Deep Learning Small Project Report

Suhas Sahu (1003370)
Sidharth Praveenkumar (1003647)

Data Preprocessing

Data Visualization

Upon gaining access to the dataset, the first thing we did was to examine the data, view the class distribution and deliberate on the possible methods to improve our model's performance.

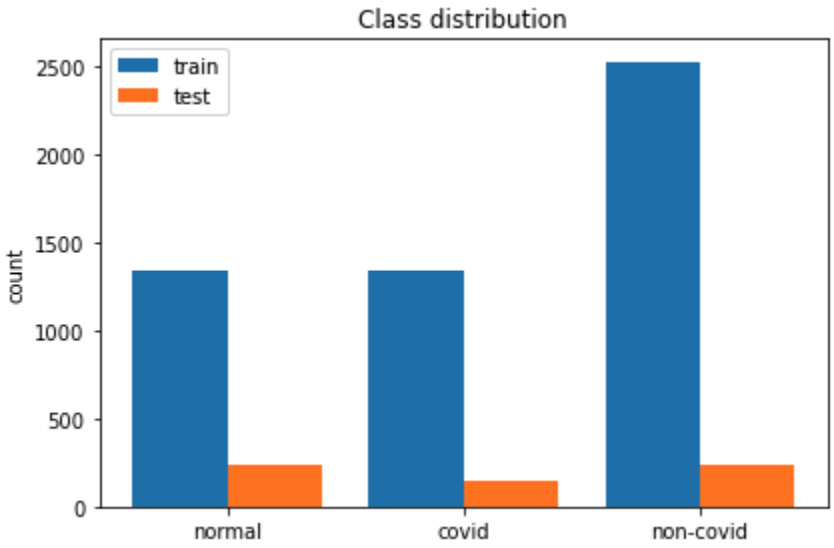

Image 1: Class Distribution of the Dataset

A quick plot of the distribution shows that there is indeed a discrepancy between the various classes. There are a total of 5216 images in the training dataset. A fair distribution among the classes would imply that each class has 1738 images. However, viewing the chart above proves that this is not the case. The data is heavily skewed towards that of non-covid, which has approximately double the samples of covid and non-covid. In the case of the test data set, however, the samples seem to be fairly evenly distributed, with 234, 138, 242 for 'normal', 'covid' and 'non-covid' respectively.

Upon conducting some research online, it seems that this is a fairly common problem when dealing with medical data, as there are much fewer cases of a particular anomaly or disease as opposed to normal samples. (Banerjee, n.d.). This typically leads to a poor classification of anomalies or disease classes as compared to normal samples. As such, it was pertinent for us to implement some of the tools available to us to overcome this.

The 2 methods we considered were undersampling and oversampling. Undersampling refers to a process in which *n_min* samples are chosen from the class with larger samples, where *n_min* refers to the number of samples in the smallest class. Undersampling typically works when we have adequate data for the smaller class. However, given our relatively small sample size, random undersampling would not work well (Blaugs et al., 2015). Oversampling on the other hand works by selecting *n_maj* samples from

the minority class to be combined with the majority class to form a balanced dataset, where *n_maj* is the number of samples in the majority class. Given our literature review on sampling techniques for biomedical data, we decided to proceed with oversampling. Oversampling was achieved by duplicating the images provided in the training set for both the 'normal' and 'covid' classes. This increased the number of samples in the 'normal' and 'covid' classes as shown in image 2 below.

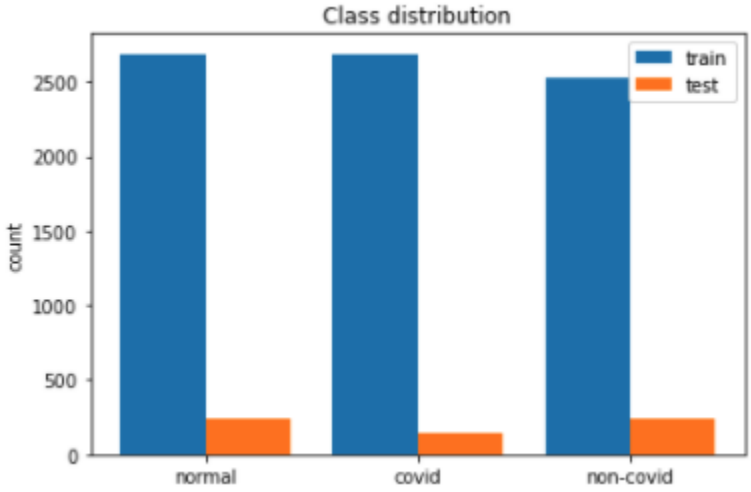

Image 2: Upsampled Class Distribution of Dataset

Creating a Dataset Object

Prior to developing the model, we had to create a general dataset object. The code makes reference to the demonstration notebook provided for the project. However, there are a few changes that have been made to facilitate the labelling and loading of data for a multiclass problem, instead of the binary problem used in the demonstration notebook.

