X-ray Bone Fracture Detection: Enhancing Medical Diagnosis with Convolutional Neural Networks

Muhammad Tahir Zia
Bachelors Computer Engineering
(GIK Institute)
Topi, Pakistan
u2021465@giki.edu.pk

Syed Roshan Ali
Bachelors Computer Engineering
(GIK Institute)
Topi, Pakistan
u2021648@giki.edu.pk

Akhtar Ali
Bachelors Computer Engineering
(GIK Institute)
Topi, Pakistan
u2021758@giki.edu.pk

***Abstract*—Early and accurate detection of bone fractures is crucial for timely treatment and improved patient outcomes. This paper presents a novel approach for X-ray bone fracture detection using a Convolutional Neural Network (CNN) architecture built with TensorFlow Keras. The proposed model leverages the power of CNNs to automatically extract and learn discriminative features from X-ray images, facilitating the classification of fractured and non-fractured bones.**

The model architecture incorporates a series of convolutional layers with ReLU activation functions, effectively learning spatial features from the input images. Max-pooling layers are employed for dimensionality reduction and enhancing the model's focus on prominent features. Flatten layers connect the convolutional layers to fully-connected layers, where higher-level features are extracted for classification. The final layer utilizes a single neuron with sigmoid activation to predict the presence or absence of a fracture, resulting in a binary classification output.

To improve the model's generalization capabilities and robustness to variations in X-ray images, a comprehensive pre-processing stage is implemented using ImageDataGenerator. Training data augmentation techniques, including rescaling, random shearing, zooming, and horizontal flipping, are applied to artificially expand the dataset and enhance its diversity. Testing data undergoes normalization using the same rescaling technique to ensure consistency with the training data. Data is loaded efficiently using the "flow-from-directory" function, facilitating image loading and batch processing during training.

This work demonstrates the potential of CNNs for accurate X-ray bone fracture detection. The proposed model and pre-processing techniques pave the way for the development of robust and automated diagnostic tools to aid medical professionals in fracture identification, leading to improved patient care.

I. Introduction

Bone fractures are a prevalent medical concern, affecting individuals of all ages due to accidents, falls, or underlying medical conditions. Early and accurate detection of fractures is critical for timely treatment interventions, such as bone setting, casting, or surgery, to promote proper healing and minimize potential complications. Traditional methods for fracture detection rely on X-ray imaging followed by expert interpretation by radiologists. While X-rays are a cornerstone diagnostic tool, the process can be time-consuming, and accuracy might be subject to inter-observer variability.

Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), have opened new avenues for automating medical image analysis tasks. CNNs excel at extracting and learning hierarchical features from visual data, including medical images like X-rays. This capability makes them well-suited for automated fracture detection, offering potential benefits in terms of:

- CNN-based models can analyze X-ray images rapidly, potentially reducing the time required for fracture diagnosis.
- By learning from large datasets of labeled X-ray images, CNNs can achieve high accuracy in fracture detection, potentially surpassing human performance in some cases.
- CNNs provide objective and consistent results, minimizing the impact of inter-observer variability in radiologist interpretation.

This paper proposes a novel approach for X-ray bone fracture detection utilizing a CNN architecture built with TensorFlow Keras. The model is designed to automatically analyze X-ray images and classify them as fractured or non-fractured. We employ a comprehensive pre-processing pipeline with data augmentation techniques to improve the model's generalization capabilities and robustness to variations in X-ray images.

The remainder of this paper is structured as follows. Section 2 describes the proposed CNN architecture and its components. Section 3 details the pre-processing steps undertaken to prepare the X-ray data for model training. Section 4 presents the experimental setup, training process, and evaluation metrics. Section 5 discusses the results of the model and highlights its performance. Finally, Section 6 concludes the paper by summarizing the key findings and outlining potential future directions for this research.

II. Methodology

A. Dataset

Accessing diverse and well-curated datasets is pivotal for the success of any deep learning project. Kaggle, a prominent platform for data science competitions and collaborative research, emerges as an invaluable resource for sourcing datasets tailored to various machine learning tasks. The Kaggle platform hosts a vibrant community of data scientists and researchers, fostering an environment where datasets are shared, discussed, and continually enriched.

