Convolutional neural network for fracture detection in x-ray images

Juan José García Aguirre

Abstract - This study introduces a convolutional neural network (CNN) tailored for the detection of bone fractures based on X-ray images. Image classification is a crucial task in computer vision, finding applications across diverse fields. In the medical field, X-ray images serve as pivotal diagnostic tools, offering insights into a patient's health condition. This work focuses on automating fracture detection using X-ray images alone, leveraging CNNs for image classification. The proposed methodology integrates various tools and techniques within a CNN framework, implemented using PyTorch and executed on the Google Colab environment with a Tesla T4 GPU. The approach aims to enhance diagnostic accuracy and efficiency in detecting bone fractures.

Introduction

Image classification is a fundamental problem in computer vision within various application fields. One of these could be the processing of X-rays, which encompass different types of radiographies to obtain predictions about a person's health based only on the X-ray images. This serves as a diagnosis tool. In [1], this approach is presented for classifying X-ray images to detect pneumonia. It is important to note that the procedure is only used as a diagnostic support tool. Another example is presented in [2], where another specific type of X-ray is used to detect gastritis applying convolutional neural networks (CNNs), which are commonly employed in image processing.

The present work describes the design of a neural network for the classification of X-ray images, determining whether a fracture exists or not, using the labels "fractured" and "not fractured". For this purpose, we start from the CNN type of neural network, integrating it with various tools. The framework for the development of the network was conducted in PyTorch, and executed entirely in the Google Colab environment, leveraging the available GPU defined as Tesla T4.

Methodology

The methodology employed is illustrated in Fig. 1, delineating two main processes: training and testing. Initially, the training dataset was augmented to enhance its robustness. This augmentation strategy primarily involved image flipping. Subsequently, the training dataset was partitioned into randomly selected subsets for training and validation purposes. The validation subset played a crucial role in evaluating the model's performance without bias towards the training data. The obtained model was then deployed for the testing phase. Prior conducting the prediction process, the test data underwent was preprocessed by image flipping.

A. Dataset

The dataset used for the work was the "Fracture detection using x-ray images" dataset [3] obtained from the Kaggle website. All images are in jpg format and have dimensions of 224 x 224 pixels. These images are separated according to their label in different folders, the dataset itself has a folder called "val", this will be defined as the "test_loader", the other folder named "train" will be used for training and validation. Images are further classified into two labels "Fractured" or "Not fractured". Table 1 shows the composition of the dataset, However, following the augmentation process, the data distribution changes.

Table 1 Dataset composition

Dataset	Fractured	Not fractured	Total
Train	4480	4383	8863
Val	360	240	600
Total	4840	4623	9463

B. Pre-processing

The preprocessing process includes several initial transformations performed on the images. One such transformation is 'ToTensor', which converts the images into PyTorch tensors, making them suitable for input into the neural network. Another crucial step is normalization, which involves calculating the mean and standard deviation of the image groups. This normalization is then applied to the images to ensure that the pixel values are standardized. By normalizing the images, the preprocessing process helps in achieving faster convergence of the model during training.

C. Augmentation

To improve the model's performance, augmentation strategies were implemented, including simple transformations such as 'flip'. Additionally, before the training process begins, the data is normalized based on its initial mean and standard deviation to ensure consistency. After performing these transformations and splitting the data into different loaders for training, validation, and testing, the dataset's distribution is analyzed. The resulting distribution is presented in Table 2, providing a clear overview of how the data is allocated across these different phases.

