Bone Fracture Classification

A Cloud Pipeline

Alex Slep, Kevin Shi, Ning Tang

Abstract
In this report, we investigate two datasets for bone fracture classification, we create three data preprocessing pipelines to mitigate the weaknesses of the datasets, and we create a classification model to be trained on these six separate datasets. Finally, we train our model on the cloud using NYU HPC and analyze our results.

Introduction

The medical field is perhaps one the fields most ripe for exploration with machine learning and computer vision techniques. Diagnostic medicine in particular is full of imagery and scans. X-rays, MRIs, and CT scans are all incredibly common imaging tools that allow doctors to peer into the body of a patient. The presented problems are simple computer vision tasks that have dozens, if not hundreds of developed approaches. In fact, a significant amount of diagnostic medicine boils down to two things: classification and detection on images. Is there a tumor? If so, what variety? Does a patient suffer from a degenerative disease? Or, most importantly for our project: is this bone broken?

Computer vision models trained to classify images on tasks like these should theoretically be simple. Beyond even that, they should be invaluable. There is perhaps no other field where human judgement on an image classification task is so expensive and requires so much training. It then may come as a surprise that the field is not full of computer vision.

On a deeper dive, the reason behind this becomes clear: HIPAA and other privacy laws make the collection and storage of any medical datasets extremely difficult [1]. This is a good thing; we want good patient privacy protections. However, it does make collecting quality data for machine learning training very difficult.

In this project, we started with a simple computer vision classification problem: classification of x-ray images based on the presence of a bone fracture. We had a dataset, which we knew to be extremely flawed. Our project's original premise was to mitigate the quality of the dataset as much as we could, with the end goal of training a classification model on the dataset using the cloud. However, upon exploring our initial results, we discovered that our dataset was even more flawed than we thought. This led us to expanding the scope of the project to other publicly available datasets.

In this paper, we will first introduce two datasets, our initial dataset and a second smaller but more robust dataset we found. Then we will discuss three separate data preprocessing pipelines we developed to mitigate the limitations of both datasets. Next we will discuss our baselines and the model we developed. Finally, we will discuss and analyze our results.

Datasets

Dataset 1: Bone Fractures: Tibia and Fibula [2]

This dataset is a simple classification dataset containing x-rays of the tibia and fibula of patients sorted into "normal" and "fracture" sets. The dataset itself is small, with only 2,125 samples. However, this is not the largest issue with the dataset. It contains a massive class imbalance, with only 126 samples in the "normal" category. Additionally, many groups of samples were from the same patients, further limiting the amount of useful information carried in the dataset.

Dataset 2: FracAtlas [3]

This second dataset was much more robust. Compiled by Abedeen et al. in their paper of the same name, this dataset contains a similar separation into "normal" and "fracture" except with 4024 samples. The class imbalance in this dataset was in the opposite direction, with 717 broken samples and 3307 normal samples. More importantly than that, however, is that the quality of the dataset was massively better. The study by Abedeen et al. is a testament to the care put into compiling this dataset for use.

Data Preprocessing

Given the massive class imbalance in the first dataset and the small size of both datasets, we needed a preprocessing pipeline built to mitigate these downsides. First and foremost, we split our data into 70/15/15 training/eval/test sets. Then we upsampled the underrepresented class in the training sets to avoid our model's performance degrading due to the class imbalance. Finally we began the process of data augmentation.

Data augmentation, or in this case image augmentation, is the process of applying a series of transformations to the images with the end goal of enhancing generalization and the amount of data [4]. These transformations come in many forms, from random cropping to reflections to adding noise. As this inherently adds samples to the dataset, the model gains more data to train on. In terms of generalization, the similar but distinct data created in these processes encourages the model to learn the features that represent each class as opposed to ancillary data like orientation or location in the frame. Using these tools we created three preprocessing pipelines to create three versions of each dataset.

Pipeline 1: Original

In this pipeline, upsampling was first performed. After this, all images went through a transformation to resize them to 512x512 pixels and converted them to single channel grayscale to limit the model size blowing up. The result of this pipeline was most similar to the original input dataset.

Pipeline 2: Augmented

This pipeline began with the same steps that are contained in the first pipeline. It then however performed horizontal and vertical reflection transformations, tripling the size of the training set.

Pipeline 3: Downsized

This pipeline was similar to the original pipeline. The only difference was the resize was to 224x224 pixels instead. The goal of this pipeline was to discover if a significantly smaller model in terms of parameter count could compete with the original model.

