Deep Learning-Based Bone Fracture Classification Using Convolutional Neural Networks

Abstract:
This study presents a deep learning approach for automated bone fracture classification using convolutional neural networks (CNNs). The proposed method leverages the power of CNNs to extract discriminative features from X-ray images, enabling accurate classification of fracture types. A dataset comprising X-ray images of various bone fractures is utilized for model training and evaluation. The CNN architecture consists of multiple convolutional and pooling layers, followed by fully connected layers for classification. Experimental results demonstrate the effectiveness of the proposed approach in accurately classifying bone fractures, achieving high classification accuracy and performance metrics. The developed CNN model offers a promising solution for automating the classification of bone fractures, facilitating faster diagnosis and treatment planning in clinical settings

Introduction:

Bone fracture detection is a critical aspect of medical diagnostics, playing a vital role in timely and accurate treatment planning. Automated detection using deep learning techniques can significantly enhance the efficiency and accuracy of this process, reducing the burden on radiologists and improving patient outcomes. In this project, we utilize three prominent deep learning models—CNN, ResNet50, and VGG16—to classify bone fractures from X-ray images.

The Convolutional Neural Network (CNN) is a foundational deep learning architecture known for its ability to automatically learn and extract hierarchical features from images. Its layers of convolutional filters, pooling operations, and activation functions enable it to capture intricate patterns and structures within the data, making it highly effective for image classification tasks.

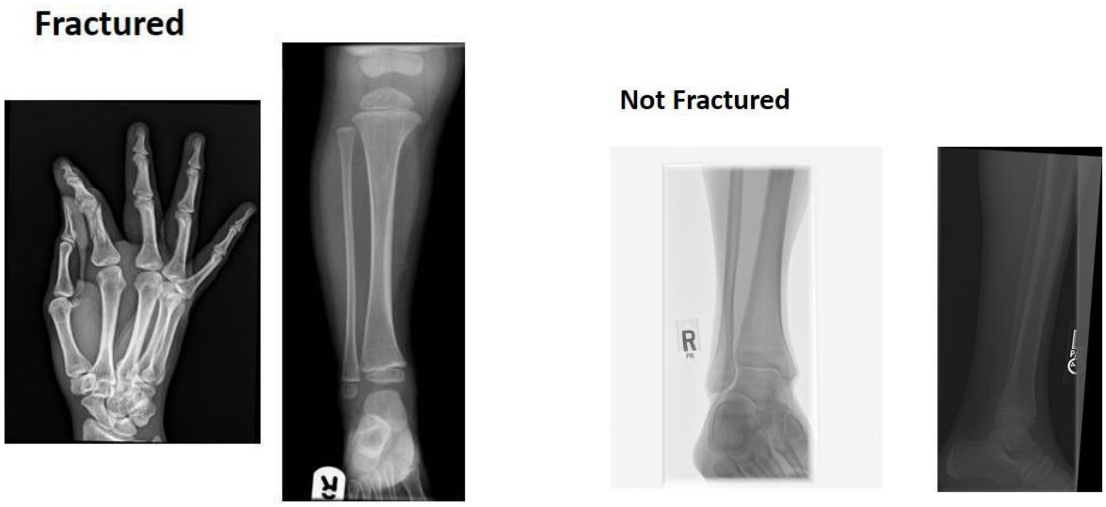
ResNet50, or Residual Network, is a deeper architecture that introduces residual learning, allowing it to train much deeper networks without suffering from the vanishing gradient problem. By incorporating shortcut connections, ResNet50 can effectively learn complex features, making it a robust choice for high-performance image recognition tasks, including medical image analysis.

VGG16 is another influential deep learning model renowned for its simplicity and depth. It consists of 16 layers and utilizes small convolutional filters (3x3), which, despite their simplicity, capture fine details and achieve high accuracy. VGG16's architecture is known for its uniformity and depth, making it a reliable model for various image classification tasks.

By leveraging these three models, this project aims to develop a comprehensive approach to bone fracture detection, comparing their performance and identifying the most effective model for clinical application. Through this study, we seek to advance the field of automated medical diagnostics, contributing to faster and more accurate bone fracture detection.

Dataset Description

The dataset utilized in this project comprises bone fracture images meticulously partitioned into training, validation, and test sets. With 9246 images in the training set, 828 in the validation set, and 506 in the test set, it offers a diverse and substantial collection for model development and evaluation. The training data is crucial for training the convolutional neural network (CNN) models, providing ample examples for effective feature learning and parameter optimization. The validation set, consisting of a moderate-sized collection, facilitates hyperparameter tuning and prevents overfitting during model training. Finally, the test set, reserved exclusively for final evaluation, ensures an unbiased assessment of model performance on unseen data, thereby validating the models' generalization ability to real-world bone fracture images. Through preprocessing and augmentation techniques, the dataset is enriched to enhance model robustness and efficacy in accurately classifying bone fracture types.



