised images, the image masks, and the final processed images of each set were displayed in a plot for visual checking.

E. Model Design

The CNN model used is from the sequential class, which takes a stack of layers and groups it into a model [11]. The design was based on examples given in previous modules, from the workflow example in the Keras documentation [12], and on the functionality of each layer [13, 14]. The network followed a general pattern of a convolutional layer, a pooling layer, and a dropout layer, with those three layers repeated two more times. This is then followed by a flatten layer, a dense layer, a dropout layer, a flatten layer again, a rescaling layer, then finally a dense layer as the output. The convolutional layers were activated using the rectified linear unit activation (relu) function [keras documentation source] with an output shape of (3, (3,3)) and an input shape of (256, 256, 3) to match the data. The output size of each convolutional layer would increase, going from (3, (3,3)), (64, (3,3)), and then (128, (3,3)) on the final layer. The pooling size remained a constant (2,2) and the dropout inputs of the first two layers were 0.2 while the last was 0.4. The flatten layer linearizes the output so the dense layer can process it properly [15, 16]. The rescale layer normalizes the values of each pixel from [0, 255] to [0, 1] to decrease the skew higher values might introduce [17, 18].

F. Training, Validation, and Testing

Before compiling and using the model, all the image arrays were converted into NumPy arrays and all the label arrays were converted into categorical NumPy arrays to comply with the compatible data types of Keras CNNs. The shapes of each array were also checked one last time for easy debugging if the need arose.

The model was compiled using Adam as the optimizer with a learning rate of 0.001. The loss function of the model was Binary Cross entropy while the metrics function was Binary Accuracy as these are the most suitable for binary classification [19, 20]. After compilation, the model was trained using the Keras fit function with the training images and labels as the x and y variables respectively and the validation data using the validation images and labels. The epochs were limited to 5 for two reasons: the main reason was to preserve allocated RAM, and then it was observed during testing that the accuracy increases and loss decreases were plateauing at around the 5 epoch mark. The batch size was set to 100 and shuffle was set to true to ensure robustness by avoiding seeing the same order of images repeatedly, which would be detrimental to binary classification.

The model was tested using the testIMG array as the data and the testLABEL array as the respective classification labels. These were passed through the predict function provided by the Keras library. These were outputted as probabilities of the image falling under class 1 (fractured) instead of the actual predicted labels, so a threshold of 0.5 was set for the labels and then saved as an integer. The accuracy evaluation was done using the evaluation function and multiplied by 100 to present as a percentage instead of a decimal. Finally, a confusion

matrix was generated based on the prediction and the ground truth.

III. Results and Discussion

A. Processed Data

Images from the database was extracted successfully. Due to the dataset being pre-divided before importing, loading the read data of each image into their own respective arrays of training, testing, and validation was relatively straightforward. An if-else statement had to be implemented to skip over files that come back as a NoneType, which are unable to be resized. The resulting counts and sizes of each set is described in the figure below:

```
training: 9243 9243
testing: 506 506
validating: 829 829
Training set: Training Images = (9243, 256, 256, 3), Training Labels = (9243, 2)
Validation set: Validation Images = (829, 256, 256, 3), Validation Labels = (829, 2)
Testing set: Training Images (506, 256, 256, 3), Training Labels = (506, 2)
```

Fig. 1. Training images. (*a. resized and denoised image, b. image masks, c. images after CLAHE*)

A total of 10,578 images were extracted. Due to the resizing of the images, the shapes of the images were consistent, with each set having a shape of (x, 256, 256, 3) where x is the number of images in the array. The labels were also consistent with the shape being (x, 2). This served as the basis for the output shapes in the creation of the CNN model.

