FracTect: A Multistage Neural Network to Classify and Detect Distal Radius Fractures in X-rays

Author(s): Jennifer Schofield[a,1]

Affiliations: [a] Department of Computer Science, University of Exeter, Exeter, EX4 4QJ

Corresponding Author: Jennifer Schofield, schofieldjenni@gmail.com

Abstract:

FracTect is a multistage neural network developed to aid medical professionals in diagnosing paediatric distal radius (wrist) fractures in X-rays. The publicly accessible retrospective GRAZPEDWRI-DX dataset was used, split 80/10/10 for training, testing, and validation sets at an image level. The classification model is a ResNeXt model trained on an evenly balanced dataset of fractured and unbroken X-rays, and the detection model is a Faster Region-Based Convolutional Neural Network (R-CNN) model trained on exclusively fractured X-rays. The classification model is a binary classification into Fractured or Not Fractured, with degrees of confidence given for its prediction, and the detection model can identify nine different classes possibly present in an X-ray. FracTect includes an associated front end, allowing users to upload X-rays for the models to evaluate and choose which classes to view in the detection model. The classification model had an Area Under the Receiver Operating Characteristic Curve (ROC-AUC) of 0.9317, an accuracy of 0.9318, a specificity of 0.9166, and a sensitivity of 0.9468, on par with state-of-the-art literature. The detection model had an overall Mean Average Precision (mAP) of 0.4831, a mAP@0.5 of 0.7933, and a mAP specifically to fractures of 0.4116. Overall, FracTect is a promising development in using artificial intelligence in the medical field, and with further development could be put into a live scenario. The code can be accessed at the linked GitHub repository, and the dataset can be accessed at the associated GRAZPEDWRI-DX paper.

Keywords: computer vision, medical AI, neural networks, X-rays, Faster R-CNN, deep convolutional neural networks, ResNeXt

[1]Is now affiliated with the School of Electrical and Electronic Engineering, University of Sheffield, Sheffield, S10 2TN

1 Introduction

Over a ten-year study, it was found that distal radius fractures were the second most seen fractures in the emergency room and the most common fracture for patients aged 14 or under (1). The Arbeitsgemeinschaft für Osteosynthesefragen/Orthopaedic Trauma Association, or AO/OTA, classifies radius and ulna fractures depending on where in the bone the fracture is. For example, a proximal fracture is a fracture closer to the elbow and is class 21 for paediatric care. Likewise, a diaphyseal fracture is in the centre of the radius/ulna and is class 22. Most importantly for this research, a distal fracture is closest to the hand, is class 23, and is also known as a wrist fracture. Medical professionals are put under immense stress in their average workday, especially when the consequences of misdiagnosis are severe. Misdiagnosis of a wrist fracture may cause long-term loss of function, periods of pain, and possibly death (2). With the intense workloads in many medical facilities, medical professionals cannot always get a second opinion on an X-ray, which would significantly reduce the risk of misdiagnosis (3). With this in mind, it falls to artificial intelligence to provide a secondary opinion to fill this void. FracTect is a proposed system in which a user may upload an X-ray image and receive opinions on whether or not it is fractured and if there are any objects within the X-ray to notice. FracTect also provides a friendly user interface required for its use in the medical field rather than relegating the software to those willing to configure it themselves. This paper discusses the design of FracTect, and various development decisions made, the dataset used to train the models, and the results gathered from various metrics. By writing this paper, the aim is to provide transparency to the possible users of FracTect, allowing users to understand why the system may perform in certain ways instead of creating a black-box system and expecting users to trust the results. The objective of FracTect's development was to create a two-stage system: one to classify the X-ray into Fractured or Not Fractured, with a confidence level to reflect its prediction accurately, and the second stage to detect where objects of interest on a fractured X-ray are. For ease of use, the system is then connected to a front end, allowing users to personalise what they would like displayed and give access to documentation and support if needed.

2 Methodology

2.1 Dataset Balance

The initial GRAZPEDWRI-DX dataset was well-prepared and had high-quality tags associated with each image (4). The dataset was created retrospectively from archived X-ray images from the University Hospital of Graz's pediatric department. Upon examining the dataset for any underrepresented classes, the split between fractured and non-fractured X-rays was 69.7% fractured to 30.3% non-fractured out of a dataset size of 20,327 images. When subdividing the dataset into single fracture, multiple fracture, and unbroken X-rays, the split was 42.7% single fracture X-rays, 30.3% unbroken X-rays, and 26.9% multiple fracture X-rays. This would prove essential for development, as only the single-fracture subclass would be used to reduce the time needed to train the model. Later, the multiple-fracture subclass was readded to the dataset once the code structure had been established. In both cases, whether the entire fractured class was used or just the single fracture class, the minority class was the Not Fractured class, which would require augmentation to correctly balance the classes for training the classification model. After acknowledging the initial split between the two classes, the Fractured class was examined closer to see where possible biases would exist. An overwhelming 95.7% of the fracture types were of OTA classification 23, with OTA classification 22 making up 3.2%. The remaining 1.1% were sparse examples of classification 72 (scaphoid), 77 (metacarpal), and 76 (other carpal bone) fractures. For multiple fracture X-rays, the case was almost the same – 95.7% contained at least one classification 23 fracture, and 4.1% contained at least one classification 22 fracture.

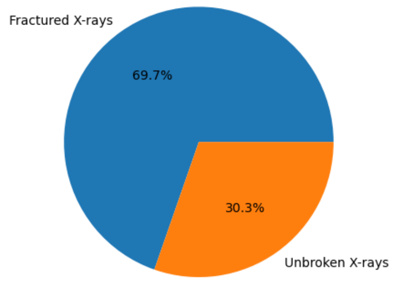


Figure 1 - The initial balance between Fractured and Unbroken X-rays

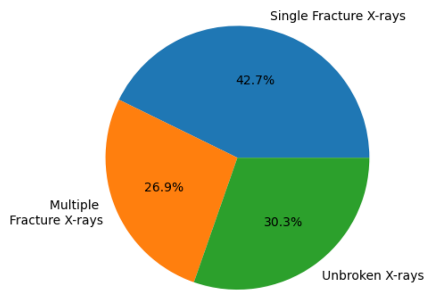


Figure 2 - The initial balance between Single Fracture, Multiple Fracture, and Unbroken X-rays

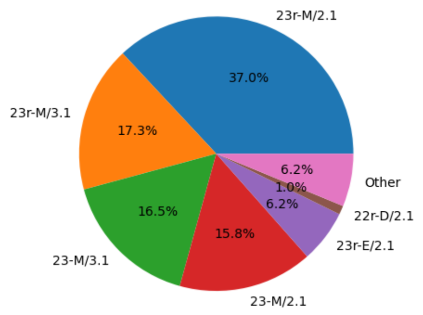


Figure 3 - AO/OTA breakdown for single fractures

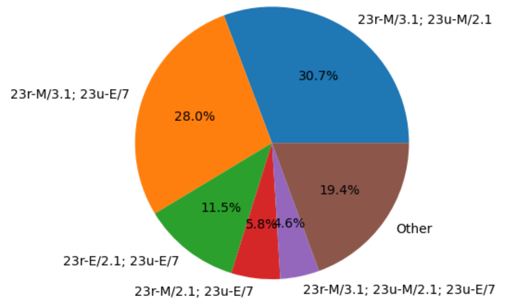


Figure 4 - AO/OTA breakdown for multiple fractures

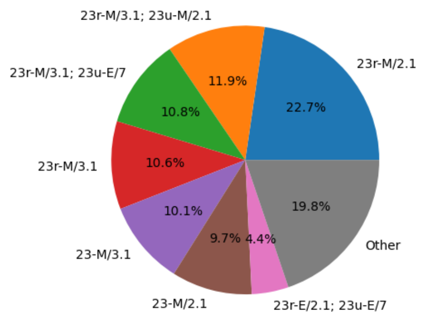


Figure 5 - AO/OTA breakdown for all fractures

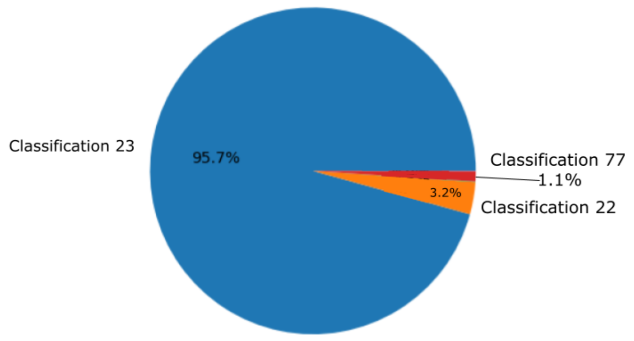


Figure 6 - AO/OTA classification group for single fractures

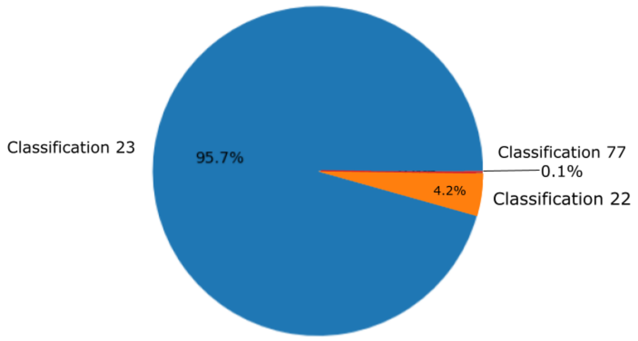


Figure 7 - AO/OTA classification groups in multiple fractures

Another important distinction was the X-ray's projection type – wrist X-rays can be anteroposterior, lateral, or oblique. The GRAZPEDWRI-DX dataset represented anteroposterior and lateral well, with an almost 50-50 split, but only had 93 examples of oblique projection X-rays. However, oblique X-rays are only sometimes considered standard views for wrist X-rays, so this underrepresentation is noted but is unlikely to cause drastic impacts. It must also be noted that although the tag associated with certain X-rays is "projection-ap", these images reflect a posteroanterior view. An anteroposterior view is taken with the back of the hand facing the table, and a posteroanterior view is taken with the palm facing the table. The differences in the X-ray are simply which side of the X-ray the radius is and which side the ulna is. Ultimately, it makes little difference to the neural networks but should be mentioned further upstream to avoid confusing medical professionals.

