# CS682 Final Project Report
# Detecting Abnormalities in Musculoskeletal X-rays

**Group Members:**

| | |
|---|---|
| Tianxiao Ren | G00929471 |
| Hao Yan | G01171541 |
| Ryan Moffatt | G00585488 |
| Alec Webb | G00433049 |

## 1 Abstract

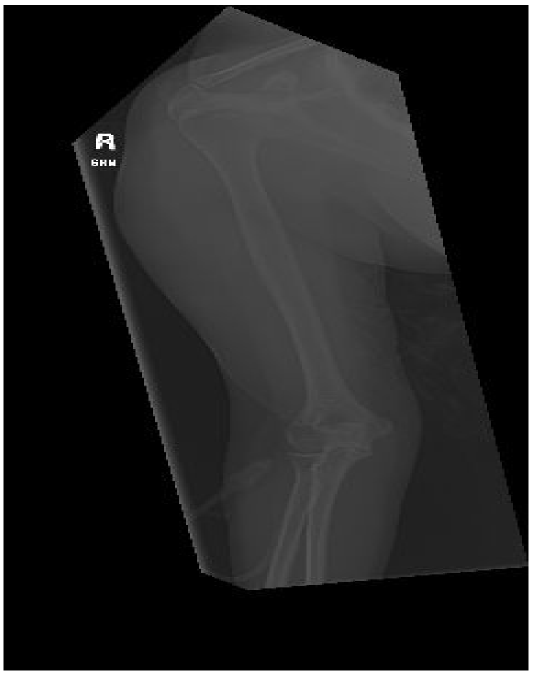
The radiograph generated by X-rays is a better and widely used way to see the anomaly of human bones by doctors. However, the fractures only can be viewed by eyes from radiographs. If a robust and reliable algorithm could be developed for automatically detecting the fracture, it will save a lot of time for both doctors and patients and save a lot of hospital resources. Moreover, it could be further developed to provide the best treatment suggestions to doctors from the analysis of past cases. MURA (musculoskeletal radiographs) is a large dataset of bone X-rays radiography. Each image has been labeled as either fracture 1 or non-fracture 0 by hand. In this project, the proposed algorithm is to identify if a bone is broken from X-rays radiographs from the MURA-HUMERUS dataset. MURA-HUMERUS is the sub-dataset of MURA which contains 1,275 training radiographs and 288 validating radiographs. The original radiographs are processed by the noise removal and edge detection methods studied in class. The classification model is built by a 512 fully connected Convolutional Neural Network after applying image preprocessing on all training radiographs. The performance of the model is evaluated on the valid set. The proposed algorithm achieves the highest probability 0.62 using the Difference of Gaussian (DoG) and the Canny operator in conjunction with contours.

## 2 Introduction and Related Work

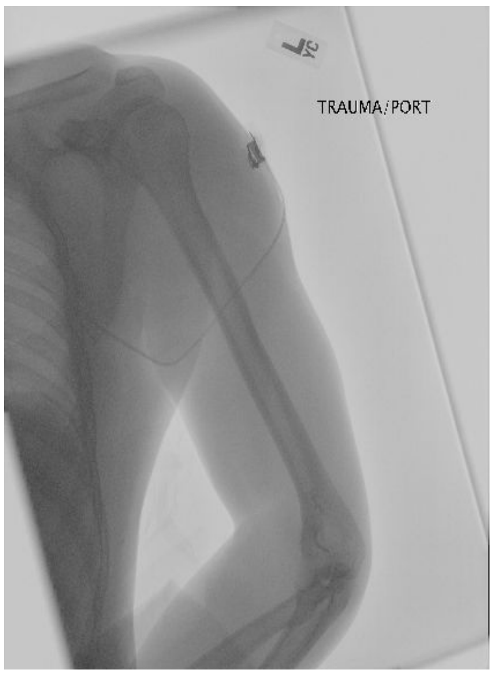
Billions of people are affected by musculoskeletal conditions worldwide. Bone fractures are one of the most common injuries, many of which are rushed to emergency rooms. The rapid diagnosis of bone abnormalities would aid patients and hospitals, such as lack of radiologists in a sudden influx of injured patients, regions that lack an abundance of radiologists, patients avoiding unneeded procedures, prioritize patients, or in general, as a computer-aided diagnosis system to prevent false positives from fatigued doctors. Abnormalities come in many ways: fractures, internal fixations, degenerative joint diseases, lesions, subluxations, etc. By using a machine learning approach to detect an abnormality, bulk diagnoses can be computed quickly to a degree of confidence, sometimes better than the average professional radiologist. Furthermore, this idea is more and more hopefully realized with the rapid development of machine learning and deep learning related techniques. In short, the machine learning or deep learning model is a supervised classification algorithm. It can extract the significant eigenvalues from the feature space of the training set then apply the model on the future studies for classification. The self-study ability makes the model powerful and sustainable.

For building the classification model, a large and high-quality dataset is required. In late 2017, Stanford released the “MURA: a large dataset for abnormality detection in [upper body] musculoskeletal radiographs[1].” A subset of this dataset was employed in our system consisting of the humerus bone alone, which contains 1,275 radiographs in the training set and 288 radiographs in the valid set. Each study contains one or more images and is manually labeled by radiologists as either normal 0 or abnormal 1 [2].