Table 2 Loaders composition

Loader	Total images
Train	15954
Val	1772
Test	600
Total	18326

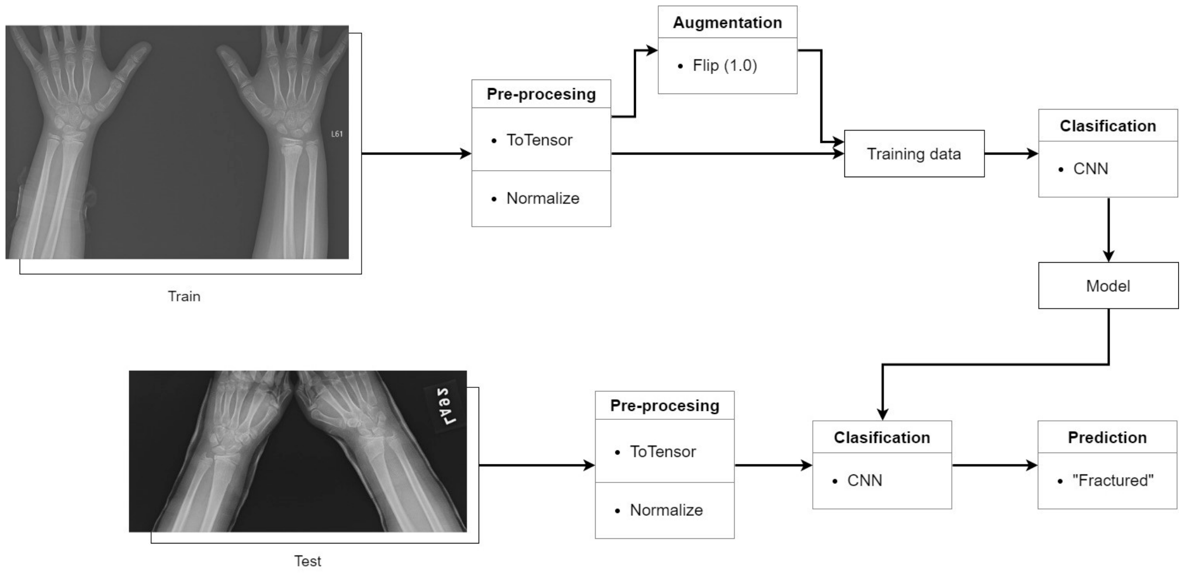


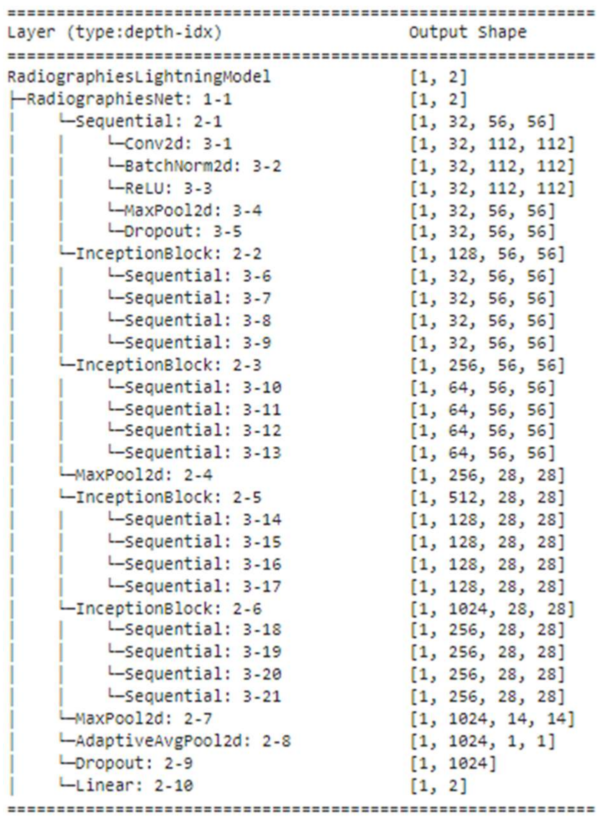
Fig. 1 Methodology of CNN

D. Model architecture and training

The architecture of the proposed model is based on a convolutional neural network with Inception blocks. These blocks are designed to extract features in parallel through different convolution and pooling operations. The Inception modules configuration with dimension reduction is used. After each Inception block, a pooling layer is added to reduce dimensions.