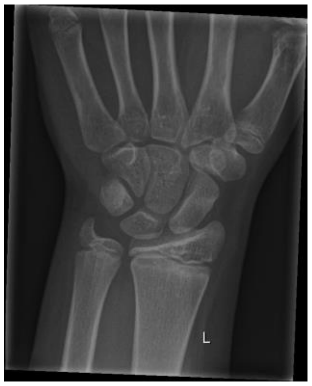


Figure 8 - An anteroposterior view wrist X-ray

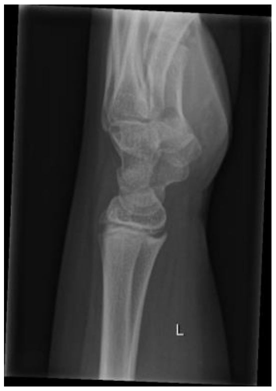


Figure 9 - A lateral view wrist X-ray

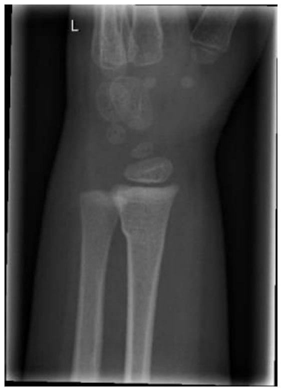


Figure 10 - An oblique view wrist X-ray

Further research would benefit from all three views being fully represented. Initially, to decrease training time, the model was trained on 8,689 single fracture X-rays and 8,689 unbroken X-rays, which had been augmented from 6,169 images to match the size of the majority class. Later, after an initial model was trained, the multi-fracture X-rays were added back into the class for a total of 14,189 images, and the model was retrained to include these images. The unbroken class was augmented once more to match the size of the majority class. The dataset was then split into a training, testing, and validation set, with an 80/10/10 split, as is standard, using Python's split-folders library. Other cases were also present in the image tags, such as a cast being present or if the initial assessor was uncertain about the diagnosis. Further development may find improvement in augmenting the number of X-rays with multiple fractures to match the number of single fracture X-rays or by augmenting the number of X-rays with casts present to expose the system to lower-quality X-rays. There may also be benefits in using "diagnosis uncertain" X-rays to test where human diagnosticians fail and measure against how the model classifies them.

For the detection model, data augmentation was unnecessary, as all the fractured images – regardless of how many fractures were present – were used for a total of 14,189 images. Similar to the classification model, future development may benefit from augmenting the multiple fracture images to match the single fracture images and by including a higher percentage of X-rays with casts. However, with the detection model, some fractured images did not have an associated fracture bounding box, rendering them useless for training purposes. These images were generated when the medical professional tagging the images wasn't fully certain where the fracture was (4). Out of 14,158 initial images used for the detection model, 773 did not have the fracture bounding box. This was a small subset; thus, these images were discarded from the dataset, reducing the dataset size to 13,385. After this, the dataset was split into training, testing, and validation sets with a standard 80/10/10 balance using Python's split-folders library. Both of these splits happened at the image level, but before any training took place, the split was manually examined to ensure that the last patient in the training set had all the views in the training set to prevent possible contamination of the testing or validation sets.
The use of the GRAZPEDWRI-DX dataset did not require additional ethical approval or informed consent, as the relevant approvals were done by the creators of the dataset, and the dataset was then fully anonymised. The ethical approval and anonymisation steps can be seen in the respective paper (4).

2.2 Data Augmentation

To balance the two classes for the classification model, various methods were used, demonstrated by Joshi *et al.*, to accurately simulate how an X-ray may be altered in real life (5). These steps included adjusting the contrast of the image by ±11%, the hue by ±44°, flipping the image horizontally or vertically, rotating it by ±9°, zooming in by ± 9%, and adjusting the saturation of the image by ±42%. Each original image was adjusted no more than twice to ensure that the model did not overfit. A further exploration would be to augment both the minority and majority classes by combining different augmentations rather than just one per image. However, due to time and processing power restraints, any further augmentation would have rendered the dataset too large to work with.

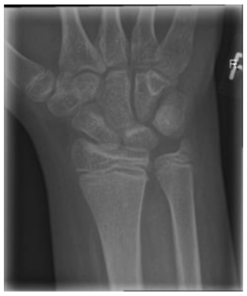
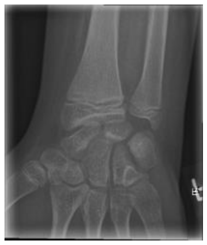
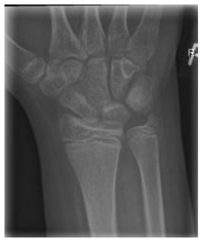
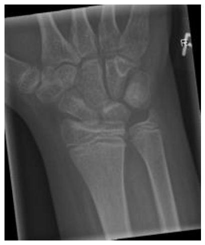
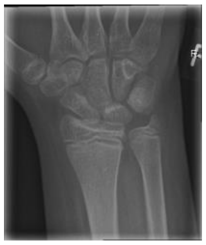
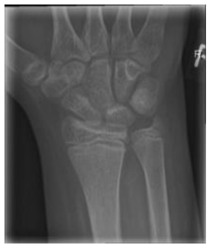
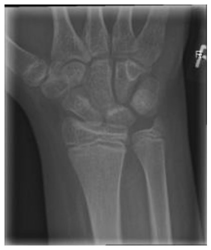
 *Figure 11 - Original image to illustrate augmentations*	 *Figure 12 - Horizontal flip augmentation*	 *Figure 13 - Zoom augmentation by 1%*	 *Figure 14 - Rotation augmentation by 7°*
	 *Figure 15 - Contrast augmentation by -11%*	 *Figure 16 - Hue augmentation by -20°*	 *Figure 17 - Saturation augmentation by -19%*

Table 1 – Table containing examples of possible augmentations for an image

2.3 Data Preprocessing

The GRAZPEDWRI-DX dataset is initially in 16-bit greyscale PNG format, which conflicts with how PyTorch handles images internally. PyTorch's DataLoader uses the Python Imaging Library (PIL) as the structure for handling images, and PIL is designed to mainly work with JPG images, only allowing direct conversion from 8-

bit PNG images to JPG. After a few tests, it was clear that the initial images needed to be transformed from 16-bit PNG to JPG before inputting them into the model. However, during the augmentation steps, which use the OpenCV library, the images were automatically converted to 8-bit PNG images. The conversion had to examine the file name structure to ensure that the 16-bit original images had a bitwise image swap occurring, while the 8-bit augmented images were normally converted using the OpenCV library. This was the issue that took the longest to fix, as the original images appeared to be acceptable upon entering the neural network, and it was only internally to PyTorch's DataLoader that the incorrect conversion was occurring, and the results from testing seemed within limits. In future projects, it would be advisable to convert all images to JPG format before augmentation, as both PyTorch and TensorFlow libraries use PIL to handle images, and it would also be advisable to avoid any conversion errors in colour adjustments when displaying images.
The classification model's preprocessing steps were simpler than most literature, mainly consisting of adjusting the input image from a skewed rectangle to a 256px by 256px square. The process had the unintended side effect of over-rotating the image occasionally, which was compensated by allowing the user to adjust the image if necessary. The image is then converted to a FloatTensor, and as a side effect, it inverts the colours. This is compensated for by inverting the image's colours in the front end before overlaying the Grad-CAM result on top. The process can be seen in Table 2, with Figures 18, 19, 20, and the final result of Figure 21.

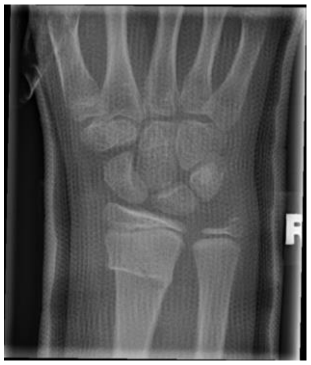
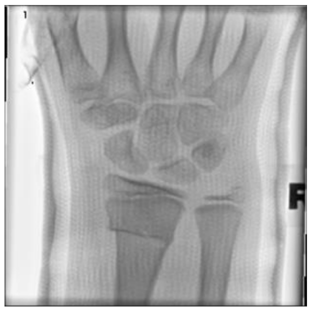
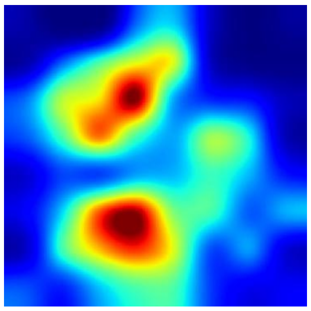
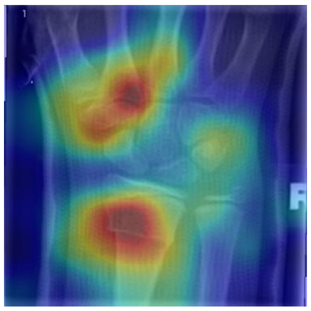
 *Figure 18 - The original image, before transforms*	 *Figure 19- The image, post-resizing and conversion to FloatTensor*	 *Figure 20 - The Grad-CAM heatmap generated*	 *Figure 21 - The final image, with the X-ray inverted and the heatmap overlaid.*

Table 2 – The process of generating Grad-CAM results with FloatTensor corruption

The method used to preprocess the classification model's images couldn't be used for the detection model as the associated bounding boxes would be misaligned. Instead, transformations that mimic how an X-ray may be corrupted in practice were applied. Prior to any transforms, the images were resized to 512px by 512px. The Albumentations library, used for the preprocessing steps, can adjust the bounding boxes along with the transforms, allowing for the system to build resilience. The steps included a 50% chance of flipping the image, a 50% chance of randomly rotating the image, a 20% chance of applying motion blur, a 10% chance of applying median blur, a 10% chance of normally blurring the image, and then converting the image to a Tensor. However, these transforms were only applied while training the model; when the model was being validated, the image was simply converted to a FloatTensor.