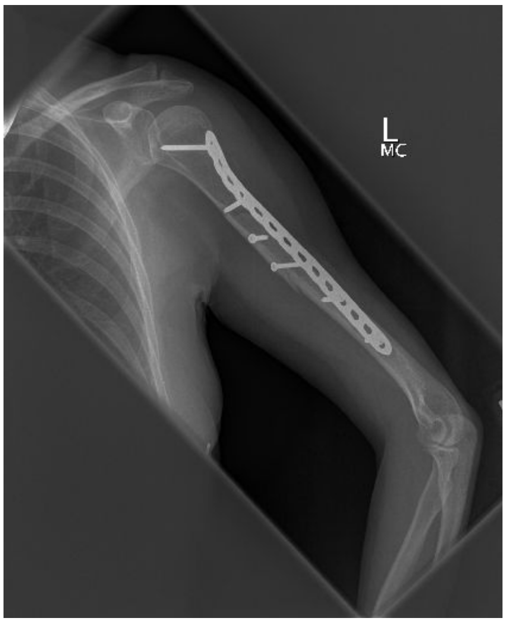
In this project, the goal is to develop a classification model that detects the anomaly of human bones. From the radiographs, the edge contours of bones are fed into a classifier to determine the probability of abnormalities. But first, X-ray images capture more than just bones from the body. Bones are dense, so they absorb much of the radiation, but the muscles and fat don’t allow for all the radiation to pass through them, so we need a technique to differentiate the bones from the soft tissue. The raw x-ray images are filtered using different noise reduction techniques, like smoothing and thresholding. Furthermore, all dimensionality reduction operations, such as resizing and flattening, were applied using non-lossy format (png). Then, the model will be trained by a 512 fully connected CNN.



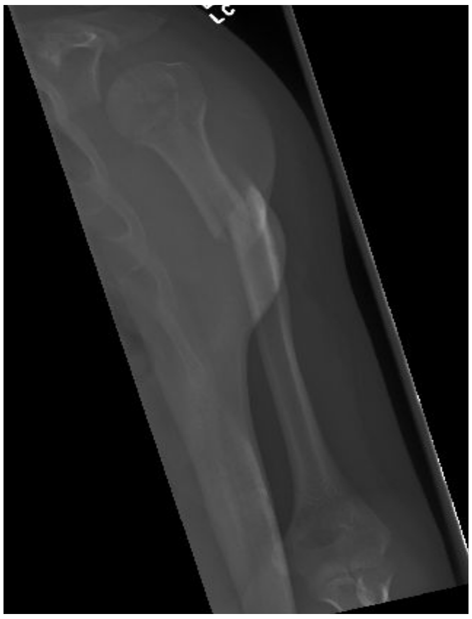
(a)



(b)



(c)



(d)

Figure 1: (a) and (b) show the normal (non-fracture) radiographs, (c) and (d) show the abnormal (fracture) radiographs from the HUMERUS subset.

There are some related works from other researchers: on the main website, [2] trained a 169-layer DenseNet to detect the abnormalities without any image preprocessing, they got 0.778 for the entire MURA dataset and 0.60 for the HUMERUS subset. [3] and [4] are from the same researchers. The model is trained by ANN in the same way but different from the image preprocessing methods. [3] proposed Contour Histogram Feature-Based (CHFB) and improved CHFB preprocessing on original radiographs. [4] introduced a Line-based Feature Extraction. [5] and [6] used the Canny Edge Detection for image preprocessing.

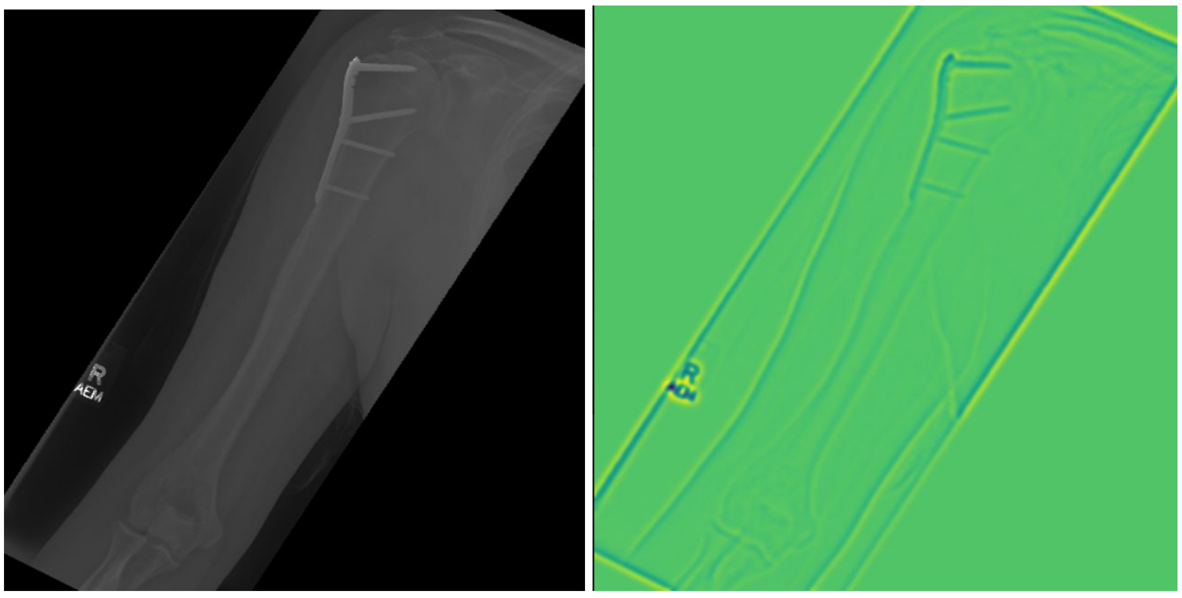
## 3 Technical Approach

### Preprocessing

In order to improve prediction accuracy, we applied two different pre-processing methods to the X-ray dataset.

Difference of Gaussians (DoG)

The first method we implemented is Difference of Gaussians(DoG). DoG is a feature enhancement algorithm which will blur the original image and enhance the edges on the original image. In our project, bones are usually wrapped with some other edges like muscles or skins. DoG helps us to enhance the edge of bones.