Model Architecture

The model can be organized into 4 main blocks. The first 3 blocks are convolutional blocks, consisting of a convolutional layer, a ReLU activation, and a batchnorm layer. The first 2 blocks also have a pooling layer to extract more relevant features from the outputs. The hyperparameters for the convolutional layers are: kernel size 3, stride 1, and padding 1. For the pooling layers, they are: kernel size 2, stride 2, and padding 0. This selection ensures that the output dimension matches the input dimension, allowing for visualization of model outputs. Furthermore, the channel size output (number of filters) goes as 32, 64, 128. Finally, the fourth and final block is a simple linear layer followed by a sigmoid activation to perform binary classification.

Model Training

With our datasets preprocessed and split into six variations—three pipelines for each of the two datasets—we then moved to model training. All models were trained for 5 epochs using the same training parameters. We used Stochastic Gradient Descent (SGD) as our optimizer with a learning rate of 0.0001 and a weight decay of 0.2. The loss function used was Binary Cross Entropy, which is a natural fit for binary classification tasks. To keep GPU memory usage manageable, especially with the larger image sizes in the original and augmented pipelines, we set the batch size to 16. Each model was trained and evaluated using the NYU High Performance Computing (HPC) Greene cluster, which allowed for consistent and efficient execution across datasets.

To ensure fair comparisons across pipelines, we maintained the same training, validation, and test splits within each dataset and followed the same data loading and training loop structure. Performance was tracked at the end of each epoch on the validation set, and final test metrics were reported only after training completion. Finally, for each pipeline and dataset combination, we saved the trained model weights for potential downstream tasks or further experimentation.

Results

The results of our training experiments reflect both the strengths and weaknesses of the datasets and the preprocessing pipelines applied. In total, we trained six models: three (original, augmented, and downsized) on each of the two datasets. Our evaluation focused on two key metrics: classification accuracy and average binary cross-entropy loss, both measured on the test sets.

Starting with Dataset 1 (the tibia and fibula x-rays) [2], the standard model achieved high performance but presented with a flawed class imbalance (many more fractured images than normal ones), reaching 99% test accuracy with a low test loss of 0.022. The augmented model, which had three times the number of training samples due to image flips, also performed very well, achieving 98% accuracy with slightly higher test loss at 0.0377. The downsized model, which used only 224x224 images, was able to match the augmented model's accuracy at 98%, though with the highest loss among the three at 0.0743. These results suggest that even with a flawed and small dataset, basic preprocessing steps—combined with a simple CNN—can produce high-performing classifiers, particularly when working with higher-resolution inputs.

Dataset 2, sourced from the FracAtlas collection [3], was significantly more balanced for the fracture and normal class. By applying the same three preprocessing pipelines, we observed a more modest but consistent pattern. The standard model achieved 78% accuracy on the test set with a loss of 0.5521. The augmented model followed closely with 77% accuracy and a slightly better loss of 0.5482. The downsized model, though lower in accuracy at 74%, remained competitive with a loss of 0.5554. These results indicate that the improvements from augmentation were more limited in this case, suggesting that model confidence was lower overall due to the dataset's inherent complexity.

Model Visualization

In addition to the standard accuracy and loss metrics, we thought it would be instructive to also look at the model outputs. In our case, we examined the outputs after training our model on Datasets 1 and 2. For this, we just used the model trained on both standard datasets.

We took two approaches to do this: The first is a less standard but previously explored method of doing so [5], which is to fix the trained model weights, generate an input image with random noise, and perform gradient descent not over the model parameters (the weights) but *over the image pixels* themselves. The objective in the binary classification case is to maximize the probability of a prediction given a particular label. For example, we want the model to produce an image that it 'thinks' best represents a fractured bone, so we fix the label to '1.' We then want the output *before the sigmoid activation* (or the logits) to go to infinity, which maximizes the probability of a broken bone after being converted to a probability distribution via the sigmoid. In such a way we want to maximize this value such that the loss can be considered the negative logit, allowing for gradient descent to be performed over the pixels. Ideally, this would produce a representative image for the desired class label.

The second is much more conventional. We basically just clip the model to exclude the classification layer, leaving only the convolutional layer outputs (which we ensured had the same output dimension as the input by setting the hyperparameters). These outputs can then be visualized by plotting them using matplotlib, which will light up pixels with higher numerical values in the output tensor. In such a way, we can see what the model is really paying attention to by seeing which pixels light up.