METHODOLOGY:

The goal of this project is to develop a robust and accurate bone fracture detection system using three deep learning models: a custom Convolutional Neural Network (CNN), ResNet50, and VGG16. The dataset used consists of bone fracture images split into training (9246 images), validation (828 images), and test (506 images) sets. The methodology includes data preprocessing, model definition, training, and evaluation. Below are the detailed steps and the structure of each model, including their layer-wise input and output sizes.

Data Preprocessing and Augmentation

The images are preprocessed to standardize their size and normalize their pixel values. The following transformations are applied:

- **Grayscale Conversion**: Convert grayscale images to three channels.
- **Resize**: Resize all images to 224x224 pixels.
- **Data Augmentation (Training Set)**: Apply random horizontal and vertical flips, rotations, and color jittering to enhance the diversity of the training data.
- **Normalization**: Normalize the images using mean **[0.485, 0.456, 0.406]** and standard deviation **[0.229, 0.224, 0.225]**.

Models

We utilized three different models for bone fracture detection: a custom CNN, ResNet50, and VGG16. Each model was modified to output two classes, indicating the presence or absence of a fracture.

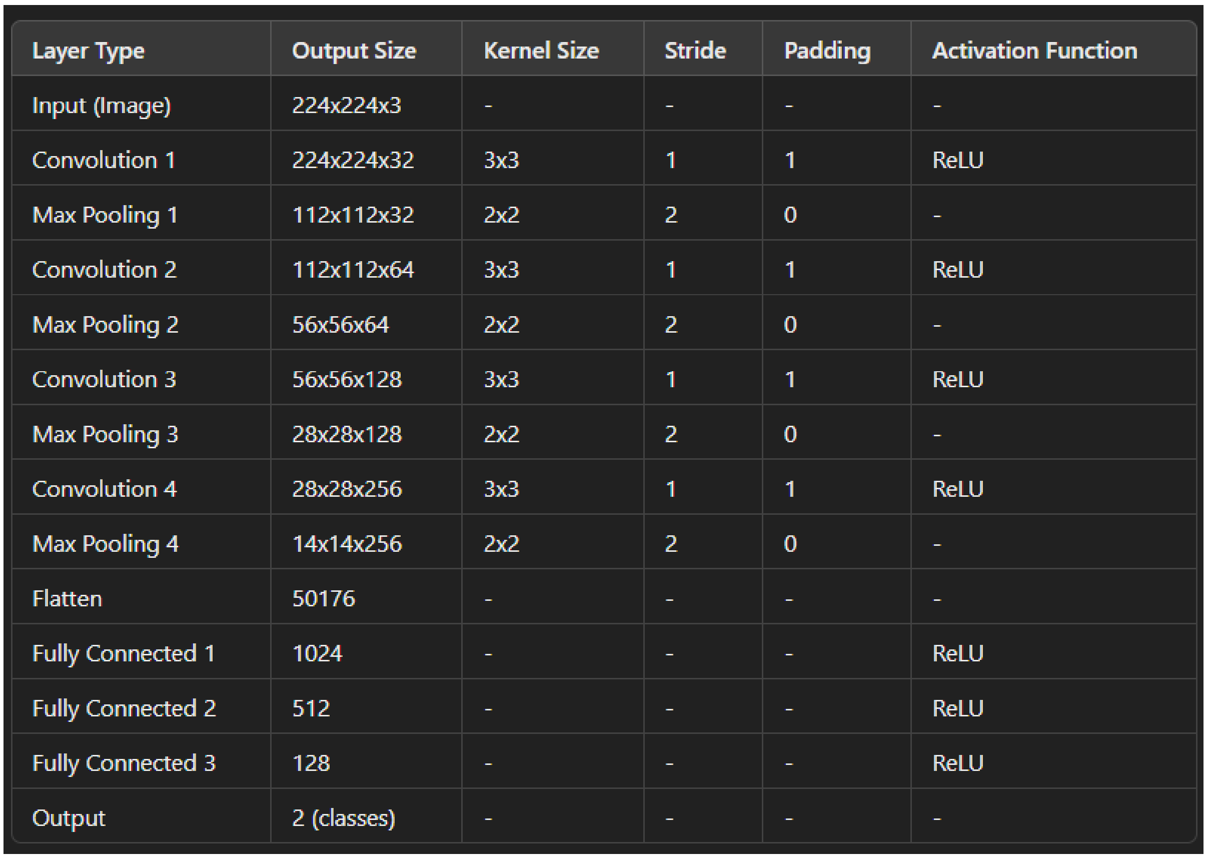
1. Custom Convolutional Neural Network (CNN)

The custom CNN consists of several convolutional layers followed by batch normalization, ReLU activation, and max-pooling layers. The structure is as follows:

- **Conv Layer 1**: Input: [batch_size, 3, 224, 224], Output: [batch_size, 32, 224, 224]
- **Max Pool 1**: Input: [batch_size, 32, 224, 224], Output: [batch_size, 32, 112, 112]
- **Conv Layer 2**: Input: [batch_size, 32, 112, 112], Output: [batch_size, 64, 112, 112]
- **Max Pool 2**: Input: [batch_size, 64, 112, 112], Output: [batch_size, 64, 56, 56]
- **Conv Layer 3**: Input: [batch_size, 64, 56, 56], Output: [batch_size, 128, 56, 56]
- **Max Pool 3**: Input: [batch_size, 128, 56, 56], Output: [batch_size, 128, 28, 28]
- **Conv Layer 4**: Input: [batch_size, 128, 28, 28], Output: [batch_size, 256, 28, 28]
- **Max Pool 4**: Input: [batch_size, 256, 28, 28], Output: [batch_size, 256, 14, 14]
- **Flatten**: Input: [batch_size, 256, 14, 14], Output: [batch_size, 256*14*14]

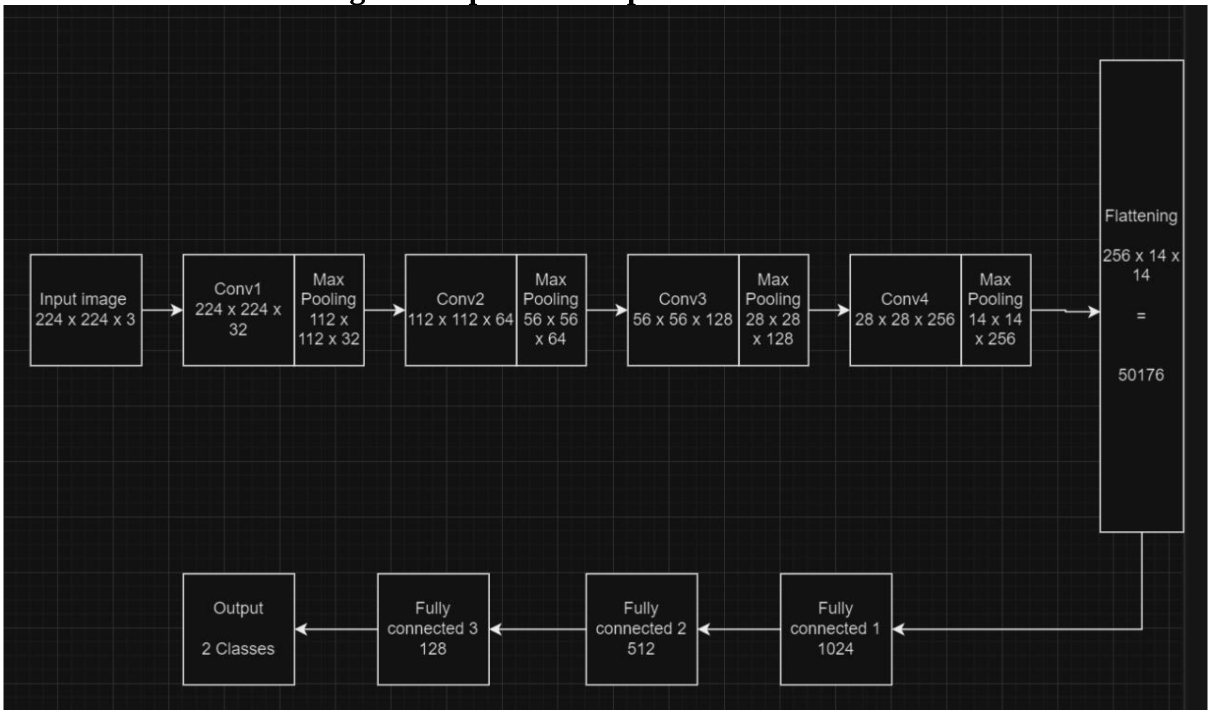
- **Fully Connected Layer 1**: Input: [batch_size, 256*14*14], Output: [batch_size, 1024]
- **Fully Connected Layer 2**: Input: [batch_size, 1024], Output: [batch_size, 512]
- **Fully Connected Layer 3**: Input: [batch_size, 512], Output: [batch_size, 128]
- **Output Layer**: Input: [batch_size, 128], Output: [batch_size, 2]

Architecture of Custom CNN



Layer Type	Output Size	Kernel Size	Stride	Padding	Activation Function
Input (Image)	224x224x3	-	-	-	-
Convolution 1	224x224x32	3x3	1	1	ReLU
Max Pooling 1	112x112x32	2x2	2	0	-
Convolution 2	112x112x64	3x3	1	1	ReLU
Max Pooling 2	56x56x64	2x2	2	0	-
Convolution 3	56x56x128	3x3	1	1	ReLU
Max Pooling 3	28x28x128	2x2	2	0	-
Convolution 4	28x28x256	3x3	1	1	ReLU
Max Pooling 4	14x14x256	2x2	2	0	-
Flatten	50176	-	-	-	-
Fully Connected 1	1024	-	-	-	ReLU
Fully Connected 2	512	-	-	-	ReLU
Fully Connected 3	128	-	-	-	ReLU
Output	2 (classes)	-	-	-	-