The x-ray images were grayscaled, denoised using Gaussian blur, masked using the OTSU algorithm, and finally underwent CLAHE to increase the contrast of the image. This was done per image array for easier display and testing of the code. The first five images of each process and image array are as follows:

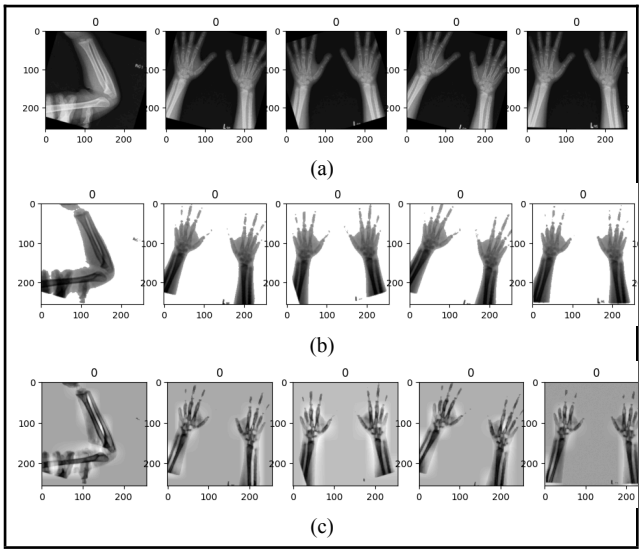

Fig. 2. Training images. (*a. resized and denoised image, b. image masks, c. images after CLAHE*)

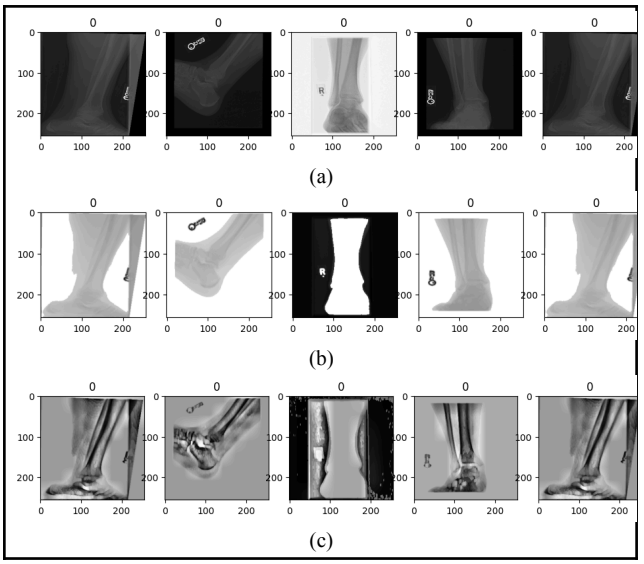

Fig. 3. Testing images. (*a. resized and denoised image, b. image masks, c. images after CLAHE*)

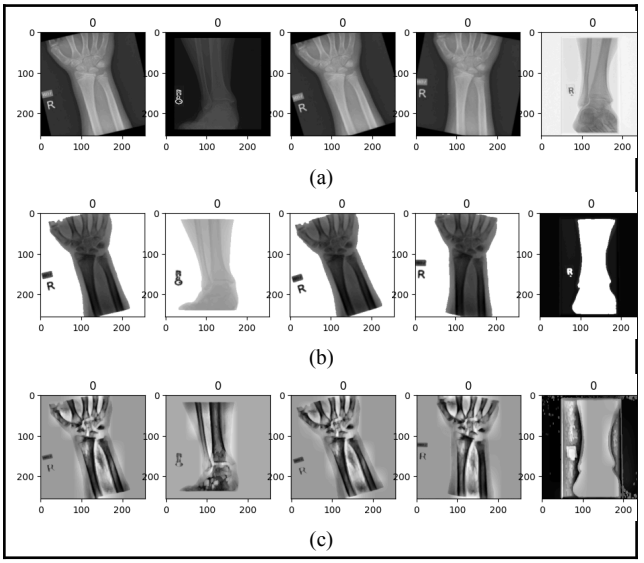

Fig. 4. Testing images. (*a. resized and denoised image, b. image masks, c. images after CLAHE*)

Most of the images were successfully processed, however, some X-rays such as the third image in Fig. 2 or the fifth image in Fig 3. ended up becoming less clear due to the histogram equalization taking into account the entire limb instead of just the bones, creating a solid white graphic. This was most likely caused by the orientation of the limb, which has multiple bones stacking on top of each other in the X-ray. The individual bones therefore had less clarity in their contours, and the resulting image was flattened. Another possible reason could be how the background is much lighter than other X-rays, however going back and processing each image with this layout to have a dark background instead for consistency would be unrealistic due to the size of the database.