Discussion

For both tests, I used the following two images (one normal, one fractured):

Image of a Normal Bone.

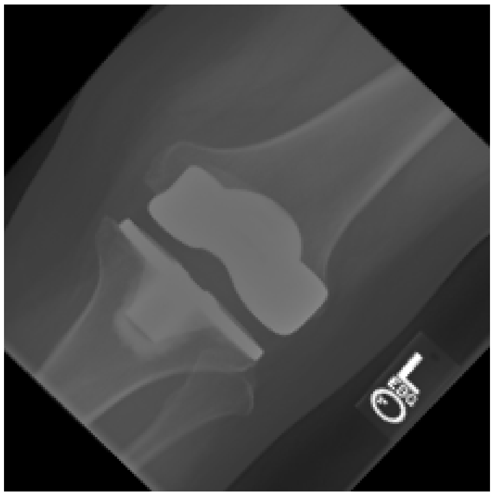
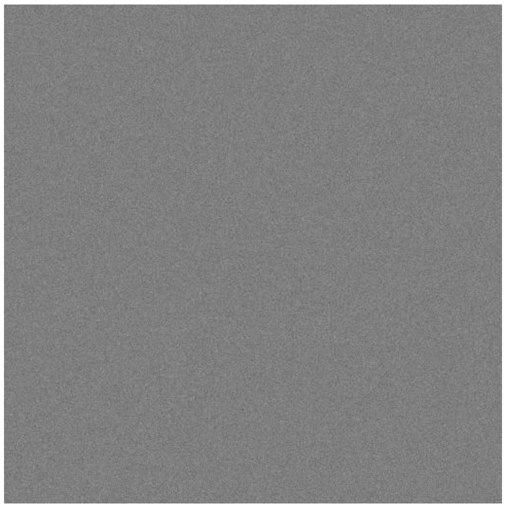
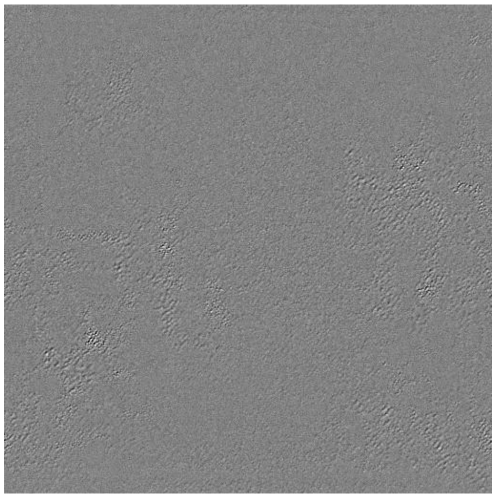


Image of a Fractured Bone.

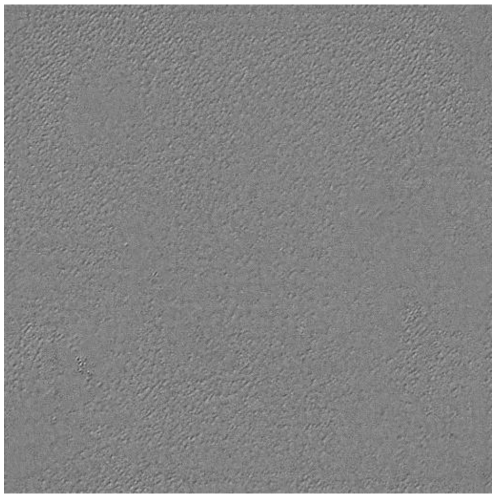
Then, I generated a random noisy image:



After performing gradient descent, the following output is produced:

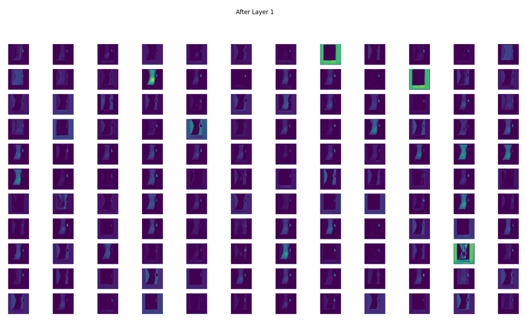


Normal Bone.