2.4 Model Creation

For the classification model, the pre-trained ResNeXt-50 model from PyTorch was used, and for the detection model, the pre-trained Faster R-CNN version 2 from PyTorch was used. By using the models that were already pre-trained on ImageNet, it enabled transfer learning. Transfer learning is when a model is pre-trained on a separate task, in this case, identifying ImageNet images and then reused on a separate task for better results. For FracTect, it allowed the model to be fine-tuned to X-ray images rather than starting completely from scratch.
The classification model used PyTorch's implementation of cross-entropy loss for its loss function, the metric to measure how well an algorithm models the data, and the Adam algorithm as the optimiser for that loss function (6). It also made use of a scheduler to reduce the learning rate when it began to plateau. The initial learning rate,

or how much the model is adjusted when it makes an error, was 0.0001, which would decrease by 10% when the learning plateaued. The batch size, or the number of images to go through before updating the model, was initially restricted to 16 due to lack of processing power, but after swapping from a local machine equipped with an NVIDIA GeForce RTX 3060 to Google Colab's A100 GPU, the batch size was increased to 32 with significant improvements. The classification model was trained for 50 epochs, but after 30 epochs, there was no significant improvement in accuracy. More experiments with batch sizes would be useful for future research, but in the scope of this project, the time and processing limitations restricted further tests. In contrast to the classification model, the detection model used a batch size of 16, as anything higher would overload even the High RAM versions of Colab's GPUs. The detection model took significantly longer than the classification model, and due to this, the decision was made to only train the model for 30 epochs and reevaluate if the results were significantly poorer. After 30 epochs, the detection model was well-trained enough to continue with the overall creation of FracTect, with the reassurance that if it was performing poorly in practical tests, it could always be trained further later on. The GRAZPEDWRI-DX dataset came prepared with bounding boxes in the PASCAL VOC challenge's XML format, which allowed the model not only to learn to detect fractures but also to detect bone lesions, bone anomalies, foreign bodies, metal, periosteal reactions, pronator signs, soft tissue, and text. However, since the dataset was comprised of fractured X-rays, the balance between the other classes was skewed. For example, out of the entire dataset of 20,327 images, only eight images contained foreign bodies. Ultimately, FracTect was not designed to detect the other classes, and having these extra categories helped to avoid false positives for fractures, a worst-case scenario. Using PyTorch's models, rather than constructing the models wholesale, allowed valuable time to be put towards other aspects, such as making the front end more user friendly and covering more edge cases in the front end.

2.5 Grad-CAM Development

Grad-CAM was chosen to add explainability to the classification model due to the ease of understanding for users (7). It generates an easy-to-understand heatmap, showing where the model is activating, giving some clarity to the user about a model's predictions. The Grad-CAM implementation was quickly created but, in practice, often did not show the level of specificity for users to correctly identify where a small fracture would be. However, it did provide necessary transparency to users on where it was looking in case of false positives. One example is how, with a younger child, the carpal bones aren't fully formed nor fused (Figure 22), which leads to a false positive (Figure 23). However, the detection model correctly identifies the fracture in its stead (Figure 24). This false positive case is shown in Table 3, where the patient has yet to form carpal bones, and the Grad-CAM implementation misclassifies the X-ray.

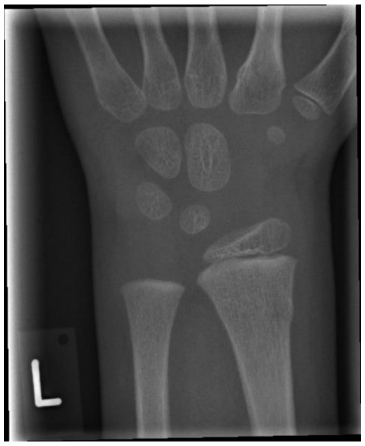
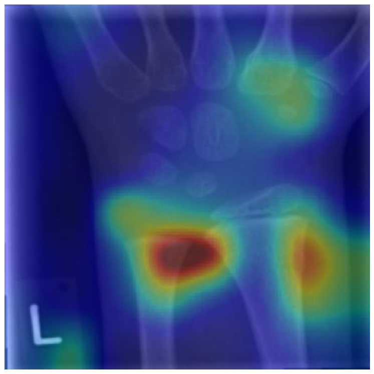
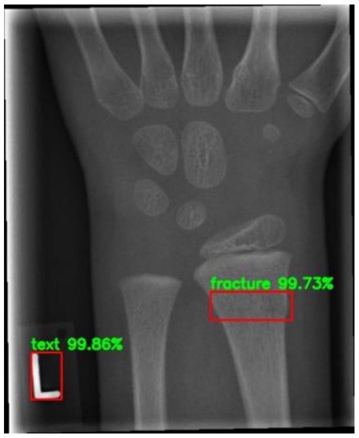
 *Figure 22 -The original image with not fully formed carpal bones*	 *Figure 23 - The Grad-CAM incorrectly identifies an X-ray as Not Fractured with 64.21% confidence*	 *Figure 24 - The detection model correctly identifies the fracture with 99.73% confidence*

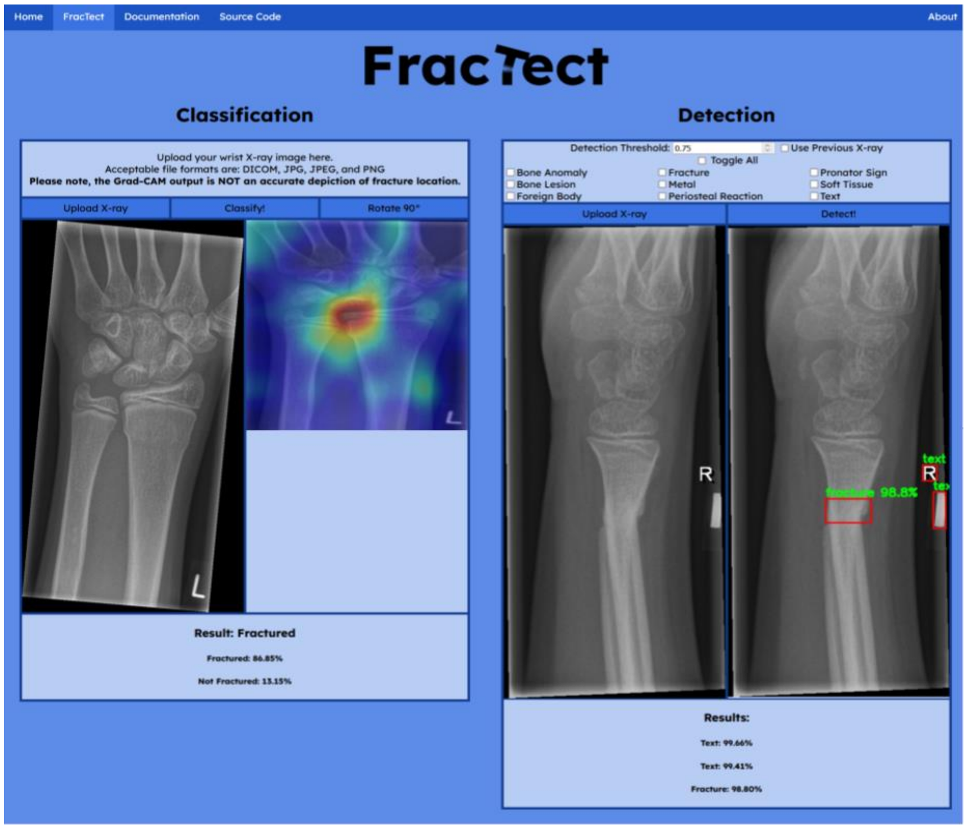
Table 3 – Grad-CAM fail case where it failed to detect a torus radius fracture due to undeveloped carpal bones

2.6 Front End Development

The front end was created from scratch using HTML, CSS, and occasional instances of JavaScript to aid functionality. The front end was designed in such a way that anyone could use it, not exclusively medical professionals, while also giving the appearance of medical software (Figure 25, Figure 26). The various shades of blue made it more appealing than stark black text on a white background. The user is also given choices where possible, mainly in the detection model's panel. The panel allows the user to adjust the detection threshold and select which objects they wish to see in the result. The result is then displayed with the predictions and associated confidence scores. For transparency, each prediction also displays the model's confidence in its prediction. For the classification model, this allowed the result to be grouped into three subclasses – Fractured/Not Fractured, where the confidence was over 85%; More Likely Fractured/Not Fractured, where the confidence was over 60% but less than 85%; and Unsure, where the confidence was under 60%. Users are also able to see the exact percentage under the image as well. The thresholds of over 60% and over 85% were arbitrarily chosen based on my assessment of how confident the model should be in its assessments, but users would benefit from the customisation. However, configuring user preferences that persist over sessions is out of the scope of this project. A further extension would implement feedback and allow other colour schemes to be added, as well as allow a user to define the thresholds for the various classifications for the classification model. In further development, it would be ideal to implement a user palette, as well as include various accessibility options, such as a dyslexia-friendly font, high-contrast colours, and screen-reader capabilities.