Comprising over 9,400 meticulously labeled X-ray scans, this repository provided a comprehensive platform to investigate the classes of Fractured and not Fractured.

The dataset, comprising approximately 9,400 X-ray scans, was already divided into pre-defined training and validation sets, ensuring a structured approach to model development and evaluation.

B. Data Preprocessing

1) File Path Creation: To streamline the access and manipulation of image data, file paths for the dataset were systematically created and joined using the os.path.join method. This approach ensures a consistent and platform-independent method for concatenating directory and file names, laying the foundation for a well-structured dataset management system.

2) Creation of Dataframes: Once the dataset paths were established, the next step involved creating dataframes for both the training and validation datasets. This organizational strategy not only simplifies data manipulation but also facilitates efficient indexing and retrieval of relevant information during the model training process. The use of dataframes enhances the ease of access to image paths, corresponding labels, and any additional metadata, promoting a coherent and comprehensible dataset structure.

```
# Create an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1.0/255,        # Normalize pixel values between 0 and 1
    shear_range=0.2,        # Apply random shear transformations
    zoom_range=0.2,         # Apply random zoom transformations
    horizontal_flip=True)  # Flip images horizontally

# Only normalize pixel values for testing
test_datagen = ImageDataGenerator(rescale=1.0/255)

# Load and augment training data
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

# Load and normalize testing data
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')
```

C. Building the Model

This subsection delves into the design and architecture of the Convolutional Neural Network (CNN) model.

The proposed CNN architecture comprises several key layers.

1) Convolutional Layers: The model utilizes multiple convolutional layers at the beginning of the architecture. These layers are equipped with learnable filters that scan the input image and extract low-level features such as edges, shapes, and textures.

2) Pooling Layers: Following the convolutional layers, pooling layers are employed for dimensionality reduction and enhancing the model's focus on prominent features. These layers typically perform downsampling operations like max pooling, which retains the maximum value from a specific region in the feature map. This reduces the spatial resolution of the data while preserving important information.

```
# Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                           input_shape=(img_width, img_height, 3)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

3) Flatten Layers: After processing by the convolutional and pooling layers, the feature maps are transformed into a one-dimensional vector by the flatten layer. This allows the model to connect the extracted spatial features to fully-connected layers for higher-level reasoning and classification.

4) Fully Connected Layers: The flattened feature vector is then fed into fully-connected layers. These layers contain neurons that are fully connected to all neurons in the previous layer, fostering the integration of extracted features. The number of neurons and the number of layers in this section are crucial hyperparameters that can be tuned to optimize model performance.

5) Output Layer: The final layer of the model is the output layer, containing a single neuron or a set of neurons depending on the number of emotions you aim to classify. In the case of single-neuron output with sigmoid activation, the model predicts the probability of a particular emotion being present in the image. Alternatively, for multi-class classification with multiple output neurons and softmax activation, the model outputs a vector of probabilities, indicating the likelihood of each emotion being present.

6) Model Training: Once the model architecture is defined, the training process commences. During training, the model iteratively adjusts its internal parameters (weights and biases) to minimize the difference between its predictions and the true labels of the training data. This optimization process is typically guided by a chosen loss function and an optimizer algorithm.

7) Effective Training: The following aspects are crucial for effective training:

- Data Augmentation: Techniques like random cropping, flipping, and color jittering can be employed to artificially expand the training data and improve the model's generalization capabilities.
- Loss Function: The choice of loss function depends on the classification task (e.g., binary cross-entropy for single-emotion or categorical cross-entropy for multi-class). This function measures the discrepancy between the model's predictions and the true labels.

- Optimizer Algorithm: Algorithms like Adam or SGD (Stochastic Gradient Descent) with momentum are commonly used to update the model's weights and biases during training, minimizing the chosen loss function.

By carefully designing the model architecture, selecting appropriate training parameters, and implementing effective pre-processing techniques, we aim to achieve a robust and accurate X-ray detection model.

D. Model Compilation

1) Model Compilation and Evaluation Metrics: After defining the CNN architecture using a sequential model from TensorFlow Keras, the next crucial step is model compilation. Compilation configures the training process by specifying the optimizer, loss function, and evaluation metrics.

2) Optimizer Selection: The chosen optimizer dictates how the model updates its internal parameters (weights and biases) during training. In this work, we employ the Adam (Adaptive Moment Estimation) optimizer. Adam is a popular choice due to its efficiency and effectiveness in optimizing a wide range of neural network architectures. It combines the benefits of both gradient descent and momentum, making it well-suited for addressing issues like vanishing gradients and local minima.

3) Loss Function: The loss function measures the discrepancy between the model's predictions and the true labels of the training data. It serves as a guide for the optimizer, indicating how well the model is performing and which direction parameter updates should take to minimize the error. In this case, we utilize the binary cross-entropy loss function.

This function is suitable for binary classification tasks, such as our X-ray fracture detection problem, where the model aims to classify images as either fractured (positive class) or non-fractured (negative class). Binary cross-entropy measures the average difference between the model's predicted probabilities for each class (fractured and non-fractured) and the actual labels (0 for non-fractured, 1 for fractured).

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# Train the model
model.fit(train_generator, epochs=epochs, callbacks=[tensorboard_callback])
```