B. Finalized Model

After declaring the layers of the CNN, the model was assembled. The table summary is seen below in Table 1:

TABLE I. MODEL LAYERS

Model: "sequential"		
Layer (type)	***Output Shape***	***Param #***
conv2d (Conv2D)	(None, 256, 256, 3)	84
max_pooling2d (MaxPooling2D)	(None, 128, 128, 3)	0
dropout (Dropout)	(None, 128, 128, 3)	0
conv2d_1 (Conv2D)	(None, 128, 128, 64)	1,792
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_1 (Dropout)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
dropout_2 (Dropout)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 128)	16,777,344
dropout_3 (Dropout)	(None, 128)	0
flatten_1	(None, 128)	0
rescaling (Rescaling)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

The resulting model had a total of 16,853,334 parameters with a size of 64.29 MB, all of which were trainable.

C. Model Testing and Accuracy

The accuracy of the model returned as 99.21%, indicating that the model was successful in accurately classifying which X-rays had bone fractures and which X-rays did not. Shown below is the generated confusion matrix:

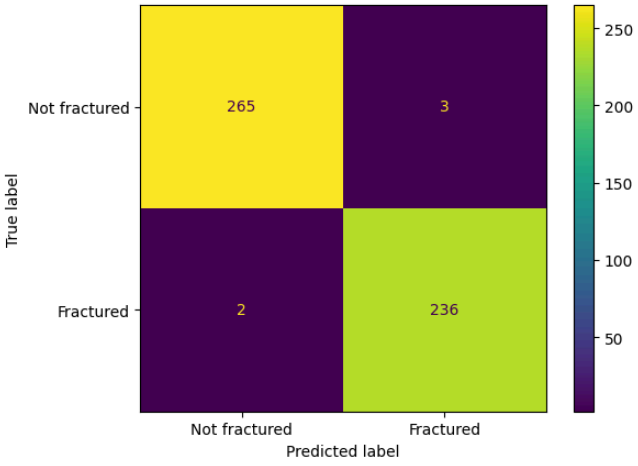

Fig. 5. Confusion Matrix

The model was able to accurately classify 265 out of 268 images as not fractured and 236 out of 238 images as fractured, with 98.88% and 99.16% as their success rates respectively.

The full code can be found in this GitHub repository: [BEELEC1-IndivProj/BEELEC1_IndivProj.ipynb at main · REYES-AFMR/BEELEC1-IndivProj · GitHub]

REFERENCES

[1] A. Pinto *et al.*, "Traumatic fractures in adults: missed diagnosis on plain radiographs in the Emergency Department.," *PubMed*, vol. 89, no. 1-S, pp. 111–123, Jan. 2018, doi: 10.23750/abm.v89i1-s.7015.

[2] L. Leapo, M. Uemura, M. C. Stahl, N. Patil, J. Shah, and T. Otteson, "Efficacy of X-ray in the diagnosis of pediatric nasal fracture," *International Journal of Pediatric Otorhinolaryngology*, vol. 162, p. 111305, Sep. 2022, doi: 10.1016/j.ijporl.2022.111305.

[3] S. M. N. Fathima, R. Tamilselvi, and M. P. Beham, "XSITRAY: a database for the detection of osteoporosis condition," *Biomedical & Pharmacology Journal*, vol. 12, no. 1, pp. 267–271, Mar. 2018, doi: 10.13005/bpj/1637.

[4] T. Aldhyani *et al.*, "Diagnosis and detection of bone fracture in radiographic images using deep learning approaches," *Frontiers in Medicine*, vol. 11, Jan. 2025, doi: 10.3389/fmed.2024.1506686.

[5] S. A. Shifani, G. Ramkumar, A. M. Clemencia, S. Maheswari, and S. Priyadharshini, "Identification of Bone Fragmentation in X-Ray Images using Contour Detection Algorithm," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 9, pp. 2121–2124, Jul. 2019, doi: 10.35940/ijitee.i7851.078919.

[6] G. Moon, S. Kim, W. Kim, Y. Kim, Y. Jeong, and H.-S. Choi, "Computer Aided Facial Bone Fracture Diagnosis (CA-FBFD) system based on object detection model," *IEEE Access*, vol. 10, pp. 79061–79070, Jan. 2022, doi: 10.1109/access.2022.3192389.