(a) (b)

Figure 2: (a) is the original radiograph, (b) is the radiograph after applying DoG.

Figure 2 clearly shows that after applying DoG, the texture of muscles are mostly blurred and the edges of bones get more contrast than the original image.
But there are also cons this method because the DoG also blurs some parts of the bones out.

**Canny/Contour**

There are two subroutines in this method. First we apply canny on the original image to remove noise and find the edges. But since the threshold of canny is fixed, the canny edge detector will not always give us the edge of bone. So then we find the contours of the edges we found, and the contours of bones are more significant than other small contours.

Figure 3 shows a good example of how canny and contours enhances the edges of bones. But still, if the threshold is not accurate enough to detect edges of bones in the first step, wrong contours will be found too.

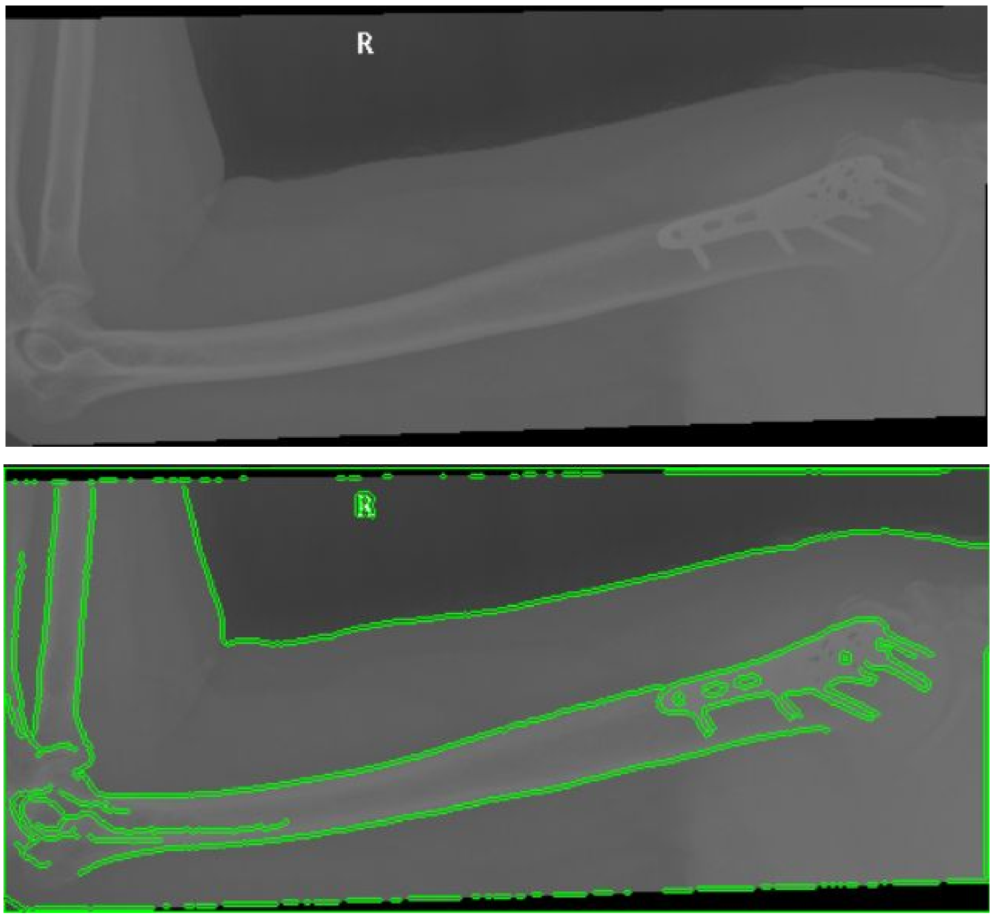


Figure 3

**Model Training**

CNN

To classify if the bone is broken or not, we apply CNN to extract features from the preprocessed images and feed the feature to a 512 fully connected neural network to train the model.

The construction of the CNN was introduced on keras official web page[7]. It works more than 70% accurate on the CIFAR10 dataset. CIFAR10 dataset consist of small images(32x32) with 10 classes but our dataset consists with large images(512*512) with 2 classes, we did some changes on the original construction. Since the bone broken regions are larger than CIFAR10 features,  to improve the performance, we replace the 3*3 convolutional kernel with 6*6 to save the convolution time.

In order to keep similarity between adjacent mini-batches and to speed up learning, RMSprop optimizer is used, which (1) has a slowly decaying learning rate (so the network slowly moves from exploration of edges to exploitation, to say, particular bone fragments) and (2) updates the neural network's weights based on the gradient of the cost divided by the moving average of recent gradients[8].

$$\Delta w_{ij} /= \alpha \sqrt{MeanSquare(w,t) = 0.9 * MeanSquare(w,t-1) + 0.1 * (\delta E / \delta w(t))^2}$$

For the deep neural networks, a good activation function is ReLU:

$$f(x) = max(0,x),$$

to overcome the vanishing gradient problem and for the last layer, softmax,

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} for i = 1,...,K and z = (z_1, \ldots, z_K),$$

cross-entropy loss to ultimately create a probability distribution (well, only 2 probabilities since it's a binary classification, so K = 2) of abnormal likelihood.