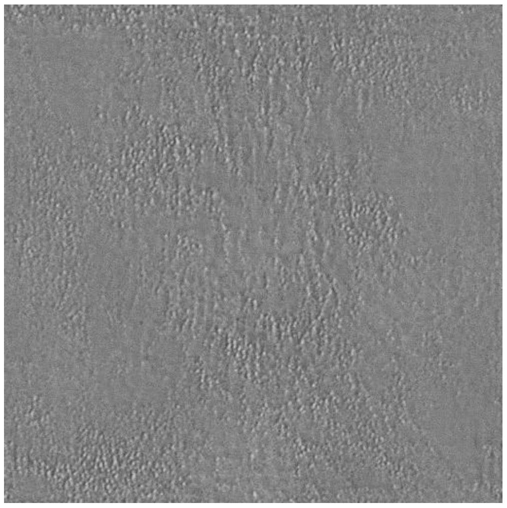
Fractured Bone.

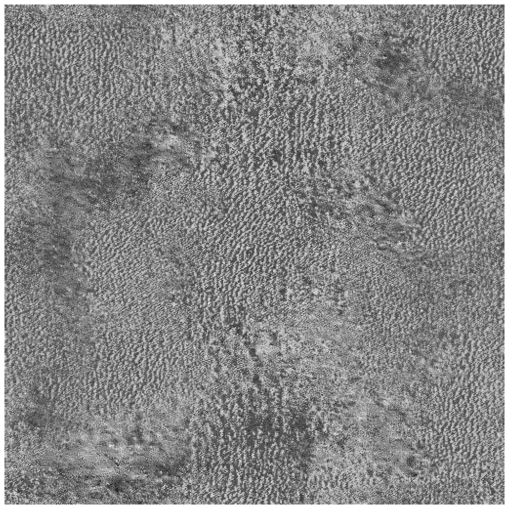
As we can see, the image is 'more' structured, but we still can't see what it is learning, nor is it what we would want or expect the model to learn. More insight into why can be gleaned from visualizing the model outputs directly:



It might be too small to see, but if you zoom in you can see that not many of the filters seem to be activating on the bones. Instead, almost ubiquitously, small artifacts to the sides of each image are lighting up, which actually turn out to be the letters to the sides of the medical images in Dataset 1. This partially explains why our first method did not produce images with any bonelike features, and also perhaps why for Dataset 1 our model performed unrealistically well. This is also corroborated by the less optimistic results of our model trained on Dataset 2, which seems to confirm this understanding of the data.

When running the same "denoising" gradient descent method on the model trained on the standard Dataset 2, we can see that much more structure appears to be learned:





The first image is of a normal bone, and the second is of a fractured bone. For the second image, darker lines and holes/pores appear to be a common theme, indicating that this is what a model 'believes' differentiates broken and normal bones (when trained on a better bone dataset). Although there don't appear to be bones appearing, (which is expected. Since the dataset only consists of bones, what differentiates broken and normal bones is more likely to be break lines and other signs of fracture), there is more to interpret here, and it appears the model is actually learning some important features.

Conclusion

There is still room for improvement, especially in regards to model architecture. The model designed by our team is extremely lightweight, and is trained on a very small amount of image data. It would be instructive to perform similar analysis and visualization on a pretrained model (like a VGG model with pretrained weights) after fine tuning it on both of the fractured bone datasets.

References

[1] B. Whittaker, “Healthcare AI and HIPAA privacy concerns: Everything you need to know,” www.tebra.com, Dec. 15, 2023. https://www.tebra.com/theintake/practice-operations/legal-and-compliance/privacy-concerns-with-ai-in-healthcare

[2] Bone Fracture Dataset – Tibia and Fibula. Mendeley Data. [Online]. Available: https://data.mendeley.com/datasets/2h62x9xzyd/1

[3] B. Whittaker, “Healthcare AI and HIPAA privacy concerns: Everything you need to know,” www.tebra.com, Dec. 15, 2023. https://www.tebra.com/theintake/practice-operations/legal-and-compliance/privacy-concerns-with-ai-in-healthcare

[4] “14.1. Image Augmentation — Dive into Deep Learning 1.0.0-beta0 documentation,” d2l.ai. https://d2l.ai/chapter_computer-vision/image-augmentation.html

[5] “Inceptionism: Going Deeper into Neural Networks.” https://research.google/blog/inceptionism-going-deeper-into-neural-networks/