The constructor has been changed to take into account features that were not previously considered. The first parameter is 'purpose', which rotates between 'train', 'val' and 'test', depending on the user's need. Next would be 'classifier', this would take an input of either 0, 1 or 2. They correspond to the classification for the dataset. If 0 is chosen the output for the labels will differentiate between normal and infected, 1 would separate normal, infected with no covid and infected with covid. The last option, 2, splits the data between infected with no covid and infected with covid. The third parameter is 'transform', which takes in boolean values, based on the user's desire to transform the dataset prior to training the model. The last parameter is 'upsample', which is a boolean as well. This parameter enables the user to decide if they would like to upsample the training datasets.

The 'describe', 'len' and 'show_img' methods are largely similar to the one provided. The 'open_img' method takes in the 'class_val' that can be set to either 'normal', 'infected', 'covid' or 'non_covid' instead of the binary classes as shown in the example. It retrieves the 'group_val' from the 'purpose' which will be input during the class initialisation. As before, this method would open an image in the desired class

and index. The '__getitem__' method returns the image and the associated label as a one-hot vector. Given the presence of 3 different models, 2 binary and 1 multiclass, it encodes the image based on the chosen model. For instance, if the model was chosen is multiclass and the index chosen is 'infected with no covid', the output will be [0, 1, 0].

Data Augmentation

After completing the initial pre-processing of our data which included creating the dataset object and upsampling the training data, the next step was to identify possible augmentations that might increase the diversity of the data and increase the model's ability in being able to distinguish between 'normal', 'infected with no covid' and 'infected with covid' classes.

Grayscale

The first augmentation that we considered applying was that of grayscale. Based on the theory covered in class and literature review, we realised that applying grayscale would have several benefits in helping the model distinguish the classes. The first is the complexity of the code. Finding edges based on both luminance and chrominance requires greater complexity than just luminance by itself. This additional colour information may thus warrant unnecessary complexity without aiding the model in distinguishing key features such as edges. Next would be the complexity of colour. While we are able to perceive colour seemingly easy, the same cannot be said for a computer. When working with coloured images it is important to ensure they follow the same lighting, camera colour calibration and other features that allow for colour to be effectively used to distinguish the classes. Lastly, using grayscale would lead to an increase in speed when training the model due to the reduction of channels it would have to consider.

Random Affine

The next augmentation that we utilised was that of RandomAffine from the Pytorch library. RandomAffine allows us to specify random translations of the image based on factors such as scaling, rotations, shearing and a few other factors. Given that the images were x-rayed, we found that randomly scaling was the affine that seemed to have the greatest impact on the model's performance, as compared to translation or shearing. While the location of the camera might be the same, each person could be standing at different positions from the camera, as such scaling the image will allow the model to generalise better.

Normalise the Pixels

Based on lectures and tutorials in school and online literature, neural networks typically process inputs that utilise small weight values, large values may slow down the learning process. As such, each pixel is normalised to ensure this does not occur.

Histogram Equalisation

One technique that seemed highly relevant for the project was that of histogram equalisation. Histogram equalisation is typically used for contrast enhancement in radiology (Dr Arridh Shahank et al., n.d.). This technique is often utilised when the sample is localised within a small band of values, such as that of an x-ray image of the lungs. The technique was utilised in several journal articles online and it works by distributing frequently occurring intensities more broadly. This increases the contrast perceived in an image. Applying this transformation proved beneficial for our model's performance as well.

Gaussian Blurring

The final augmentation that we felt would have a positive impact on our model was that of Gaussian Blurring. Gaussian Blurring is typically used to reduce noise in an image and detail. The main advantage of applying Gaussian blurring is that it enhances image structures at different scales and damps the high frequencies present in the image.

Results

Upon making the decision of which augmentations to apply to our dataset, we decided to conduct a test of its performance in relation to the original dataset of our model.

Base Model

Model without augmentation, preprocesing, LR scheduler and regularsation

```
In [23]: torch.cuda.empty_cache()
         !python ./train_binary.py --gpu --epochs 25 --batchsize 32 --upsample Fals
```

```
85.203
Epoch: 23/25 @ 2021-03-21 07:21:42.393114
 Training Loss: 0.088 -  Validation Loss: 1.172 -  Validation Accuracy:
72.846
Epoch: 24/25 @ 2021-03-21 07:21:51.743095
 Training Loss: 0.079 -  Validation Loss: 0.978 -  Validation Accuracy:
76.911
Epoch: 25/25 @ 2021-03-21 07:22:00.792993
 Training Loss: 0.081 -  Validation Loss: 1.126 -  Validation Accuracy:
71.382



Test Accuracy of model 1:
tensor([[ 53., 181.],
        [  0., 381.]])
Test set accuracy:  70.5691056910569 %
Figure(1200x600)
plot saved as model_classifier1_graph.png
```