## 4 Results and Discussion

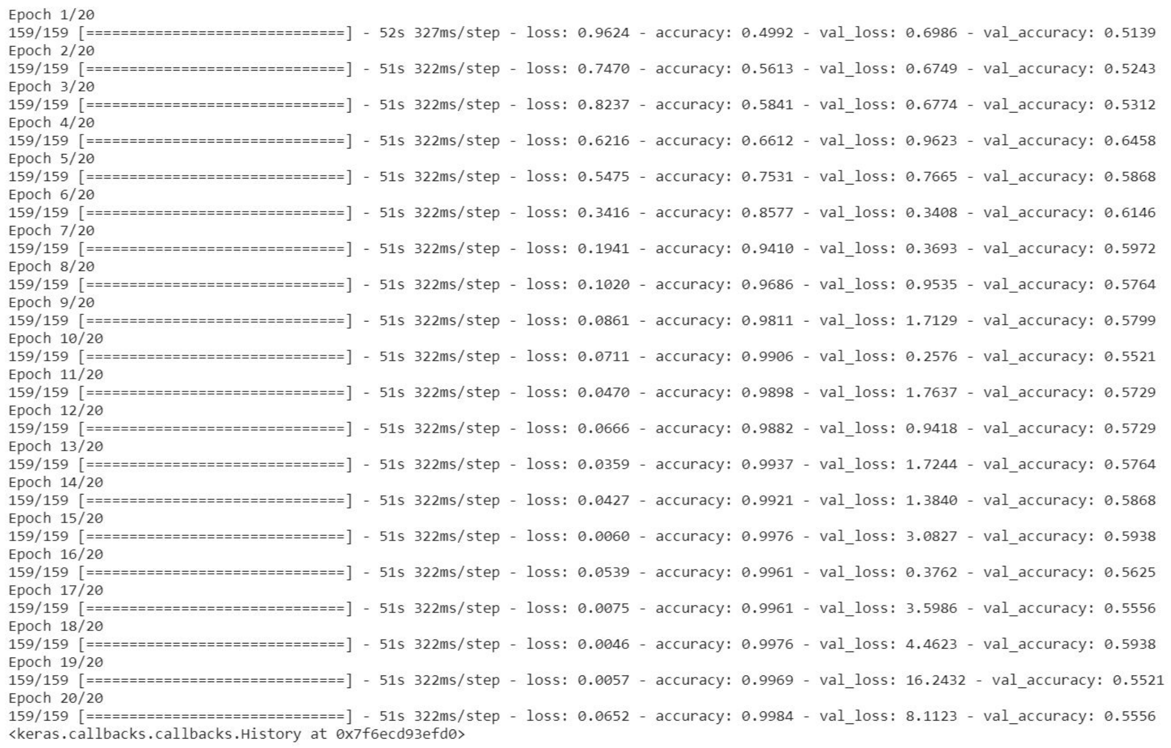
Six board-certified radiologists from Stanford hand-labeled their opinions of thousands of radiographs as normal or abnormal bones. The majority of their votes for each image is taken as the official *truth*. In order to compare human labels with the CNN model, each radiologist's opinion is individually compared to the truth with *Cohen's kappa statistic* ($k$), to account for the possibility that radiologists make guesses, sometimes, due to uncertainty (see Figure 4 below).

$$k = \frac{p_0 - p_e}{1 - p_e}$$

$k$ measures the reliability based on each of their ratings (i.e. no agreement, $k = 0$; everyone agrees, $k = 1$). For the humerus bone, below we will show that our classifier is comparable to the 169-layer neural net results of the Stanford ML group.

### Result of Original Images

First we feed images without preprocessing to our model. Here are the results after 20 epochs:

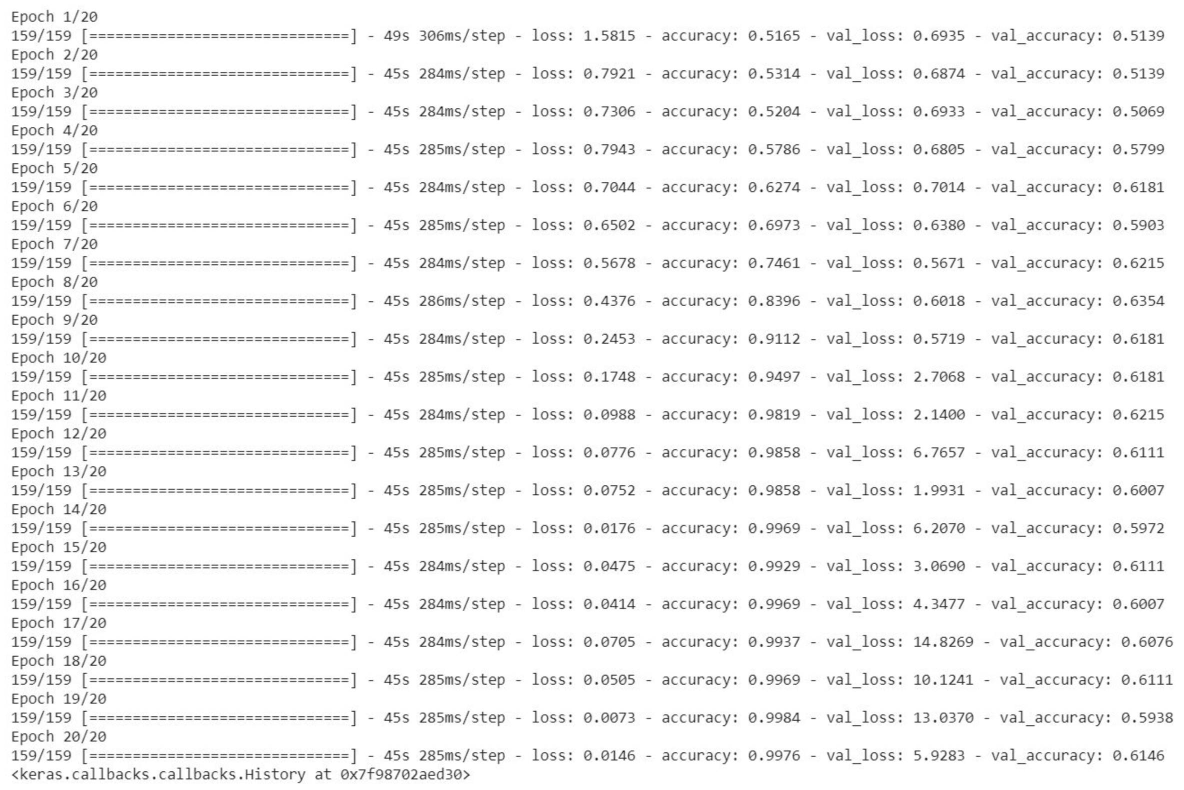


```
Epoch 1/20
159/159 [==============================] - 52s 327ms/step - loss: 0.9624 - accuracy: 0.4992 - val_loss: 0.6986 - val_accuracy: 0.5139
Epoch 2/20
159/159 [==============================] - 51s 322ms/step - loss: 0.7470 - accuracy: 0.5613 - val_loss: 0.6749 - val_accuracy: 0.5243
Epoch 3/20
159/159 [==============================] - 51s 322ms/step - loss: 0.8237 - accuracy: 0.5841 - val_loss: 0.6774 - val_accuracy: 0.5312
Epoch 4/20
159/159 [==============================] - 51s 322ms/step - loss: 0.6216 - accuracy: 0.6612 - val_loss: 0.9623 - val_accuracy: 0.6458
Epoch 5/20
159/159 [==============================] - 51s 322ms/step - loss: 0.5475 - accuracy: 0.7531 - val_loss: 0.7665 - val_accuracy: 0.5868
Epoch 6/20
159/159 [==============================] - 51s 322ms/step - loss: 0.3416 - accuracy: 0.8577 - val_loss: 0.3408 - val_accuracy: 0.6146
Epoch 7/20
159/159 [==============================] - 51s 322ms/step - loss: 0.1941 - accuracy: 0.9410 - val_loss: 0.3693 - val_accuracy: 0.5972
Epoch 8/20
159/159 [==============================] - 51s 322ms/step - loss: 0.1020 - accuracy: 0.9686 - val_loss: 0.9535 - val_accuracy: 0.5764
Epoch 9/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0861 - accuracy: 0.9811 - val_loss: 1.7129 - val_accuracy: 0.5799
Epoch 10/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0711 - accuracy: 0.9906 - val_loss: 0.2576 - val_accuracy: 0.5521
Epoch 11/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0470 - accuracy: 0.9898 - val_loss: 1.7637 - val_accuracy: 0.5729
Epoch 12/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0666 - accuracy: 0.9882 - val_loss: 0.9418 - val_accuracy: 0.5729
Epoch 13/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0359 - accuracy: 0.9937 - val_loss: 1.7244 - val_accuracy: 0.5764
Epoch 14/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0427 - accuracy: 0.9921 - val_loss: 1.3840 - val_accuracy: 0.5868
Epoch 15/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0060 - accuracy: 0.9976 - val_loss: 3.0827 - val_accuracy: 0.5938
Epoch 16/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0539 - accuracy: 0.9961 - val_loss: 0.3762 - val_accuracy: 0.5625
Epoch 17/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0075 - accuracy: 0.9961 - val_loss: 3.5986 - val_accuracy: 0.5556
Epoch 18/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0046 - accuracy: 0.9976 - val_loss: 4.4623 - val_accuracy: 0.5938
Epoch 19/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0057 - accuracy: 0.9969 - val_loss: 16.2432 - val_accuracy: 0.5521
Epoch 20/20
159/159 [==============================] - 51s 322ms/step - loss: 0.0652 - accuracy: 0.9984 - val_loss: 8.1123 - val_accuracy: 0.5556
<keras.callbacks.callbacks.History at 0x7f6ecd93efd0>
```

The accuracy converges to about 55% on the testing set. This accuracy is close to a random classifier which means CNN doesn’t work that well on original X-ray images. We still need some preprocessing to make the features more significant for our model.

**Result of Canny&Contour Preprocessed Images**

Then we feed the model with canny and contours processed images and here are the results after 20 epoches:

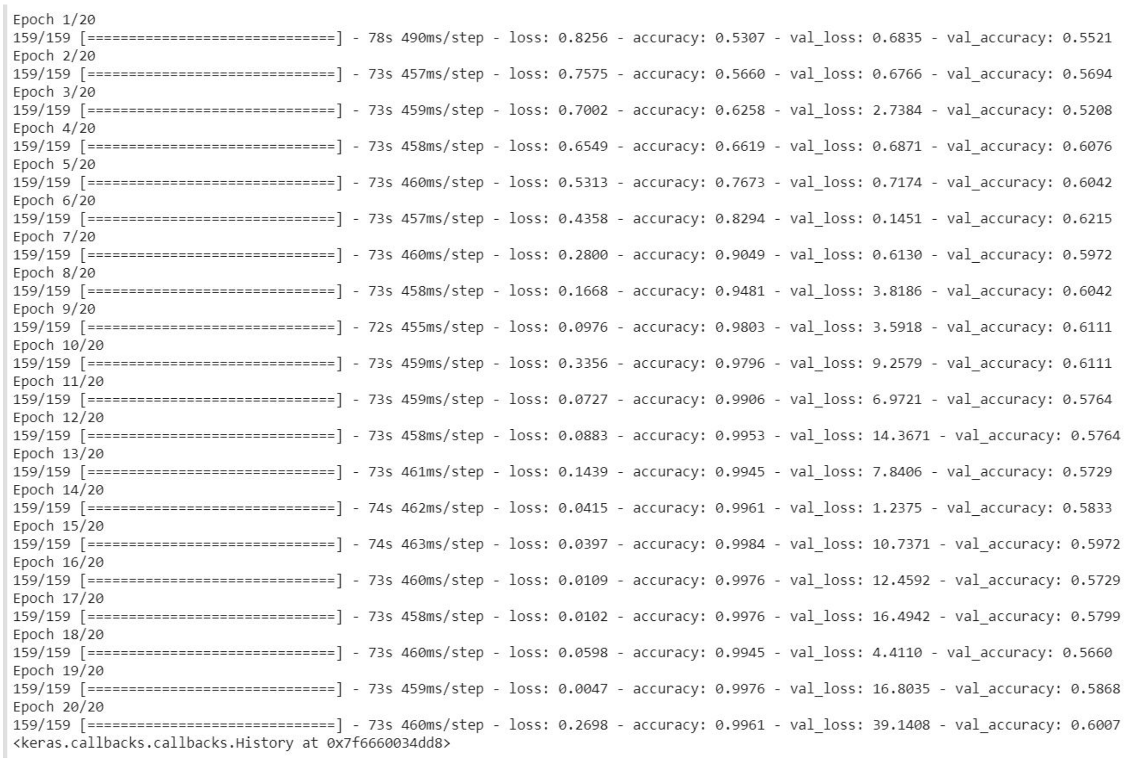


```
Epoch 1/20
159/159 [==============================] - 49s 306ms/step - loss: 1.5815 - accuracy: 0.5165 - val_loss: 0.6935 - val_accuracy: 0.5139
Epoch 2/20
159/159 [==============================] - 45s 284ms/step - loss: 0.7921 - accuracy: 0.5314 - val_loss: 0.6874 - val_accuracy: 0.5139
Epoch 3/20
159/159 [==============================] - 45s 284ms/step - loss: 0.7306 - accuracy: 0.5204 - val_loss: 0.6933 - val_accuracy: 0.5069
Epoch 4/20
159/159 [==============================] - 45s 285ms/step - loss: 0.7943 - accuracy: 0.5786 - val_loss: 0.6805 - val_accuracy: 0.5799
Epoch 5/20
159/159 [==============================] - 45s 284ms/step - loss: 0.7044 - accuracy: 0.6274 - val_loss: 0.7014 - val_accuracy: 0.6181
Epoch 6/20
159/159 [==============================] - 45s 285ms/step - loss: 0.6502 - accuracy: 0.6973 - val_loss: 0.6380 - val_accuracy: 0.5903
Epoch 7/20
159/159 [==============================] - 45s 284ms/step - loss: 0.5678 - accuracy: 0.7461 - val_loss: 0.5671 - val_accuracy: 0.6215
Epoch 8/20
159/159 [==============================] - 45s 286ms/step - loss: 0.4376 - accuracy: 0.8396 - val_loss: 0.6018 - val_accuracy: 0.6354
Epoch 9/20
159/159 [==============================] - 45s 284ms/step - loss: 0.2453 - accuracy: 0.9112 - val_loss: 0.5719 - val_accuracy: 0.6181
Epoch 10/20
159/159 [==============================] - 45s 285ms/step - loss: 0.1748 - accuracy: 0.9497 - val_loss: 2.7068 - val_accuracy: 0.6181
Epoch 11/20
159/159 [==============================] - 45s 284ms/step - loss: 0.0988 - accuracy: 0.9819 - val_loss: 2.1400 - val_accuracy: 0.6215
Epoch 12/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0776 - accuracy: 0.9858 - val_loss: 6.7657 - val_accuracy: 0.6111
Epoch 13/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0752 - accuracy: 0.9858 - val_loss: 1.9931 - val_accuracy: 0.6007
Epoch 14/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0176 - accuracy: 0.9969 - val_loss: 6.2070 - val_accuracy: 0.5972
Epoch 15/20
159/159 [==============================] - 45s 284ms/step - loss: 0.0475 - accuracy: 0.9929 - val_loss: 3.0690 - val_accuracy: 0.6111
Epoch 16/20
159/159 [==============================] - 45s 284ms/step - loss: 0.0414 - accuracy: 0.9969 - val_loss: 4.3477 - val_accuracy: 0.6007
Epoch 17/20
159/159 [==============================] - 45s 284ms/step - loss: 0.0705 - accuracy: 0.9937 - val_loss: 14.8269 - val_accuracy: 0.6076
Epoch 18/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0505 - accuracy: 0.9969 - val_loss: 10.1241 - val_accuracy: 0.6111
Epoch 19/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0073 - accuracy: 0.9984 - val_loss: 13.0370 - val_accuracy: 0.5938
Epoch 20/20
159/159 [==============================] - 45s 285ms/step - loss: 0.0146 - accuracy: 0.9976 - val_loss: 5.9283 - val_accuracy: 0.6146
<keras.callbacks.callbacks.History at 0x7f98702aed30>
```

We can see that the accuracy finally converges to about 62% which is a 7 % improvement to the original one. The canny/contour preprocessing actually helped us to enhance the features needed to detect broken bones.