Detailing with input and output for CNN architecture



1. Convolutional Layers:
Importance: Convolutional layers are fundamental to CNNs. They learn spatial hierarchies of patterns in the input images by applying convolution operations with learnable filters.
In This Application: The convolutional layers in LeNet-5 extract features like edges, corners, and textures from the input handwritten digit images. These features are critical for distinguishing between different digits.
2. Pooling Layers:
Importance: Pooling layers help reduce the spatial dimensions of the feature maps while retaining important information. They introduce spatial invariance and reduce computational complexity.
In This Application: Average pooling layers in LeNet-5 down sample the feature maps, reducing their size and retaining the most important features. This helps in focusing on the most relevant information for digit recognition.

3. Flatten Layer:
Importance: The flatten layer converts the 2D feature maps from the convolutional layers into a 1D vector. This prepares the data for input to the fully connected layers.
In This Application: After feature extraction and dimensionality reduction, the flatten layer in LeNet-5 ensures that the features are ready for processing by the dense layers.

4. Dense Layers (Fully Connected Layers):
Importance: Dense layers are used for learning non-linear combinations of features extracted by convolutional and pooling layers. They perform classification based on high-level features.
In This Application: The fully connected layers in LeNet-5 perform the final classification of digits. They take the flattened feature vector as input and output the probabilities of each class (0-9).
Activation Functions Used
In the context of neural networks, activation functions introduce non-linearity into the model, allowing it to learn complex patterns and representations. In this project, the following activation functions were used:

1. **ReLU (Rectified Linear Unit)**
2. **Softmax**

1. ReLU (Rectified Linear Unit)
Formula:$\mathrm{ReLU}(x)=\max(0,x)$
Usage: ReLU is used in the convolutional layers of the custom CNN, ResNet50, and VGG16 models. It is applied after every convolutional layer and before max-pooling layers to introduce non-linearity. ReLU is simple and efficient, helping to mitigate the vanishing gradient problem by allowing gradients to pass through unchanged when the input is positive.
Advantages:

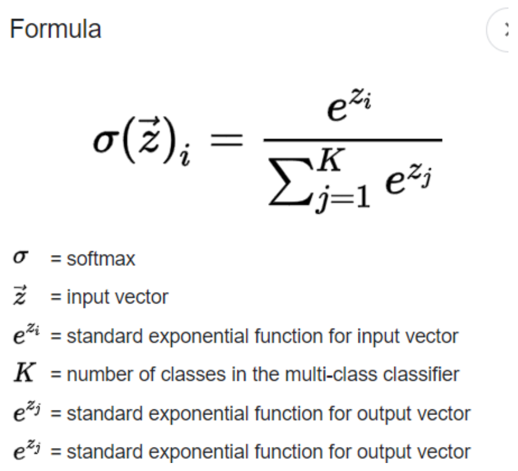
- Efficient computation.
- Helps with gradient-based optimization by mitigating the vanishing gradient problem.
- Sparse activation, which can lead to more efficient representations.

Impact on Models:

- **Custom CNN**: Applied after each convolutional layer to activate the learned features.
- **ResNet50**: Used in residual blocks to maintain efficient gradient flow.
- **VGG16**: Applied after each convolutional and fully connected layer to introduce non-linearity and help the model learn complex patterns.

2. Softmax

Formula:



Formula

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ	= softmax
$\vec{z}$	= input vector
e^{z_i}	= standard exponential function for input vector
K	= number of classes in the multi-class classifier
e^{z_j}	= standard exponential function for output vector
e^{z_j}	= standard exponential function for output vector

Usage: Softmax is used in the output layer of each model to convert the logits into probabilities. This function ensures that the output values are between 0 and 1 and that they sum up to 100%, making it suitable for multi-class classification problems.

Advantages:

- Converts logits to probabilities, which are easier to interpret.
- Ensures outputs are normalized and sum to 1, making them suitable for probability distribution.

2. ResNet50

ResNet50 is a deep residual network with 50 layers. The key modification made to this pre-trained model was in the final fully connected layer:

- **Input Size**: [batch_size, 3, 224, 224]
- **Output Size**: [batch_size, 2]

The ResNet50 model uses residual blocks to enable training of deeper networks by mitigating the vanishing gradient problem. The last fully connected layer is replaced to match the number of output classes for the binary classification task.