The model starts with an initial convolutional layer that processes input images of 3 channels (RGB) and produces 32 features using a 7x7 kernel. This is intended to provide a broad analysis of the image, aiming to find patterns that can be used for prediction. Additionally, a stride of 2 is used to counteract the high kernel, reducing the amount of data for memory saving. Then, a series of Inception blocks follow, each followed by a pooling layer to reduce dimensions. After the Inception blocks, the model has final layers, including fully connected linear layers and a *softmax* output layer to classify the images into the desired classes.

Regarding training, the cross-entropy loss function is used to calculate the error between the model's predictions and the actual labels. The Adam optimizer is employed to adjust the model's weights during training, the learning rate used was 0.001, as it is a common value for neural networks of this type. Regularization is incorporated through dropout after some



```
==========================================================
Layer (type:depth-idx)                   Output Shape
==========================================================
RadiographiesLightningModel              [1, 2]
├─RadiographiesNet: 1-1                  [1, 2]
│    └─Sequential: 2-1                   [1, 32, 56, 56]
│    │    └─Conv2d: 3-1                  [1, 32, 112, 112]
│    │    └─BatchNorm2d: 3-2             [1, 32, 112, 112]
│    │    └─ReLU: 3-3                    [1, 32, 112, 112]
│    │    └─MaxPool2d: 3-4               [1, 32, 56, 56]
│    │    └─Dropout: 3-5                 [1, 32, 56, 56]
│    └─InceptionBlock: 2-2               [1, 128, 56, 56]
│    │    └─Sequential: 3-6              [1, 32, 56, 56]
│    │    └─Sequential: 3-7              [1, 32, 56, 56]
│    │    └─Sequential: 3-8              [1, 32, 56, 56]
│    │    └─Sequential: 3-9              [1, 32, 56, 56]
│    └─InceptionBlock: 2-3               [1, 256, 56, 56]
│    │    └─Sequential: 3-10             [1, 64, 56, 56]
│    │    └─Sequential: 3-11             [1, 64, 56, 56]
│    │    └─Sequential: 3-12             [1, 64, 56, 56]
│    │    └─Sequential: 3-13             [1, 64, 56, 56]
│    └─MaxPool2d: 2-4                    [1, 256, 28, 28]
│    └─InceptionBlock: 2-5               [1, 512, 28, 28]
│    │    └─Sequential: 3-14             [1, 128, 28, 28]
│    │    └─Sequential: 3-15             [1, 128, 28, 28]
│    │    └─Sequential: 3-16             [1, 128, 28, 28]
│    │    └─Sequential: 3-17             [1, 128, 28, 28]
│    └─InceptionBlock: 2-6               [1, 1024, 28, 28]
│    │    └─Sequential: 3-18             [1, 256, 28, 28]
│    │    └─Sequential: 3-19             [1, 256, 28, 28]
│    │    └─Sequential: 3-20             [1, 256, 28, 28]
│    │    └─Sequential: 3-21             [1, 256, 28, 28]
│    └─MaxPool2d: 2-7                    [1, 1024, 14, 14]
│    └─AdaptiveAvgPool2d: 2-8            [1, 1024, 1, 1]
│    └─Dropout: 2-9                      [1, 1024]
│    └─Linear: 2-10                      [1, 2]
==========================================================
```