**Result of DIfference of Gaussian Preprocessed Images**

The DoG preprocessing takes a longer time so for each epoch it takes about 75s which will be a problem for the speed of performance. And here's the result after 20 epoches:



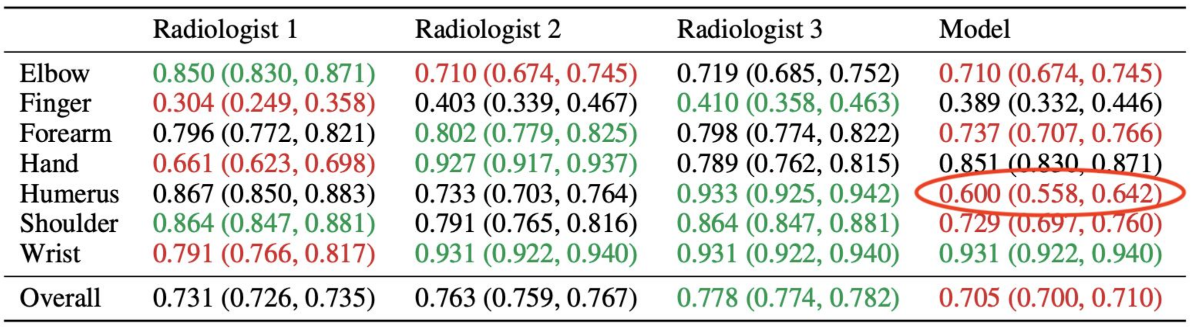
```
Epoch 1/20
159/159 [==============================] - 78s 490ms/step - loss: 0.8256 - accuracy: 0.5307 - val_loss: 0.6835 - val_accuracy: 0.5521
Epoch 2/20
159/159 [==============================] - 73s 457ms/step - loss: 0.7575 - accuracy: 0.5660 - val_loss: 0.6766 - val_accuracy: 0.5694
Epoch 3/20
159/159 [==============================] - 73s 459ms/step - loss: 0.7002 - accuracy: 0.6258 - val_loss: 2.7384 - val_accuracy: 0.5208
Epoch 4/20
159/159 [==============================] - 73s 458ms/step - loss: 0.6549 - accuracy: 0.6619 - val_loss: 0.6871 - val_accuracy: 0.6076
Epoch 5/20
159/159 [==============================] - 73s 460ms/step - loss: 0.5313 - accuracy: 0.7673 - val_loss: 0.7174 - val_accuracy: 0.6042
Epoch 6/20
159/159 [==============================] - 73s 457ms/step - loss: 0.4358 - accuracy: 0.8294 - val_loss: 0.1451 - val_accuracy: 0.6215
Epoch 7/20
159/159 [==============================] - 73s 460ms/step - loss: 0.2800 - accuracy: 0.9049 - val_loss: 0.6130 - val_accuracy: 0.5972
Epoch 8/20
159/159 [==============================] - 73s 458ms/step - loss: 0.1668 - accuracy: 0.9481 - val_loss: 3.8186 - val_accuracy: 0.6042
Epoch 9/20
159/159 [==============================] - 72s 455ms/step - loss: 0.0976 - accuracy: 0.9803 - val_loss: 3.5918 - val_accuracy: 0.6111
Epoch 10/20
159/159 [==============================] - 73s 459ms/step - loss: 0.3356 - accuracy: 0.9796 - val_loss: 9.2579 - val_accuracy: 0.6111
Epoch 11/20
159/159 [==============================] - 73s 459ms/step - loss: 0.0727 - accuracy: 0.9906 - val_loss: 6.9721 - val_accuracy: 0.5764
Epoch 12/20
159/159 [==============================] - 73s 458ms/step - loss: 0.0883 - accuracy: 0.9953 - val_loss: 14.3671 - val_accuracy: 0.5764
Epoch 13/20
159/159 [==============================] - 73s 461ms/step - loss: 0.1439 - accuracy: 0.9945 - val_loss: 7.8406 - val_accuracy: 0.5729
Epoch 14/20
159/159 [==============================] - 74s 462ms/step - loss: 0.0415 - accuracy: 0.9961 - val_loss: 1.2375 - val_accuracy: 0.5833
Epoch 15/20
159/159 [==============================] - 74s 463ms/step - loss: 0.0397 - accuracy: 0.9984 - val_loss: 10.7371 - val_accuracy: 0.5972
Epoch 16/20
159/159 [==============================] - 73s 460ms/step - loss: 0.0109 - accuracy: 0.9976 - val_loss: 12.4592 - val_accuracy: 0.5729
Epoch 17/20
159/159 [==============================] - 73s 458ms/step - loss: 0.0102 - accuracy: 0.9976 - val_loss: 16.4942 - val_accuracy: 0.5799
Epoch 18/20
159/159 [==============================] - 73s 460ms/step - loss: 0.0598 - accuracy: 0.9945 - val_loss: 4.4110 - val_accuracy: 0.5660
Epoch 19/20
159/159 [==============================] - 73s 459ms/step - loss: 0.0047 - accuracy: 0.9976 - val_loss: 16.8035 - val_accuracy: 0.5868
Epoch 20/20
159/159 [==============================] - 73s 460ms/step - loss: 0.2698 - accuracy: 0.9961 - val_loss: 39.1408 - val_accuracy: 0.6007
<keras.callbacks.callbacks.History at 0x7f6660034dd8>
```

The overall accuracy on the test set is about 59% Still, there's small improvement after the DoG preprocessing but it's not as good as the canny/contour one.

Like our eyes can see features most clearly on the canny/contour processed images, the model works best on that one too. So after all, canny/contour processing is the most efficient and also the most accurate method.