Overview

ResNet-50 is a deep convolutional neural network with 50 layers. The model consists of:

- An initial convolutional layer
- A max-pooling layer
- 16 residual blocks
- An average pooling layer
- A fully connected (dense) layer

Key Components

1. Initial Layers

- **Conv1**:
    - 7x7 convolutional layer with 64 filters, a stride of 2, and padding of 3.
    - Followed by a batch normalization and a ReLU activation.
    - Outputs: [batch_size, 64, 112, 112] for an input size of [batch_size, 3, 224, 224].
- **Max Pooling**:
    - 3x3 max pooling with a stride of 2.
    - Outputs: [batch_size, 64, 56, 56].

2. Residual Blocks

ResNet-50 consists of 16 residual blocks, which are grouped into four stages. Each stage processes the data at a different resolution. The blocks can be categorized into two types:

- **Identity Block**: Maintains the same dimensions.
- **Convolutional Block**: Changes the dimensions (e.g., spatial resolution or number of filters).

Each block consists of three convolutional layers.

Stage 1:

- 3 residual blocks, each with 64 filters.
    - Conv Block 1:
        - [batch_size, 256, 56, 56] -> [batch_size, 256, 56, 56] (dimensions maintained).
- **Stage 2**:
    - 4 residual blocks, each with 128 filters.
        - Conv Block 2:
            - [batch_size, 128, 28, 28] -> [batch_size, 512, 28, 28].
    - Dimensionality reduction with strides and projection shortcut.
- **Stage 3**:
    - 6 residual blocks, each with 256 filters.
        - Conv Block 3:
            - [batch_size, 256, 14, 14] -> [batch_size, 1024, 14, 14].
    - Similar to Stage 2 but deeper.
- **Stage 4**:
    - 3 residual blocks, each with 512 filters.
        - Conv Block 4:
            - [batch_size, 512, 7, 7] -> [batch_size, 2048, 7, 7].
    - Follows similar principles as previous stages.

3. Bottleneck Design

Each residual block (both identity and convolutional) follows a bottleneck design:

- **Conv Layer 1**: 1x1 convolution to reduce the number of channels (down-sampling).
- **Conv Layer 2**: 3x3 convolution for spatial feature extraction.
- **Conv Layer 3**: 1x1 convolution to restore the number of channels (up-sampling).
- Each convolution is followed by batch normalization and ReLU activation.
- A shortcut connection bypasses these convolutions and is added to the output.

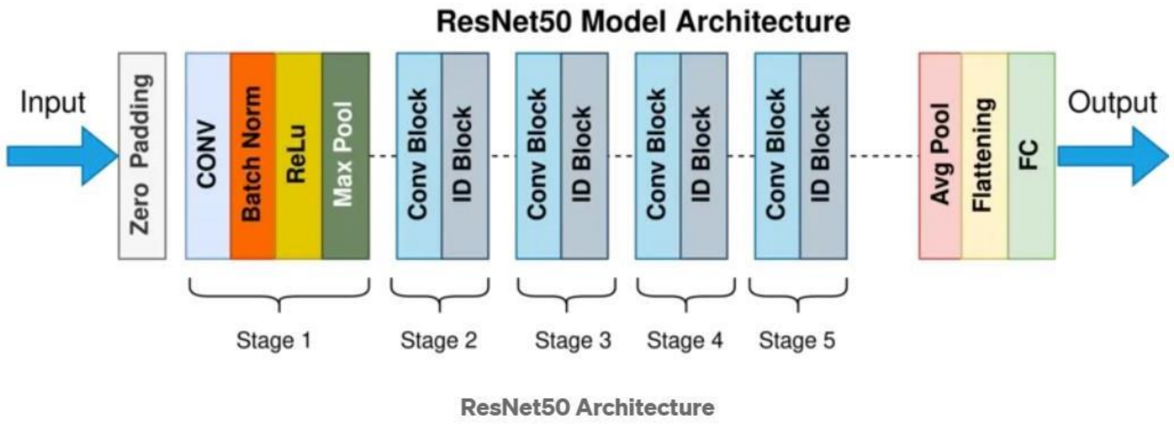
4. Final Layers

Average Pooling:

Global average pooling reduces the spatial dimensions to 1x1.
Outputs: [batch_size, 2048].
Fully Connected Layer:

A dense layer with the number of units equal to the number of classes (e.g., 1000 for ImageNet).
Outputs: [batch_size, 1000]

RESNET50 for Binary Classification



3. VGG16
VGG16 is a deep network with 16 layers, known for its simplicity and uniform architecture. The modifications include:

- **Input Size**: [batch_size, 3, 224, 224]
- **Output Size**: [batch_size, 2]

The classifier part of VGG16 is replaced with a custom sequence of fully connected layers ending with an output layer for binary classification.
VGG16 is a convolutional neural network architecture named after the Visual Geometry Group (VGG) at the University of Oxford. It was introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014. VGG16 is characterized by its depth, as it consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. Here's a breakdown of the VGG16 architecture:
Architecture:

1. **Input Layer**:
   - VGG16 takes an input image with dimensions 224x224x3 (RGB channels).
2. **Convolutional Blocks**:
   - VGG16 consists of 13 convolutional layers, each followed by a ReLU activation function, and 5 max-pooling layers interspersed throughout the architecture.
   - The convolutional layers use small 3x3 filters with a stride of 1 and same padding, resulting in a receptive field that can capture small spatial features.
   - The max-pooling layers use 2x2 filters with a stride of 2, reducing the spatial dimensions of the feature maps while increasing the receptive field.
3. **Fully Connected Layers**:
   - After the convolutional blocks, VGG16 has three fully connected layers, also known as dense layers.
   - The first two fully connected layers have 4096 neurons each, followed by a ReLU activation function and dropout regularization.
   - The final fully connected layer has 1000 neurons (corresponding to the 1000 classes in the ImageNet dataset), followed by a softmax activation function to output class probabilities.
4. **Output Layer**:
   - The output layer produces the final classification probabilities for the input

image across the 1000 ImageNet classes.

Key Features:

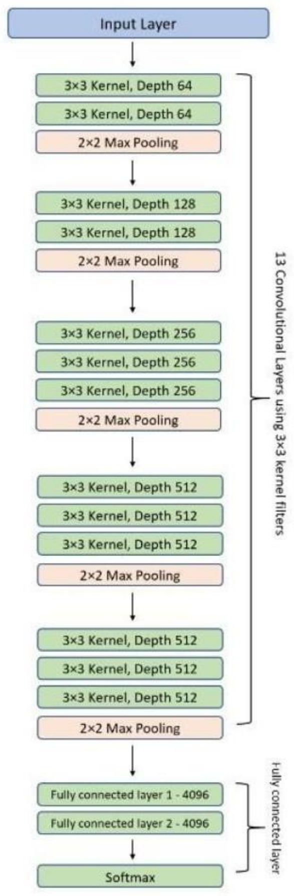
- **Uniform Architecture**: VGG16 has a uniform architecture, with small 3x3 filters and max-pooling layers throughout. This simplicity contributes to its ease of understanding and implementation.
- **Deep Network**: VGG16 is deeper than previous CNN architectures, which allows it to capture more complex features and patterns in images.
- **Pre-Trained Models**: Pre-trained VGG16 models trained on large datasets like ImageNet are widely used for transfer learning tasks, where the model is fine-tuned on smaller datasets for specific tasks like object detection or image classification.

Applications:

- Image Classification: VGG16 is often used as a base model for image classification tasks due to its strong performance on benchmarks like ImageNet.
- Feature Extraction: The convolutional layers of VGG16 can be used to extract features from images for tasks like object detection, segmentation, and image retrieval.

VGG16 has been influential in the development of deep learning and convolutional neural networks, serving as a baseline architecture for subsequent models with increased depth and complexity. Despite being surpassed by newer architectures in terms of performance, VGG16 remains a popular choice for its simplicity and effectiveness.

VGG16 Architecture:



VGG-16 Architecture of a VGG16 model

Training and Evaluation
The models were trained using the CrossEntropyLoss function and optimized using the Adam optimizer. Learning rate schedulers with a step size of 7 and gamma of 0.1 were used to adjust the learning rate during training. The training was conducted over 10 epochs. The evaluation involved calculating the precision, recall, F1 score, and plotting the confusion matrix for each model. The training loss was recorded and visualized to ensure the models' convergence.

Experimental Results and Inferences:

1. Precision:
Precision measures the accuracy of the positive predictions. It is calculated as the ratio of true positive predictions to the total number of positive predictions made by the model.

$$\text{Precision}=\frac{\text{True Positives}}{True\ Positives+False\ Positives}$$

- **Inference**: Precision values are high for all classes, indicating that the model has a low false positive rate. For example, for class 1, the precision is 1.00, which means that when the model predicts an image as class 1, it is correct 100% of the time.

2. Recall:
Recall measures the ability of the model to find all the relevant cases within a dataset. It is calculated as the ratio of true positive predictions to the total number of actual positive instances in the data.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$$

- **Inference**: Recall values are also high for all classes, indicating that the model captures most of the positive instances. For example, for class 1, the recall is 0.99, meaning that the model correctly identifies 99% of all actual class 1 instances.

3. F1-score:
The F1-score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.

$$F1 - score = 2 \times \frac{Precision \times Recalles}{Precision + Recall}$$

- **Inference**: F1-score values are high for all classes, indicating a good balance between precision and recall. A high F1-score suggests that the model has both low false positives and low false negatives.

4. Support:
Support is the number of actual occurrences of the class in the specified dataset.

- **Inference**: Support values vary for each class based on the number of instances present in the test dataset for each class.