2.7 Web App Construction

The final step in making FracTect was to connect the front end with the models in the back end. The Flask framework was chosen as the web app architecture due to prior experience with it, and it allowed for minimal overhead for joining the models to the website design. However, time and budget constraints did not allow for publicly hosting the website, which forces users to run it locally. Ideally, for a future developed project, it would be run on an internal server to the hospital, protecting patients' privacy while allowing medical professionals to simply access FracTect rather than having to go through a lengthy download process. However, within this project's scope, locally running FracTect is adequate.



Upload your wrist X-ray image here.
Acceptable file formats are: DICOM, JPG, JPEG, and PNG
Please note, the Grad-CAM output is NOT an accurate depiction of fracture location.

Detection Threshold: 0.75 ☐ Use Previous X-ray
☐ Toggle All
☐ Bone Anomaly ☐ Fracture ☐ Pronator Sign
☐ Bone Lesion ☐ Metal ☐ Soft Tissue
☐ Foreign Body ☐ Periosteal Reaction ☐ Text

Result: Fractured

Fractured: 86.85%

Not Fractured: 13.15%

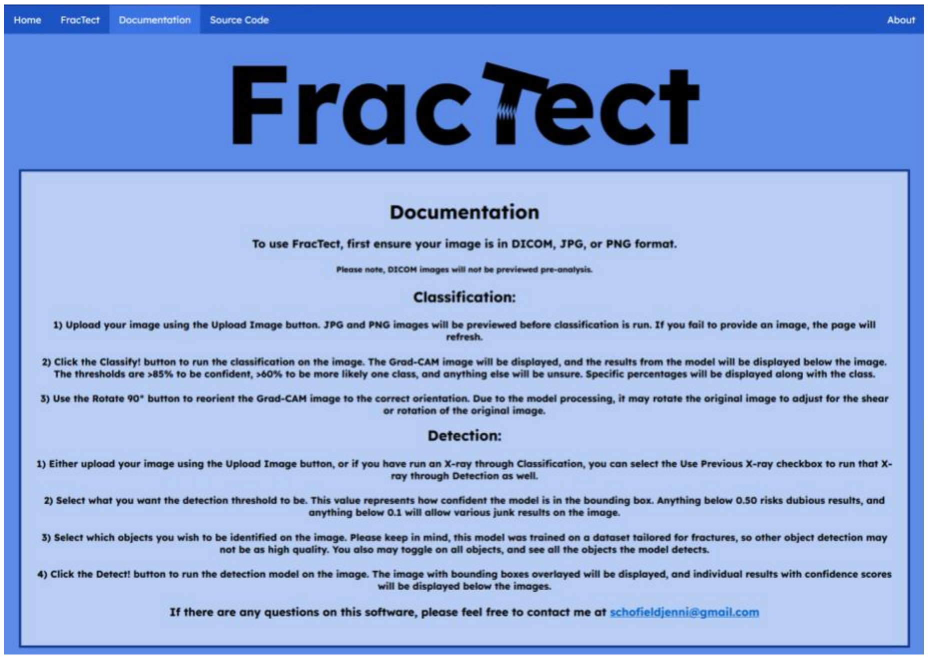
Results:

Text: 99.66%

Text: 99.41%

Fracture: 98.80%

Figure 25 - The main page of FracTect, with a classification result and a detection result



Documentation

To use FracTect, first ensure your image is in DICOM, JPG, or PNG format.

Please note, DICOM images will not be previewed pre-analysis.

Classification:

1) Upload your image using the Upload Image button. JPG and PNG images will be previewed before classification is run. If you fail to provide an image, the page will refresh.

2) Click the Classify! button to run the classification on the image. The Grad-CAM image will be displayed, and the results from the model will be displayed below the image. The thresholds are >85% to be confident, >60% to be more likely one class, and anything else will be unsure. Specific percentages will be displayed along with the class.

3) Use the Rotate 90° button to reorient the Grad-CAM image to the correct orientation. Due to the model processing, it may rotate the original image to adjust for the shear or rotation of the original image.

Detection:

1) Either upload your image using the Upload Image button, or if you have run an X-ray through Classification, you can select the Use Previous X-ray checkbox to run that X-ray through Detection as well.

2) Select what you want the detection threshold to be. This value represents how confident the model is in the bounding box. Anything below 0.50 risks dubious results, and anything below 0.1 will allow various junk results on the image.

3) Select which objects you wish to be identified on the image. Please keep in mind, this model was trained on a dataset tailored for fractures, so other object detection may not be as high quality. You also may toggle on all objects, and see all the objects the model detects.

4) Click the Detect! button to run the detection model on the image. The image with bounding boxes overlayed will be displayed, and individual results with confidence scores will be displayed below the images.

If there are any questions on this software, please feel free to contact me at schofieldjenni@gmail.com

Figure 26 - The documentation page of FracTect, giving users an overview of how to use FracTect

2.8 Resources

One of the preliminary choices for FracTect was which library to use – PyTorch or TensorFlow. After initial attempts at using TensorFlow and multiple conflicts with other modules, PyTorch was chosen as the basic library. PyTorch also offered pre-constructed and pre-trained models of ResNeXt-50 and Faster R-CNN, whereas TensorFlow only offered a Faster R-CNN implementation, which was not pre-trained. However, for data augmentation, elements from TensorFlow were used to augment the images in specific ways – namely, adjusting the saturation, hue, and contrast of an image.
The augmentation, dataset balancing, and training code were all initially created in a Jupyter notebook locally, which allowed for flexibility when trialling different segments of the code instead of having to set up a full Python file structure while experimenting with a new software library. The initial development took place on a Windows PC with an NVIDIA GeForce RTX 3060, running Python 3.12. After organising the initial dataset balance, the notebook was moved to Google Colab Pro+ to use Colab's GPUs, specifically the A100 GPU running Python 3.10, due to local resources not being available. Using Colab brought its own problems, as the GPU runtime would often time out, causing a long-running model development to be lost. Multiple times, a model would be left to train for multiple hours, only for the saving to fail, leading to long delays. Future development would ideally use on-site hardware instead of relying on a third-party service. However, once the initial models were trained, the swap from a Jupyter notebook to a full Python file structure was made for multiple reasons. Firstly, Colab did not support linking in the necessary external files for evaluating the detection model, which was needed to save time calculating model results. Secondly, the Flask framework was used to combine the front end with the back end, and refactoring it into separate Python files increased the overall legibility of the code and made it much easier to integrate with Flask. Finally, the code needed to be refactored into separate files for ease of initialisation, as the required module would have to be installed locally, and time constraints did not allow for automating this process.

3 Results

3.1 Classification Metrics and Results

A binary classification model, such as the one present in FracTect, can be measured by calculating various statistics from the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). A true positive is when the model correctly classifies an image in the positive class – for FracTect, the positive class is Not Fractured, and the negative class is Fractured. This is due to PyTorch assigning Not Fractured as the positive class, with the value of 1, but the initial creation of the DataLoader uses zero-based indexing, assigning Fractured the value of 0, and thus the negative class. However, in medical terminology, a positive result would be a test showing that the condition did exist, meaning that in future research, the classes should be inverted at the start. A true negative is when the model correctly predicts the negative class. A false positive is when the model incorrectly predicts the positive class for a negative class. A false negative is when the model incorrectly predicts the negative class for a positive class. A confusion matrix is used to display these results, where the test set is run to determine each quantity. The matrix can show if a model is solely predicting one class or favouring another. Ideally, the diagonals should be starkly different colours, as seen in Figure 27.

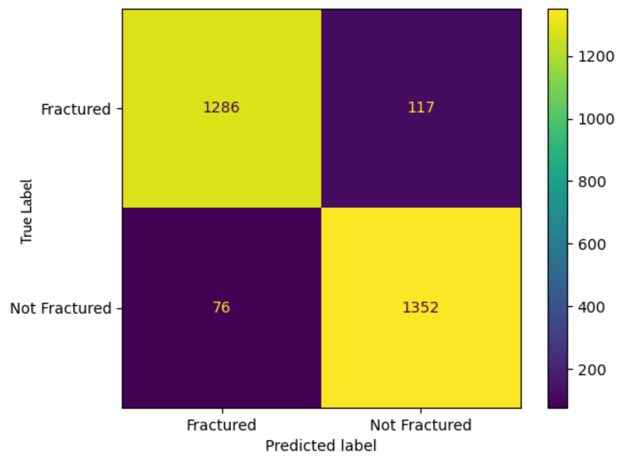


Figure 27- The confusion matrix for the classification model

FPR = FP / (FP + TN)	FNR = FN / (FN + TP)	Precision = TP / (TP + FP)	NPV = TN / (TN + FN)
TPV = TP / (TP + FN)	TNR = TN / (TN + FP)	F1 Score = (2 × Precision × Recall) / (Precision + Recall)	Accuracy = (TN + TP) / (TN + TP + FP+ FN)

Table 4 – Formulas for non-iterative metrics

The False Positive Rate (FPR), or Type I errors, is the probability that the model will incorrectly predict the positive class instead of the negative class. The formula divides the number of false positives by the sum of the number of true negatives and the number of false positives. Conversely, the False Negative Rate (FNR), or Type II errors, is the probability that the model will incorrectly predict the negative class instead of the positive class. It is calculated by dividing the number of false negatives by the sum of the number of true positives and the number of false negatives. Precision, or the positive predictive value (PPV), measures how many positive predictions are actually positive and is calculated by the number of true positives divided by the number of true positives and false positives. The Negative Predictive Value (NPV) is similar to precision but measures the number of true negatives over the number of true negatives plus the number of false negatives. Recall, or the true positive rate (TPV), or sensitivity, is the measure of out of all the real positive cases, how many are predicted positive, and is calculated by dividing the number of true positives by the true positives plus false negatives. Similarly, specificity, or the true negative rate (TNR), is the number of true negatives divided by true negatives and false positives. However, these metrics by themselves do not help to fully evaluate a model's success; for this, we use accuracy, the F1-Score, and the ROC-AUC value. Accuracy is the measure of how often the model is correct. It is calculated by dividing the number of true positives and true negatives by the total number of predictions. A good overall metric is the F1-Score, which is the harmonic mean of precision and recall. It is calculated by 2 times precision times recall and divided by precision plus recall. Finally, the Area Under the Receiver Operating Characteristic Curve (ROC-AUC) is the area under the curve generated by plotting the true positive rate, or sensitivity, against the false positive rate. The ROC-AUC value is a good way to evaluate the model's overall performance, especially when used in tandem with the F1-Score. The Average Precision (AP) is a similar metric determined by calculating the precision at different thresholds on the graph of precision versus recall. Average Precision gives a better reflection of how the model handles the positive class, whereas the ROC-AUC value reflects both the positive and negative classes. The formulas used can be seen in Table 4, and the classification model's results can be seen in Table 5. Example outputs of the model can be seen in Table 6 with Figures 28, 29, and 30.