E. Evaluation Metrics

While the loss function plays a crucial role during training, it's essential to evaluate the model's performance on unseen data after training is complete. Here, we utilize the following metrics to assess the model's effectiveness in X-ray fracture detection:

- Accuracy: This metric represents the proportion of correctly classified images by the model. It's calculated as the total number of correctly predicted images divided by the total number of test images. While accuracy provides a general overview of model performance, it can be misleading in imbalanced datasets.
- Sensitivity (Recall): This metric focuses on the model's ability to correctly identify fractured cases (positive class). It's calculated as the proportion of true positive predictions (correctly classified fractured images) divided by the total number of actual fractured cases in the test set. A high sensitivity signifies the model's effectiveness in not missing fracture cases.
- Specificity: This metric evaluates the model's ability to correctly identify non-fractured cases (negative class). It's calculated as the proportion of true negative predictions (correctly classified non-fractured images) divided by the total number of actual non-fractured cases in the test set. High specificity indicates the model's capability to avoid false alarms for non-fractured images.

By considering both accuracy and these additional metrics, we gain a more comprehensive understanding of the model's performance in classifying fractured and non-fractured X-ray images.

III. Results

The Results section of this journal presents a comprehensive examination of the outcomes of our research, delving into the meticulously conducted analyses, model evaluations, and insightful discoveries. At the Gradio GUI, the model predicts perfectly well.

IV. Conclusion

Our exploration of X-ray Imaging has yielded valuable insights. We have not only evaluated existing models, but also constructed our own, pushing accuracy and robustness as much as we can with the limitations in place. However, this is merely the foundation upon which future advancements will be built.

Moving forward, several key avenues present themselves for further exploration. Expanding and diversifying datasets to encompass a wider range of Images is crucial to develop universally applicable technology. Refining the model architecture through innovative neural networks and transfer learning techniques holds the potential for enhanced precision and recognition.

REFERENCES

[1] O. Bandyopadhyay, A. Biswas, and B. B. Bhattacharya, "Long-bone fracture detection in digital x-ray images based on digital-geometric techniques," *Computer methods and programs in biomedicine*, vol. 123, pp. 2–14, 2016.

[2] F. Hardalaç, F. Uysal, O. Peker, M. Çiçeklidağ, T. Tolunay, N. Tokgöz, U. Kutbay, B. Demirciler, and F. Mert, "Fracture detection in wrist x-ray images using deep learning-based object detection models," *Sensors*, vol. 22, no. 3, p. 1285, 2022.

[3] L. Tanzi, E. Vezzetti, R. Moreno, and S. Moos, "X-ray bone fracture classification using deep learning: a baseline for designing a reliable approach," *Applied Sciences*, vol. 10, no. 4, p. 1507, 2020.

[4] T. Meena and S. Roy, "Bone fracture detection using deep supervised learning from radiological images: A paradigm shift," *Diagnostics*, vol. 12, no. 10, p. 2420, 2022.

[5] A. Barhoom, M. R. Jubair, and S. S. Abu-Naser, "A survey of bone abnormalities detection using machine learning algorithms," in *AIP Conference Proceedings*, vol. 2808, no. 1. AIP Publishing, 2023.

[1] [2] [3] [4] [5]