## 5 Conclusions

The accuracy of our classifier over all the humerus bones was ~62%, which is actually better than the Stanford neural network model for the humerus (60%).



| | Radiologist 1 | Radiologist 2 | Radiologist 3 | Model |
|---|---|---|---|---|
| Elbow | 0.850 (0.830, 0.871) | 0.710 (0.674, 0.745) | 0.719 (0.685, 0.752) | 0.710 (0.674, 0.745) |
| Finger | 0.304 (0.249, 0.358) | 0.403 (0.339, 0.467) | 0.410 (0.358, 0.463) | 0.389 (0.332, 0.446) |
| Forearm | 0.796 (0.772, 0.821) | 0.802 (0.779, 0.825) | 0.798 (0.774, 0.822) | 0.737 (0.707, 0.766) |
| Hand | 0.661 (0.623, 0.698) | 0.927 (0.917, 0.937) | 0.789 (0.762, 0.815) | 0.851 (0.830, 0.871) |
| Humerus | 0.867 (0.850, 0.883) | 0.733 (0.703, 0.764) | 0.933 (0.925, 0.942) | 0.600 (0.558, 0.642) |
| Shoulder | 0.864 (0.847, 0.881) | 0.791 (0.765, 0.816) | 0.864 (0.847, 0.881) | 0.729 (0.697, 0.760) |
| Wrist | 0.791 (0.766, 0.817) | 0.931 (0.922, 0.940) | 0.931 (0.922, 0.940) | 0.931 (0.922, 0.940) |
| Overall | 0.731 (0.726, 0.735) | 0.763 (0.759, 0.767) | 0.778 (0.774, 0.782) | 0.705 (0.700, 0.710) |

Figure 4: Cohen's kappa statistic, describing the radiologists' agreement (first 3 columns) and its comparison with Stanford's CNN (right column) [2]

Although their CNN may train over many more bones all together, (1) the humerus is distinctly different when compared with most other upper body bones and (2) our model did not have manual feature extraction, so it is not unreasonable to say that our model is good or at the very least, comparable. But it is still worse than the Stanford radiologists themselves. Interestingly, of all of the bones, the skilled radiologists actually classified the wrist *and the humerus bones* the *best* [2].

## 6 Future Improvements

A different scale/orientation/flipping, brightness variability, and contrast of a bone could affect the CNN learning results. For example, by re-orienting a bone to fit a template using an algorithm like in SIFT [9], could result in more accurate results [1]. Another complication to this is the different sizes of bones from person-to-person.

Some abnormalities appear as missing parts of the bone edges. Sometimes these are difficult to detect, because the contour lines might be missing due to noise reduction or its lack of thickness. A more careful reevaluation of thresholding around these areas might elucidate a deformity, before tossing it out during the preprocessing stage.

Some fractures are less distinct than others: those across the shaft are easy to distinguish, while those near the rounded ends are more difficult to pinpoint. From prior knowledge of the curvature of a bone, one could traverse around the bone, comparing curvature to a template, until a spike in the difference in curvature is detected. These results could then be accentuated in the preprocessing stage to aid the CNN.

CNN: Adaptive learning rate, to adapt to the problem within fewer epochs.

CNN: Momentum for speed improvement.

CNN: K-fold validation to reduce biases between humerus bones, as the Stanford group has hidden their testing set.

### Some improvements required by radiology imaging

Multiple views of the bone from different angles would provide different perspectives of deformations.

How many views of these images from a single patient were trained upon? These should be limited, so the network does not overfit classification on the larger patient profiles.

## 7 Contrubution

Tianxiao: Coded and tested the CNN to classify X-ray bone images. Reorganized the preprocessing code so it can be feed into keras image generator directly without any extra work.

Alec:Coded/implemented preprocessing of MURA image data set by applying Difference of Gaussian (DoG) and Canny edge detection with a contour overlayed on the original image. The preprocessing was then applied to the CNN.

Ryan: Study journals of similar research. Test code for different preprocessing techniques. Assist in advising in the usefulness and critiquing various techniques. Write motivation, conclusion, and avenues for future improvements, as well as some technical purpose aspects and formulas in final report and presentation slides.

Hao: Find and study papers from related works. Implemented the project with some data mining algorithms based on different feature extraction techniques compared to the one we implemented in the final project. Write some parts of the final report and presentation slides.

## References

[1] "X-Ray Bone Fracture Classification Using Deep Learning: A Baseline for Designing a Reliable Approach - ProQuest." https://search-proquest-com.mutex.gmu.edu/docview/2365865302?accountid=14541&rfr_id=info%3Axri%2Fsid%3Aprimo (accessed May 06, 2020).

[2] P. Rajpurkar *et al.*, "MURA: Large Dataset for Abnormality Detection in Musculoskeletal Radiographs," *ArXiv171206957 Phys.*, May 2018, Accessed: May 05, 2020. [Online]. Available: http://arxiv.org/abs/1712.06957.

[3] A. Y. Yang and L. Cheng, "Long-Bone Fracture Detection using Artificial Neural Networks based on Contour Features of X-ray Images," *ArXiv190207897 Cs*, Feb. 2019, Accessed: May 06, 2020. [Online]. Available: http://arxiv.org/abs/1902.07897.

[4] A. Y. Yang and L. Cheng, "Long-Bone Fracture Detection using Artificial Neural Networks based on Line Features of X-ray Images," *ArXiv190207458 Cs*, Feb. 2019, Accessed: May 06, 2020. [Online]. Available: http://arxiv.org/abs/1902.07458.

[5] M. Pant, K. Ray, T. K. Sharma, S. Rawat, and A. Bandyopadhyay, Eds., *Soft Computing: Theories and Applications: Proceedings of SoCTA 2016, Volume 2*, vol. 584. Singapore: Springer Singapore, 2018.

[6] N. Khan, "BONE FRACTURE DETECTION USING OPENCV," Accessed: May 06, 2020. [Online]. Available: https://www.academia.edu/34833622/BONE_FRACTURE_DETECTION_USING_OPENCV.

[7] "CIFAR-10 CNN - Keras Documentation." https://keras.io/examples/cifar10_cnn/ (accessed May 07, 2020).

[8] "lecture_slides_lec6.pdf." Accessed: May 07, 2020. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[9] "Introduction to SIFT (Scale-Invariant Feature Transform) — OpenCV-Python Tutorials 1 documentation." https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html. (accessed May 07, 2020).