False Positive Rate	False Negative Rate	Precision	Negative Predictive Value	Recall
0.0834	0.0532	0.9204	0.9442	0.9468
Specificity	**F1-Score**	**Accuracy**	**ROC-AUC**	**Average Precision**
0.9166	0.9333	0.9318	0.9317	0.8982

Table 5 – Table of results from the classification model

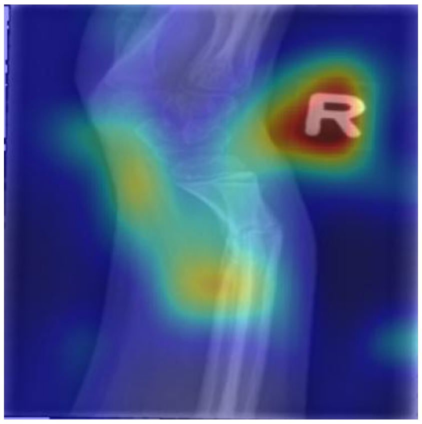
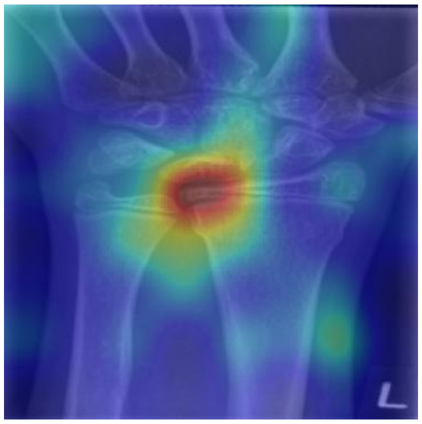
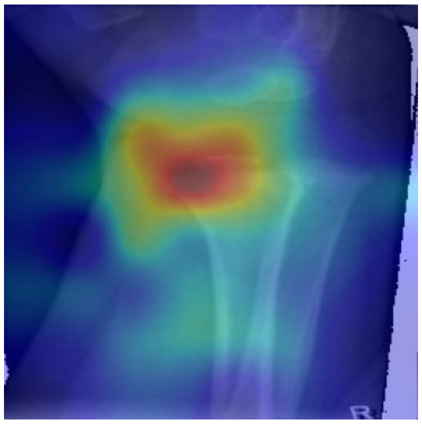
 *Figure 28 - A single fracture X-ray, identified as "Fractured" with 93.20% confidence*	 *Figure 29 - A multiple fracture X-ray, identified as "Fractured" with 86.97% confidence*	 *Figure 30 - An unbroken X-ray, identified as "Not Fractured" with 85.27% confidence*

Table 6 – Example outputs from the classification model

3.2 Detection Metrics and Results

For evaluating a detection model, the concept of true positives, false negatives, true negatives, and false positives is not the best way, especially since true negatives do not apply to detection models. A detection model aims to detect objects, not detect the lack of an object. Instead, the success of a prediction must be calculated by determining the Intersection over Union (IoU) and using those values to classify a success or failure. The intersection is the overlap between the model's prediction and the reference standard, and the union is the total area of the prediction and the reference standard, as seen in Figure 31. Detection model metrics rely on IoU, specifically Mean Average Precision (mAP) and the associated Mean Average Recall (mAR). Mean Average Precision takes the Average Precision at different IoU thresholds and then averages across those values. For context, the Average Precision is the average precision value across the precision-recall curve, where the precision is graphed against the recall. This is different from the Area Under Curve metric seen in the classification section, where the entire area under the curve is calculated instead of across the curve. For the thresholding for mAP, if the IoU threshold is 0.5, a prediction must have an IoU value of at least 0.5 or greater to be considered correct. Continuing from this, after determining the thresholds, the Average Precision is taken at each threshold. After all the Average Precisions are gathered at each threshold, they are averaged to produce the Mean Average Precision. Mean Average Recall is similar but is the average recall at all the IoU thresholds. To gather these metrics, the PyTorch MeanAveragePrecision class was used, which generates metrics in the style of the Common Objects in Context (COCO) detection challenge, as seen in Figure 32 (8).

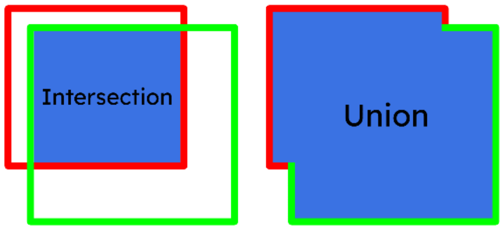
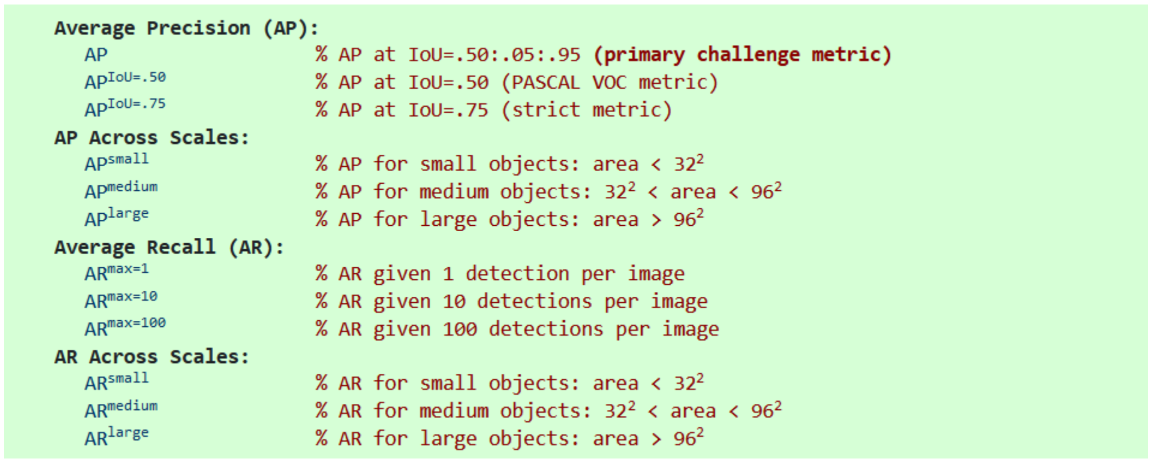


Figure 31 - Diagram of Intersection and Union



```
Average Precision (AP):
   AP                    % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^{IoU=.50}          % AP at IoU=.50 (PASCAL VOC metric)
   AP^{IoU=.75}          % AP at IoU=.75 (strict metric)
AP Across Scales:
   AP^{small}            % AP for small objects: area < 32^2
   AP^{medium}           % AP for medium objects: 32^2 < area < 96^2
   AP^{large}            % AP for large objects: area > 96^2
Average Recall (AR):
   AR^{max=1}            % AR given 1 detection per image
   AR^{max=10}           % AR given 10 detections per image
   AR^{max=100}          % AR given 100 detections per image
AR Across Scales:
   AR^{small}            % AR for small objects: area < 32^2
   AR^{medium}           % AR for medium objects: 32^2 < area < 96^2
   AR^{large}            % AR for large objects: area > 96^2
```

Figure 32 - COCO detection challenge metric definition

However, the PyTorch implementation also allows for individual class metrics, which is useful to see if a specific class is affecting the overall metrics but only shows the classes that existed in the test set. This means that three classes are not included in the metrics – bone anomalies (192 examples in the entire dataset), bone lesions (42 examples), and foreign bodies (8 examples). Soft tissue was not present in the test set (439 examples), but the model detected an instance of soft tissue; in this case, the mAP is -1.0. Tables 7 to 10 show the results for each object class and the relevant detection metric results, and Table 11 shows example results from the detection model (Figures 33, 34, 35) overlaid with the reference standard boxes.

mAP	**mAP@.50**	**mAP@.75**	**mAP – small**	**mAP – med**	**mAP – large**
0.4831	0.7933	0.4945	0.2556	0.4664	0.4916

Table 7 – Table containing overall Mean Average Precision values

mAR – 1	**mAR – 10**	**mAR – 100**	**mAR – small**	**mAR – med**	**mAR – large**
0.5372	0.5921	0.5921	0.2817	0.5509	0.6083

Table 8 – Table containing overall Mean Average Recall values

mAP – fracture	**mAP – metal**	**mAP – periosteal reaction**	**mAP – pronator sign**	**mAP – soft tissue**	**mAP – text**
0.4116	0.7843	0.2224	0.3255	-1.0	0.6717

Table 9 – Table containing Mean Average Precision values per class

mAR – fracture	**mAR – metal**	**mAR – periosteal reaction**	**mAR – pronator sign**	**mAR – soft tissue**	**mAR – text**
0.5068	0.85	0.3261	0.5667	-1.0	0.7110