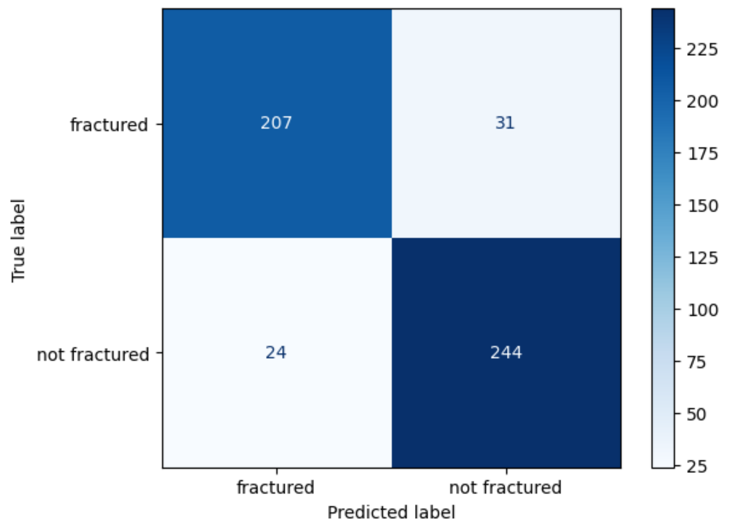
5. Accuracy:
Accuracy measures the overall correctness of the model's predictions. It is calculated as the ratio of correctly predicted instances to the total number of instances.

$$\text{Accuracy}=\frac{\text{True Positives+True Negatives}}{\text{Total Instances}}$$

Inference: The overall accuracy of the model is 98.67%, indicating that the model correctly predicts the class for nearly 99% of the instances in the test dataset.

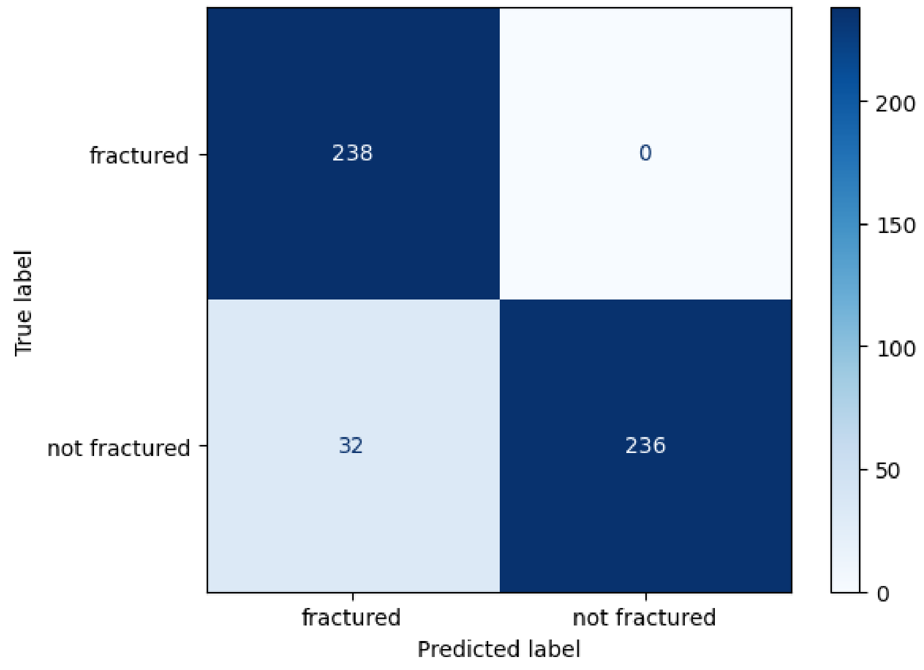
Here based on Training loss and Test loss also we can say that our model is over fitting or no:
Result: -
Training Loss: 0.0132
Training Accuracy: 0.9965
Test Loss: 0.0423
Test Accuracy: 0.9867

CNN Results:



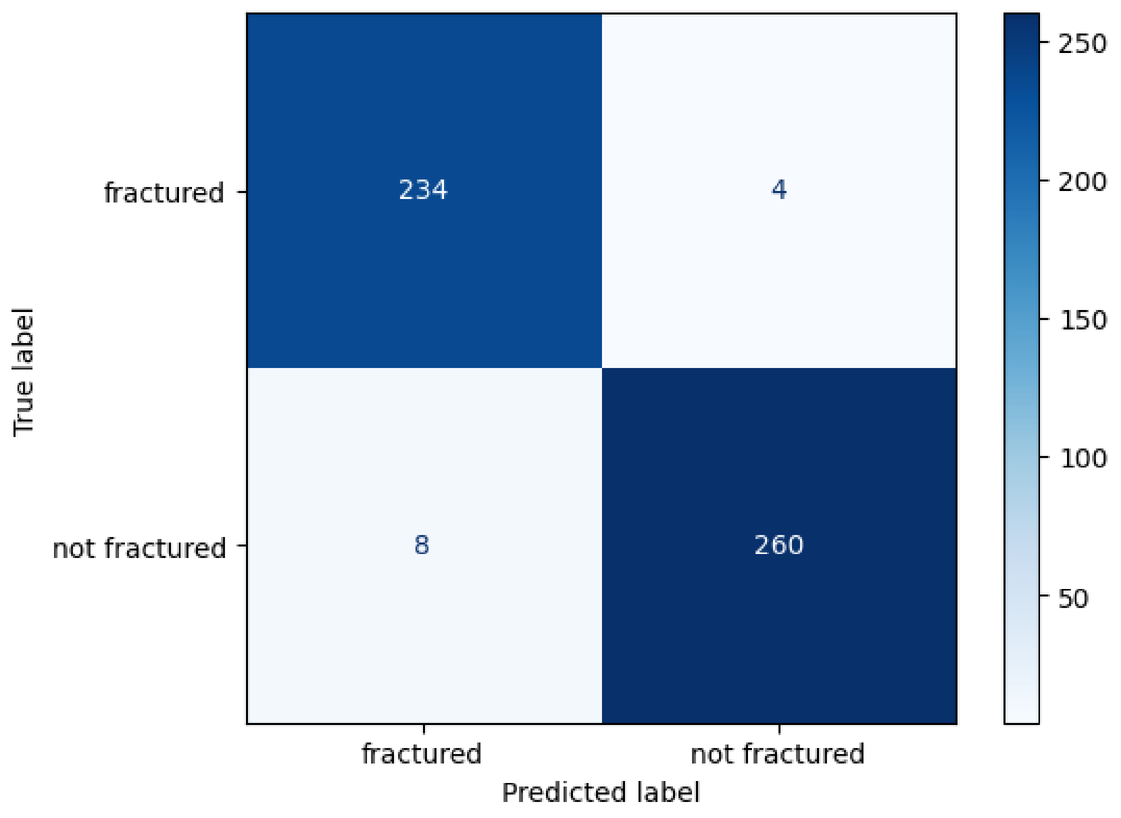
```
Loss: 0.2100 Acc: 0.8913
Precision: 0.8873 Recall: 0.9104 F1 Score: 0.8987
```

RESNET50:



```
Loss: 0.1556 Acc: 0.9368
Precision: 1.0000 Recall: 0.8806 F1 Score: 0.9365
```

VGG16 Results:



```
Loss: 0.0638 Acc: 0.9763
Precision: 0.9848 Recall: 0.9701 F1 Score: 0.9774
```

Based on the results obtained from the evaluation of the CNN, ResNet, and VGG16 models on the test dataset, we can draw the following inferences and conclusions:

Performance Comparison:

The VGG16 model achieved the highest accuracy (97.63%), followed by the ResNet model (93.68%), and then the CNN model (89.13%).
Both the VGG16 and ResNet models outperformed the CNN model in terms of accuracy, indicating that these deeper architectures were more effective in capturing complex features and patterns in the dataset.
Loss Analysis:

The VGG16 model had the lowest loss (0.0638), followed by the ResNet model (0.1556), and then the CNN model (0.2100).
Lower loss values indicate better convergence and model performance, suggesting that the VGG16 model achieved the best balance between minimizing prediction errors and generalizing well to unseen data.

Precision, Recall, and F1 Score:

The VGG16 model achieved high precision (98.48%), recall (97.01%), and F1 score (97.74%), indicating its ability to correctly classify positive cases (fractures) while minimizing false positives and false negatives.
The ResNet model achieved perfect precision (100%), but a lower recall (88.06%) compared to VGG16, resulting in a slightly lower F1 score (93.65%).
The CNN model had slightly lower precision (88.73%) and recall (91.04%), leading to a lower F1 score (89.87%) compared to both ResNet and VGG16.

Conclusion:

The VGG16 model demonstrated the best overall performance among the three architectures, achieving the highest accuracy, lowest loss, and highest precision, recall, and F1 score.
ResNet also performed well, with high accuracy and a perfect precision score, but slightly lower recall compared to VGG16.
The CNN model, while achieving decent accuracy, suffered from higher loss and lower precision, recall, and F1 score compared to ResNet and VGG16, indicating its limitations in capturing intricate features in the dataset.
Overall, the deeper architectures like VGG16 and ResNet showed superior performance in fracture detection compared to the simpler CNN model, suggesting their effectiveness in medical image classification tasks.

References:

https://mdpi-res.com/d_attachment/diagnostics/diagnostics-13-03245/article_deploy/diagnostics-13-03245.pdf?version=1697636570
https://www.mdpi.com/2076-3417/10/4/1507
https://www.mdpi.com/1424-8220/20/24/7237
https://link.springer.com/article/10.1007/s00542-020-05477-y
https://www.kaggle.com/datasets/bmadushanirodrigo/fracture-multi-region-x-ray-data
https://ieeexplore.ieee.org/document/7780459