Fig. 2 Summary of proposed architecture

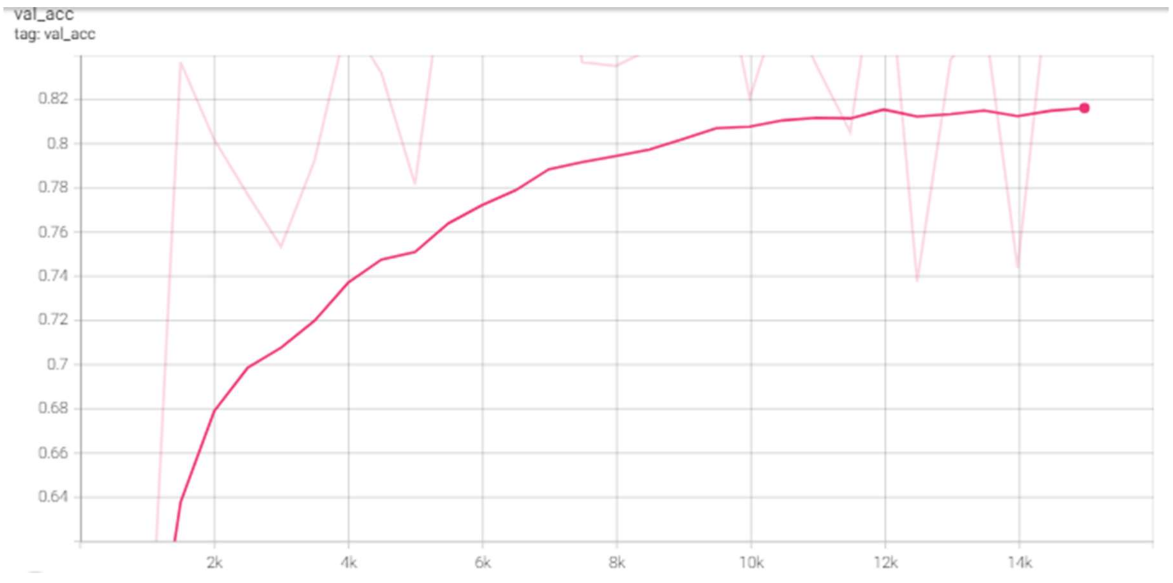


Fig. 3 Validation acuracy vs steps

convolutional and fully connected layers to prevent overfitting. The model is trained using training data, and its performance is validated using validation data.

Results

A. Training

The training process was carried out with a total of 15,954 images as indicated in Table 2, over a total of 30 epochs. As shown in the Fig. 3, a validation accuracy of 85% was obtained for the model, which is a good performance compared to the neural networks developed in [2]. It should be noted that they are different types of X-rays, However, they can serve as a reference point to gauge what constitutes adequate values. Furthermore, in the Fig. 4, it is possible to see how the loss value for training converges throughout the epochs or steps.

B. Testing results

Regarding the testing results, an accuracy of 81% was obtained with the data loader that was previously defined as "Test loader", indicating a reasonable predictive capability for detecting fractures in X-ray images. This result demonstrates a good performance of the trained model and that can be used as a support tool for fracture detection.

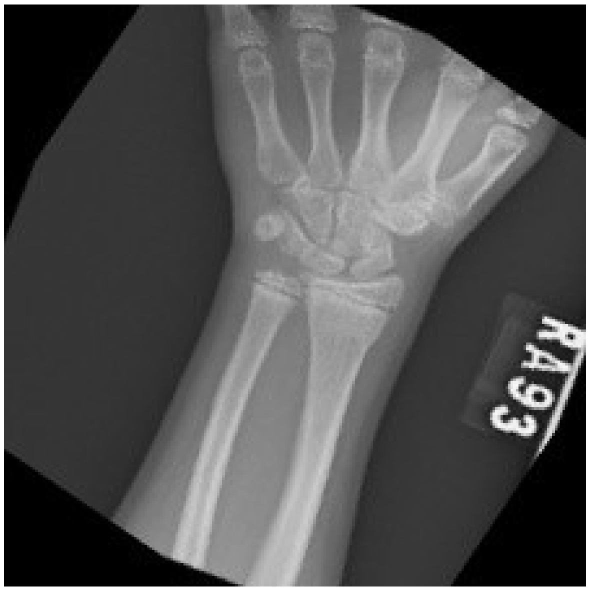


Fig. 5 Failed prediction example

C. Failure pattern

After the evaluation, it is necessary to know or understand the reasons why the model fails to predict, for this reason, it is decided to observe the type of images which the prediction fails, for this, when evaluating the model, it is possible to see that it exists a tendency to failure for images that are from