Table 10 – Table containing Mean Average Recall values per class

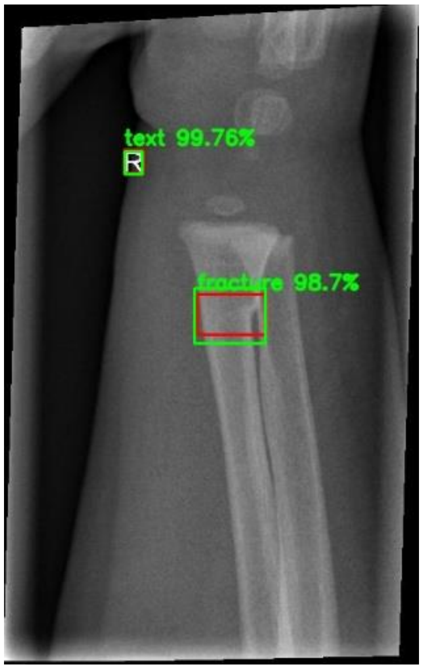
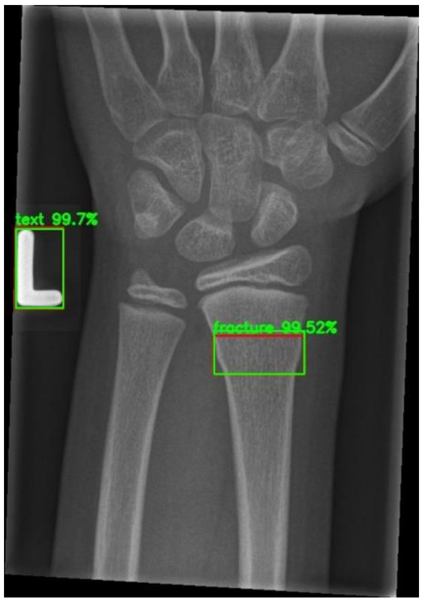
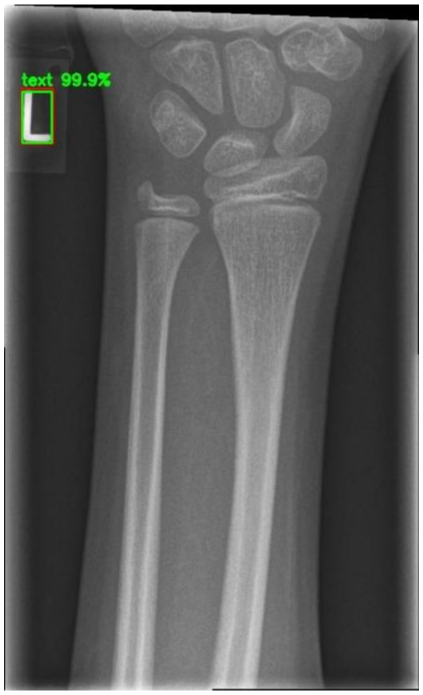
 *Figure 33 - A fractured X-ray with undeveloped carpal bones, a torus fracture, and text, with 98.7% fracture confidence and 99.76% text confidence*	 *Figure 34 - A fractured X-ray with unfused growth plates, a torus fracture, and text, with 99.52% fracture confidence and 99.75% text confidence*	 *Figure 35 - An unbroken X-ray with unfused growth plates and text, with a 99.9% text confidence*

Table 11 – Example outputs from the detection model, overlaid with the reference standard bounding boxes

3.3 Fail Cases

As seen in the metrics section, the model is not 100% accurate, so it is important to examine the failure cases for each model.

3.3.1 Classification Fail Cases

3.3.1.1 Classification False Positive Errors

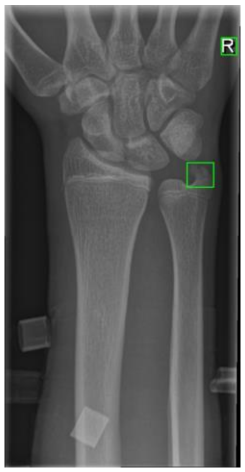
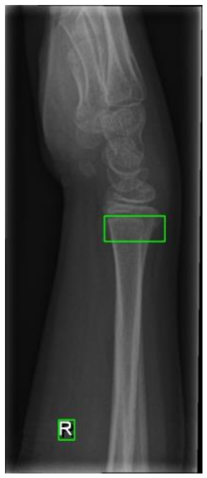
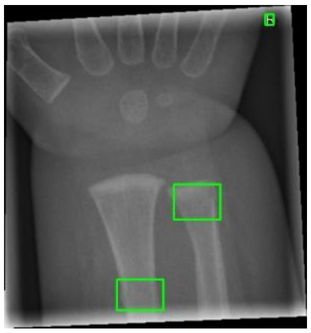
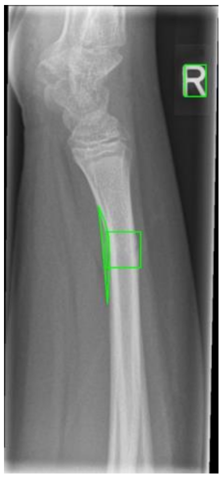
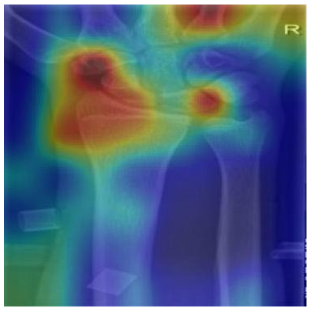
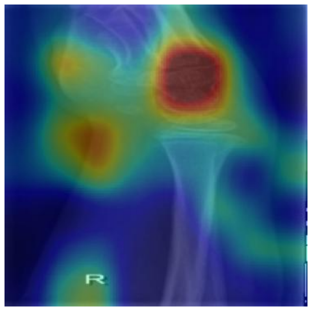
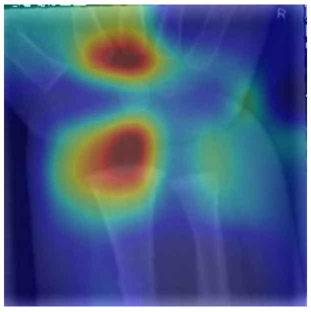
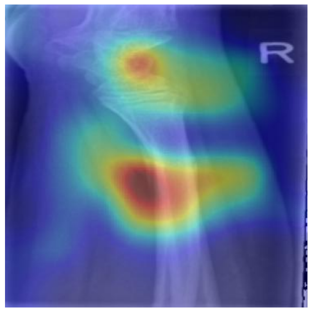
 *Figure 36 - An X-ray showing an avulsion of the ulna*	 *Figure 37 - An X-ray showing a torus fracture of the radius*	 *Figure 38 - An X-ray showing torus fractures of the radius and ulna*	 *Figure 39 - An X-ray showing a greenstick fracture of the radius and a periosteal reaction*
 *Figure 40 - Incorrect prediction of "Unsure", with 59.50% confidence of Not Fractured*	 *Figure 41 - Incorrect prediction of "More Likely Not Fractured" with confidence of 80.26%*	 *Figure 42 - Incorrect prediction of "Unsure" with 51.57% confidence of Not Fractured*	 *Figure 43 - Incorrect prediction of "More Likely Not Fractured" with confidence of 61.93%*

Table 13 – False positive failure cases for the classification model

For each of the false positive cases seen in Table 13 (Figures 36, 37, 38, 39, 40, 41, 42, 43), the model activates on discrepancies in the X-ray, namely the growth plates almost being fused in Figure 36, the lack of carpal bones in Figure 38, and the periosteal reaction in Figure 39. Despite this, it identifies the discrepancy but not the fractures present. In future research, these false positives could be assuaged by augmenting more images with underdeveloped carpal bones and avulsions, as these are rare classes in the dataset.

3.3.1.2 Classification False Negative Errors

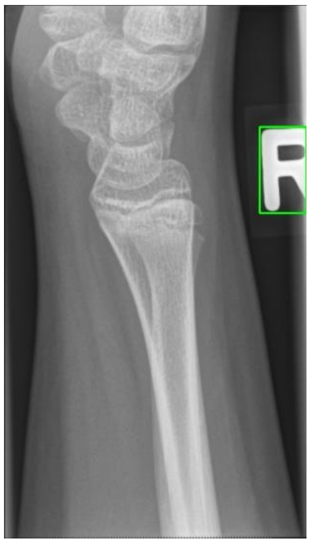
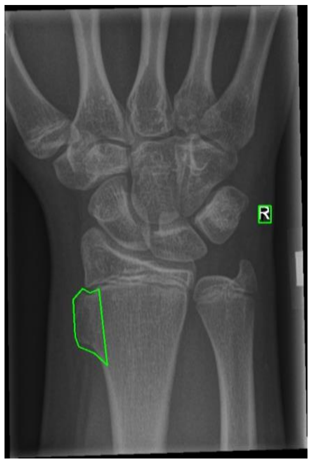
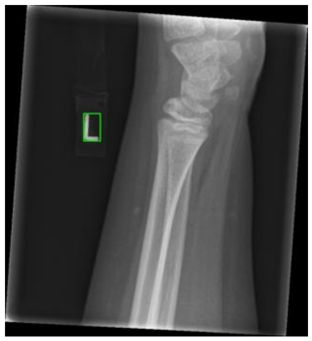
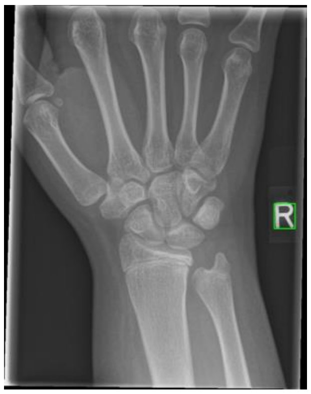
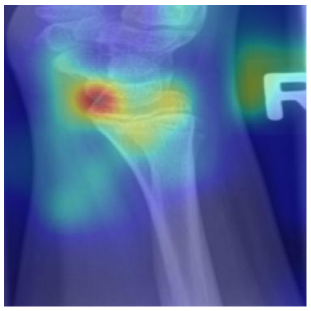
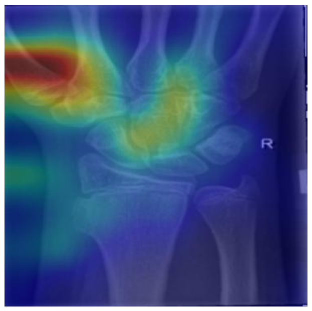
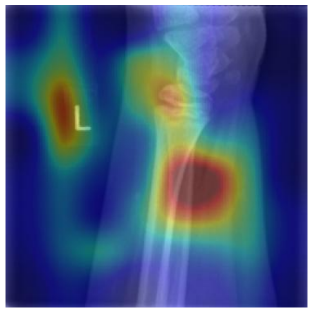
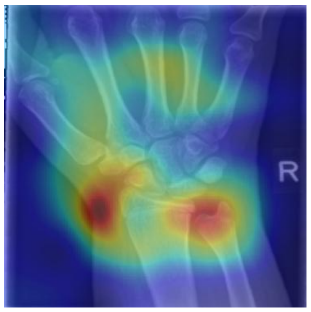
 *Figure 44 - An unbroken lateral X-ray, with only text present*	 *Figure 45 - An unbroken anteroposterior X-ray, with text and a bone lesion present*	 *Figure 46 - An unbroken lateral X-ray with only text present*	 *Figure 47 - An unbroken anteroposterior X-ray with only text present*
 *Figure 48 - Incorrect prediction of "More Likely Fractured" with a confidence of 69.60%*	 *Figure 49 - Incorrect prediction of "Fractured" with confidence of 89.69%*	 *Figure 50 - Incorrect prediction of "More Likely Fractured" with confidence of 71.52%*	 *Figure 51 - Incorrect prediction of "More Likely Fractured" with confidence of 62.94%*