[7] H. Sun *et al.*, "Automated rib fracture detection on chest X-Ray using contrastive learning," *Journal of Digital Imaging*, vol. 36, no. 5, pp. 2138–2147, Jul. 2023, doi: 10.1007/s10278-023-00868-z.

[8] M. Rodrigo, "Bone Fracture Multi-Region X-ray Data." Apr. 23, 2024. [Kaggle]. Available: https://www.kaggle.com/datasets/bmadushanirodrigo/fracture-multi-region-x-ray-data/code?datasetId=4854718&sortBy=voteCount

[9] K. Team, "Keras documentation: Getting started with Keras." https://keras.io/getting_started/?fbclid=IwZXh0bgNhZW0CMTAAAR03AkkHNjOxFrylW7yn13t16DVfrhnslyX0imXEq2MmeudQuBxiJIucfUU_aem_-TnzYXoH_avSpIaFuwgutg

[10] GeeksforGeeks, "Image Resizing using OpenCV | Python," *GeeksforGeeks*, Aug. 09, 2024. https://www.geeksforgeeks.org/image-resizing-using-opencv-python/

[11] K. Team, "Keras documentation: The Sequential class." https://keras.io/api/models/sequential/?fbclid=IwZXh0bgNhZW0CMTAAAR3wuKdq6c8bcKiWv87LYrGRcMUlKoR_wMPE1abBZm3movtEKgMswRI6Z58_aem_dLO5mfs0TeqLJ3NPYM4FmA

[12] K. Team, "Keras documentation: Image classification from scratch." https://keras.io/examples/vision/image_classification_from_scratch/?fbclid=IwZXh0bgNhZW0CMTAAAR34fxQzL-nn2_l9p3Sq3KaZ7fF4IE4j0d_Vk0-PTa2v67gvY-Zd0gQdQm4_aem_EcJ_Rme6sjFT41hgTFFPdw#introduction

[13] A. Deshpande, "A Beginner's Guide to understanding convolutional neural networks." https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[14] A. Deshpande, "A Beginner's Guide to Understanding Convolutional Neural Networks Part 2."

https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[15] "Dense Layer vs convolutional layer - when to use them and how," *Data Science Stack Exchange*. https://datascience.stackexchange.com/questions/85582/dense-layer-vs-convolutional-layer-when-to-use-them-and-how?fbclid=IwZXh0bgNhZW0CMTAAAR0-VcpYk1HcAoKfXsIu0LBbqctIhxeHkLS6G-e_C0bCJH3n1VKSwNtPJz0_aem_KU14QSDzQmYeTiiTESdb-A

[16] "When to use Dense, Conv1/2D, Dropout, Flatten, and all the other layers?," *Data Science Stack Exchange*. https://datascience.stackexchange.com/questions/44124/when-to-use-dense-conv1-2d-dropout-flatten-and-all-the-other-layers?fbclid=IwZXh0bgNhZW0CMTAAAR1DNy3YMXkbj2js7Hh9VmMRsJYEO9f53EXdpBknX9pB4g5uZX42bjx0p8E_aem_LHTPt8TRuFTPrs4eEfay0A

[17] "Rescale parameter in data augmentation," *Data Science Stack Exchange*. https://datascience.stackexchange.com/questions/92499/rescale-parameter-in-data-augmentation?fbclid=IwZXh0bgNhZW0CMTAAAR2hnVom07skHCX-TarLuK13x_oVkTweFUgQDtwKSPXlXLownQT7bwL6bWg_aem_7KlgZeF7SY-sxCqBpAkRyw

[18] "Data augmentation," *TensorFlow*. https://www.tensorflow.org/tutorials/images/data_augmentation?fbclid=IwZXh0bgNhZW0CMTAAAR1DNy3YMXkbj2js7Hh9VmMRsJYEO9f53EXdpBknX9pB4g5uZX42bjx0p8E_aem_LHTPt8TRuFTPrs4eEfay0A

[19] "How do I determine the binary class predicted by a convolutional neural network on Keras?," *Stack Overflow*. https://stackoverflow.com/questions/52018645/how-do-i-determine-the-binary-class-predicted-by-a-convolutional-neural-network

[20] K. Team, "Keras documentation: Probabilistic losses." https://keras.io/api/losses/probabilistic_losses/#binarycrossentropy-class