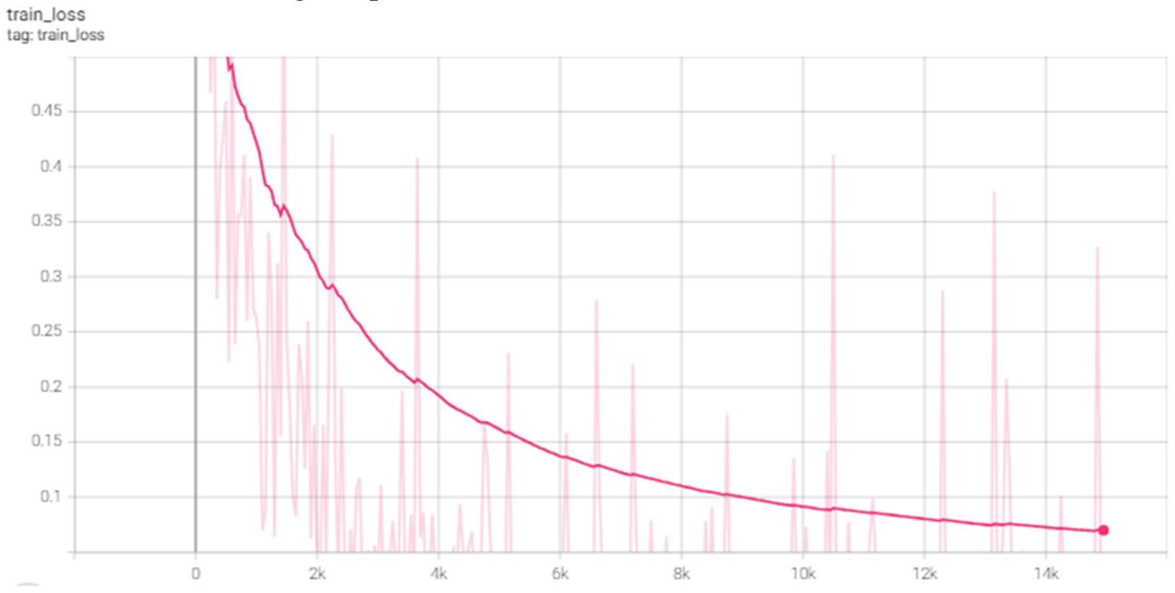


Fig. 4 Training loss vs steps

hands, an example is shown in the Fig. 5 where an images is obtained that the model predicts as “not fractured”, but according to the initial label it should be predicted as “fractured”, this shown a starting point in which is the type of x-ray images where the model does not find a way to differentiate, perhaps parts where the bones are very small, so an alternative could be the application of different augmentation techniques such as cropping or scaling.

Conclusion

This project proposes a CNN architecture with an augmentation strategy to detect fractures in X-ray images. The experiment results show that the architecture used produces optimal results. It seems that the model may be undergoing overfitting, but it could be improved by increasing the size of the data used. Compared to other models, this dataset can be considered small, so the performance achieved (81% test accuracy) demonstrates that the model can be used with small datasets or much larger ones. All relevant project information is located in a GitHub repository [4].

References

[1] S. A. Khoiriyah, A. Basofi, and A. Fariza, “Convolutional Neural Network for Automatic Pneumonia Detection in Chest Radiography,” *IES 2020 - International Electronics Symposium: The Role of Autonomous and Intelligent Systems for Human Life and Comfort*, pp. 476–480, Sep. 2020, doi: 10.1109/IES50839.2020.9231540.

[2] R. Togo *et al.*, “Detection of gastritis by a deep convolutional neural network from double-contrast upper gastrointestinal barium X-ray radiography,” *J Gastroenterol*, vol. 54, doi: 10.1007/s00535-018-1514-7.

[3] “Fracture detection using x-ray images.” Accessed: May 24, 2024. [Online]. Available: https://www.kaggle.com/datasets/devbatrax/fracture-detection-using-x-ray-images/data

[4] “juang1910/Radiographies.” Accessed: Jun. 04, 2024. [Online]. Available: https://github.com/juang1910/Radiographies