Table 14 – False negative failure cases for the classification model

For the false negative failures, as shown in Table 14 (Figures 44, 45, 46, 47, 48, 49, 50, 51), the model activates on discrepancies such as the radius and ulna not being fully inline for a lateral view or the ulna's styloid process being more visible. These errors could be improved in future research by training two separate classification models, one for lateral views and another for anteroposterior views. This would allow each model to expect possible discrepancies for that view, such as the misaligned radius and ulna.

3.3.2 Detection Fail Cases

For clarity, the result images only displayed fractures, as the detection threshold was set to 0 to ensure it was a true failure case.

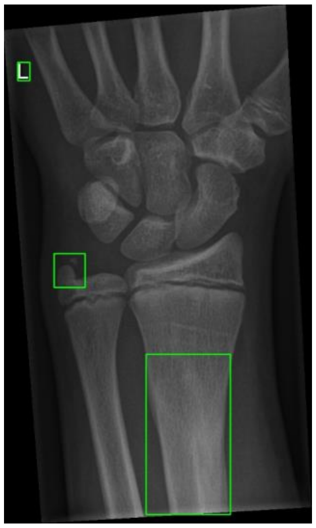
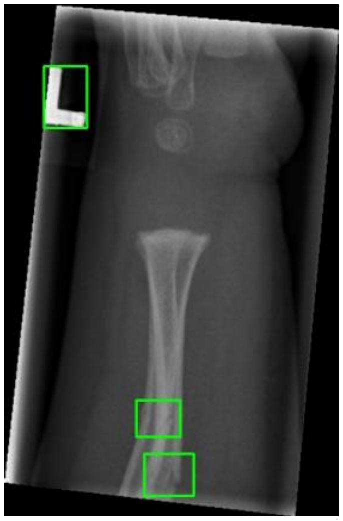
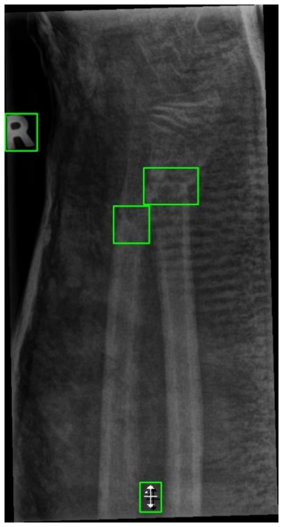
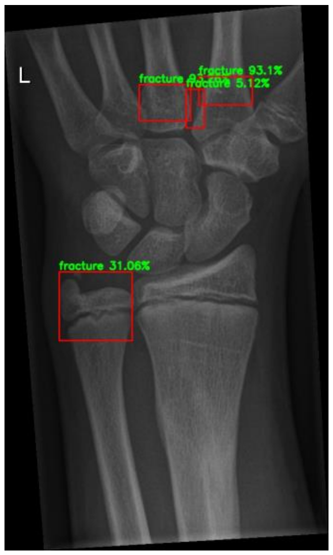
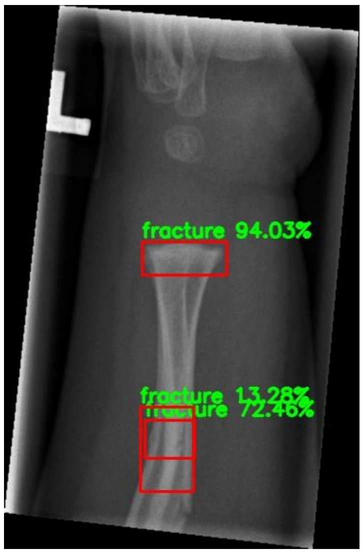
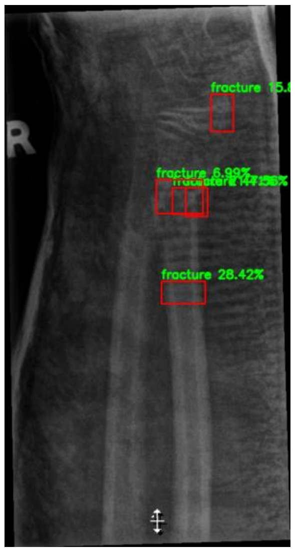
 *Figure 52 - An anteroposterior X-ray showing an avulsion of the styloid*	 *Figure 53 - A lateral X-ray showing greenstick fractures in both the radius and ulna*	 *Figure 54 - A lateral X-ray showing a simple complete fracture of the radius and a torus fracture of the ulna*
 *Figure 55 - The detection model's results, detecting the avulsion but encompassing the ulna's entire head while missing the damage to the radius and incorrectly detecting carpal fractures*	 *Figure 56 - The detection model's results, incorrectly detecting a fracture where the epiphysis has not developed, and missing the second greenstick fracture.*	 *Figure 57 - The detection model's results, incorrectly detecting a fracture near the styloid, as well as a second fracture of the radius, caused by the cast affecting the quality. It misses the ulna's torus fracture.*

Table 15 – Failures from the detection model, where it completely misses a fracture

For the detection fail cases, it was harder to determine examples, as often the model would correctly find a fracture, only to have a low confidence in it, and it to be filtered out by the detection threshold in the front end. The detection model was only trained on fractured images, and often, it would assume a fracture was present when given an unbroken X-ray. The examples in Table 15 (Figures 52, 53, 54, 55, 56, 57) demonstrate the case where the model completely missed a fracture. In Figure 52, the fracture is completely separate from the bone, and the radius's fracture has a large trauma, uncommonly seen. In Figure 53, the model correctly identifies part of the true fracture but misses the second fracture further down. This is plausibly due to the model rarely seeing diaphyseal fractures, and it would be out of range. In Figure 54, the X-ray of low quality and the addition of a cast obscures the ulna's torus fracture as well as generates a line across the radius that the model detected as a fracture.

3.4 Comparison to Others

With any research, it is important to evaluate the results against other academic papers discussing similar topics. The comparison has been broken into two sections, one discussing the classification results against similar classification tasks and the other discussing the detection results against comparable detection research.

3.4.1 Classification Model Comparison

In Lindsey *et al.*'s paper, they created an architecture based on U-Net and pre-trained the model on a variety of X-rays before specifying the training to exclusively wrist X-rays (9). Similar to FracTect's classification model, their custom model returns both a probability and a heatmap, although they do not use Grad-CAM and instead opt for their own implementation. Their study also compared their results to a group of medical professionals using the software, as well as a control group of unaided medical professionals. Ideally, future research on FracTect would include a comparison to a group of medical professionals to adequately measure whether the software improves overall diagnostic ability in a real-world scenario. Kim and McKinnon's paper uses the Inception V3 architecture to classify X-rays as fractured or not fractured (10). Interestingly, Kim and McKinnon excluded X-rays with a cast, X-rays where the growth plates were not fused, and X-rays where it specifically was not a distal radius/ulna fracture. This is the direct opposite of FracTect, where, since the GRAZPEDWRI-DX dataset was comprised of paediatric X-rays, the X-rays rarely had fused growth plates, and often, many of the bones were not present in the X-ray at all. X-rays with casts and with other types of fractures, such as scaphoid and other carpal bone fractures, were not removed from the dataset but were not balanced to match the total number of distal radius fractures. The primary conclusion of Kim and McKinnon's paper was that applying transfer learning, where the model is pre-trained on similar images, is a viable subject for future medical AI research. This paper influenced the decision to use the pre-trained model from PyTorch, but in future research, the model should be trained on X-ray–like images instead of the ImageNet dataset that the model is currently trained on. Tobler *et al.* split their dataset into subsets, each focusing on a different object present in the X-ray – e.g. metal and casts – and focused on classifying X-rays into different sub-categories, rather than just the binary classification task that's found in FracTect (11). Tobler *et al.* used the ResNet-18 architecture and aimed for multi-class classification, notably determining single fractures versus multiple fractures, which FracTect does not distinguish between, rather than grouping into one fractured class. Table 16 contains the results from other papers compared to FracTect's metrics; however, not every metric is presented in every paper.

	Lindsey *et al.*	**Kim and McKinnon**	**Tobler *et al.***	**FracTect**
AUC	0.967 and 0.975	0.954	0.98	0.932
Specificity	0.945	0.88	Not Given	0.9166
Sensitivity	0.939	0.9	Not Given	0.9468
Accuracy	Not Given	Not Given	0.94	0.9318

Table 16 – Table with comparisons between state-of-the-art papers and FracTect's classification model

3.4.2 Detection Model Comparison

In Guan *et al.*, they created a custom architecture that specifically targeted arm fractures and used the MURA dataset (12,13). Their paper was focused primarily on their architecture's design but also offered comparison metrics to other architectures, such as Faster R-CNN and Cascade R-CNN. Their paper offered insight into how a custom architecture would benefit computer-aided detection, especially when compared to the results of other architectures, specifically Faster R-CNN's 32.72% mAP@0.5. Yahalomi *et al.*'s paper is more applicable, as they specifically used a Faster R-CNN to identify distal radius fractures, similar to FracTect (14). They also tested their model's ability to perform classification, allowing for comparison against a medical professional's skill. In future research, testing the detection model's ability to classify X-rays would be beneficial and would simulate a real-world environment. Most notably, Yahalomi *et al.*'s initial dataset contained only 95 images, which was then augmented to 4,476 images. Considering their success, this is promising for balancing the dataset in the future for underrepresented classes. In FracTect's case, this augmentation would include images with casts or images that contain carpal bone fractures. Table 17 compares the mAP@0.5 of Guan *et al.* and Yahalomi *et al.*'s models against that of FracTect.

	Guan *et al.*	**Yahalomi *et al.***	**FracTect**
mAP@0.5	0.6204	0.866	0.7933

Table 17 – Table comparing the mAP@0.50 between the state-of-the-art papers and FracTect's detection model

FracTect's detection model underperforms when compared to Yahalomi *et al.*'s model; however, it is important to note that FracTect was trained as a multi-class detection, identifying up to nine classes, whereas Guan *et al.* and Yahalomi *et al.*'s models were trained exclusively to detect fractures. Future research would find it beneficial to get baseline metrics for multi-class X-ray detection models rather than focusing on the wrist/arm section of the body for comparison.

4 Discussion

FracTect's novelty is in its merits as a system, as similar literature develops their models in a vacuum. Both the classification and detection models perform on par with the current literature, but each model it was compared to was exclusively designed for that task. The comparative literature was also developed exclusively considering their models and lacked the overarching system necessary for medical professionals to interact with said models. FracTect fills this gap, providing a front end that can be improved with user feedback, as well as combining the classification and detection tasks into one interface. FracTect is a good step into creating a full diagnostic aid system rather than isolating parts of said system.

Overall, FracTect provides a solid tool for medical professionals to aid diagnostic abilities, but it isn't infallible. In its current state, FracTect should not be released into general use within the healthcare system for multiple reasons. Primarily, FracTect does not conform to the high levels of data privacy needed for use in a live medical environment. FracTect currently runs in a user's local environment, and while any identifying patient information is not stored, as it stands, FracTect cannot guarantee full data privacy. This would be a concern addressed with future development, though, as future development would make use of a hosting service and would add a user database for medical professionals to customise their workspace, as well as the possibility of viewing past results if desired. The GRAZPEDWRI-DX dataset's associated paper mentions DICOM images having "burned in" patient data, and special care would have to be taken to ensure all images are completely anonymised. At this stage of development, it would be ill-advised to release the software without verifying against patient privacy laws. Certain other quality-of-life issues would need to be fixed, such as results disappearing on a page reload and having to configure the system locally, but these would be improved quickly in comparison to ensuring data privacy. Future development may also find benefits in dividing the classification model into two separate models: one to classify anteroposterior X-rays and one to classify lateral X-rays. It may

also be beneficial to test FracTect using images from another dataset, ideally MURA, and ensure the quality is preserved.
Another point to consider is that although the metrics are good by general machine-learning model standards, they would need to be improved prior to widespread release, as medical software must conform to high accuracy to be used on actual patients since the penalty for failure directly affects human lives.
A further extension of FracTect would be adding functionality for other types of X-rays: despite wrist fractures being the second most common fractures, a live environment would require a diverse range of possible X-rays (1). Additionally, medical professionals may also require MRI scans to assess a possible fracture, which opens up further research, especially as MRI scans generate three-dimensional images rather than a two-dimensional X-ray.
This being said, the combination of a classification model and a detection model with an associated front end is a novel concept and would aid medical professionals, especially in coping with large workloads and confirming or rejecting initial assumptions. The results achieved are on par with the current research despite the time and processing power restraints. In future development, it is advisable to work with local resources when possible, as this would allow for larger datasets with a better balance between subclasses to be trained for longer. It is also advisable to test FracTect in a user test group comprised of medical professionals as well as people with less medical knowledge to compare the usefulness of the system between the two groups.

5 Conclusion

FracTect is a promising foray into medical AI usage and offers a prototype that can evolve with further work. Work on quality-of-life improvements and bringing the system into full compliance with medical regulations are still needed. Still, FracTect can make a material difference in patient care and improve the stress levels of medical professionals so long as the time is taken to ensure the security of patients' personal data. Future research would extend FracTect's system to external dataset testing, as well as having users with medical experience test the system. FracTect's research data can be accessed at the relevant GitHub repository (DOI: 10.5281/zenodo.14013673) (15).

6 Acknowledgements

I am extremely grateful to my supervisor for this project, Dr Sareh Rowlands, for her advice and support throughout the entire development of FracTect. Without her, FracTect would never have gotten off the ground in the first place.

7 Statements and Declarations

Ethical Considerations

No ethical approval was needed, as the dataset used gained their ethical approval for the data collection, and the dataset was fully anonymised. The dataset anonymised dataset can be seen in the respective paper (4).

Consent To Participate

Not Applicable.

Consent To Publication

Not Applicable.

Declaration of Conflicting Interest

The author declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding Statement

The author received no financial support for the research, authorship, and/or publication of this article.

Data Availability

The code is available at the relevant GitHub repository, https://github.com/jennischofield/fractect, and the GRAZPEDWRI-DX dataset is available with the relevant paper online (4). Any questions about the code can be directed to the corresponding author.

8 References

1. Jennison T, Brinsden M. Fracture admission trends in England over a ten-year period. Ann R Coll Surg Engl. 2019 Mar;101(3):208–14.
2. Shurlock T. NHS Resolution ED report Missed Fractures.pdf [Internet]. National Health Service; 2022 Mar [cited 2023 Nov 14] p. 30. (Clinical Negligence Claims in Emergency Departments in England). Report No.: 2. Available from: https://resolution.nhs.uk/wp-content/uploads/2022/03/2-NHS-Resolution-ED-report-Missed-Fractures.pdf
3. Hallas P, Ellingsen T. Errors in fracture diagnoses in the emergency department – characteristics of patients and diurnal variation. BMC Emerg Med. 2006 Dec;6(1):4.
4. Nagy E, Janisch M, Hržić F, Sorantin E, Tschauner S. A pediatric wrist trauma X-ray dataset (GRAZPEDWRI-DX) for machine learning. Sci Data. 2022 May 20;9(1):222.
5. Joshi D, Singh TP, Joshi AK. Deep learning-based localization and segmentation of wrist fractures on X-ray radiographs. Neural Comput Appl. 2022 Nov;34(21):19061–77.
6. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: Bengio Y, LeCun Y, editors. 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings [Internet]. 2015. Available from: http://arxiv.org/abs/1412.6980
7. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In: 2017 IEEE International Conference on Computer Vision (ICCV) [Internet]. 2017 [cited 2025 Apr 2]. p. 618–26. Available from: https://ieeexplore.ieee.org/document/8237336
8. COCO - Common Objects in Context [Internet]. [cited 2024 Apr 29]. Available from: https://cocodataset.org/#detection-eval
9. Lindsey R, Daluiski A, Chopra S, Lachapelle A, Mozer M, Sicular S, et al. Deep neural network improves fracture detection by clinicians. Proc Natl Acad Sci. 2018 Nov 6;115(45):11591–6.
10. Kim DH, MacKinnon T. Artificial intelligence in fracture detection: transfer learning from deep convolutional neural networks. Clin Radiol. 2018 May 1;73(5):439–45.
11. Tobler P, Cyriac J, Kovacs BK, Hofmann V, Sexauer R, Paciolla F, et al. AI-based detection and classification of distal radius fractures using low-effort data labeling: evaluation of applicability and effect of training set size. Eur Radiol. 2021 Sep;31(9):6816–24.
12. Guan B, Zhang G, Yao J, Wang X, Wang M. Arm fracture detection in X-rays based on improved deep convolutional neural network. Comput Electr Eng. 2020 Jan 1;81:106530.
13. Rajpurkar P, Irvin J, Bagul A, Ding D, Duan T, Mehta H, et al. MURA Dataset: Towards Radiologist-Level Abnormality Detection in Musculoskeletal Radiographs. In: Medical Imaging with Deep Learning [Internet]. 2018. Available from: https://openreview.net/forum?id=r1Q98pjiG
14. Yahalomi E, Chernofsky M, Werman M. Detection of Distal Radius Fractures Trained by a Small Set of X-Ray Images and Faster R-CNN. In: Arai K, Bhatia R, Kapoor S, editors. Intelligent Computing. Cham: Springer International Publishing; 2019. p. 971–81.
15. Schofield J. jennischofield/fractect: v1.0.0 [Internet]. Zenodo; 2024. Available from: https://zenodo.org/doi/10.5281/zenodo.14013673