Image 3: Performance of Base Model

Model with Augmentation and Preprocessing

Model with augmentation, preprocesing but wihtout LR scheduler and regularsation

```
In [26]: torch.cuda.empty_cache()
         !python ./train_binary.py --gpu --epochs 25 --batchsize 32 --upsample True
```

```
86.179
Epoch: 23/25 @ 2021-03-21 07:28:29.728280
 Training Loss: 0.090 -  Validation Loss: 0.596 -  Validation Accuracy:
80.650
Epoch: 24/25 @ 2021-03-21 07:28:38.721440
 Training Loss: 0.085 -  Validation Loss: 1.758 -  Validation Accuracy:
66.504
Epoch: 25/25 @ 2021-03-21 07:28:47.820205
 Training Loss: 0.080 -  Validation Loss: 0.615 -  Validation Accuracy:
84.553



Test Accuracy of model 1:
tensor([[150.,  84.],
        [ 11., 370.]])
Test set accuracy:  84.55284552845528 %
Figure(1200x600)
plot saved as model_classifier1_graph.png
```

Image 4: Performance of Model with Augmentation and Preprocessing

According to the results above, the ground truth was in tandem with the literature review. Both the augmentations and the preprocessing indeed helped improve the model's performance from 70.6% to 84.6%. This shows that the techniques applied were beneficial for x-ray images.

Model Architecture

The heart of this project is to build a neural network that is most reliable, robust and accurate for diagnosing COVID-19 from X-Ray Images. The performance of such a model depends largely on the type of architecture we use for the model, and two such architectures were proposed, namely the multi-class classifier and the binary classifiers. Instead of intuitively coming up with pros and cons for each model, we decided that it was best to start off by building a naive classifier for each of the architecture and then developing on the one that proved to be most reliable and accurate.

- Three-Class Classifier: This is the most intuitive approach to perform classification when we are dealing with more than two classes. In this architecture, each class will be trained on its corresponding X-ray images and then validated with its performance in classifying the classes correctly.

- Binary Classifier: This architecture can be regarded as an alternative to the first, and is also often referred to as the 'One vs Rest' architecture for multi-class classification. Contrary to the first approach, now we rely on two classifiers by grouping two of the classes together, namely 'covid' and 'non-covid' as 'infected'. The former classifier would then be a binary classifier that outputs the predicted class. This would be trained on the entire dataset, but the labels for both 'covid' and 'non-covid' classes would be the same, as they are regarded to belong to the same class in this context. The latter classifier would be a binary classifier that takes infected images as inputs and outputs the prediction belonging to 'covid' or 'non-covid'. This classifier would be trained only in two classes - i.e. we don't require the 'normal' dataset anymore, only the other two classes.

Upon training and evaluation of both types of architecture, we observed that the overall performance of the two binary classifiers was better than the multi-class classifier. Our approach for this project too is fixated on the binary classifier architecture. It was intuitive to understand why this architecture outperforms the multi-class, the primary advantage being the ability to redefine our network architecture for each of the classifier, something which we cannot perform for the multi-class classifier. Since this is a cascading of two binary classifiers, we have the potential to customize our classifier (i.e deeper or shallower networks depending on the feature differences among the classes). The obvious disadvantage to this type of architecture unfortunately stems from its advantage too, being that we have to develop two classifiers, which increases development and computational time drastically.

Network Architecture

The solution that we aim to construct comprises a dataset of X-Ray images, making this an image classification task at its heart. The network architecture is primarily of Convolutional layers. The corresponding parameters and functions are discussed below.

Layers and Parameters

For our first classifier, we use 4 convolution blocks, each with a kernel size of 3, stride and padding 1. Each convolution block embeds a max-pooling layer of size 2 - stride 2, along with a ReLu Activation function(this goes in hand with the Kaiming initialization we use for our model parameters, which is discussed on the following page). The output of this then passed into a batch normalization layer to ensure that the output to the next block is standardized and not extensively low or high. At the very end, we have employed two fully connected linear layers, which is essential because the depth of these FC layers had a huge impact on our performance (1 was too little and 3 made the training slower). For our second classifier, we employed the same architecture parameters, but we increased the depth of the convolution blocks from 4 blocks to 6 blocks and added one more Fully Connected Layer. This increase in depth of network was essential because the features to differentiate covid x rays from non-covid x rays were more complex features within the lung images, as compared to distinguishing normal lung x rays from pneumonia infected lung x rays.

Mini Batch Size - 32

We experimented with multiple mini-batch sizes, ranging from 16 to 256. The optimal mini-batch size for our architecture proved to be 32. The difference in performance between large and small batch size is very noticeable, especially with regards to the test accuracy. We observed that training with a large batch size (more than 128) usually led to a faster reduction in train loss but poor performance on the test set. On the other hand, too small of batch size (like 4 or 8) would increase training time largely. We decided on choosing the optimal batch size that strikes a balance between training time and generalization, which upon experimenting turned out to be 32.

Loss Function - Binary Cross-Entropy

As far as loss functions are concerned, we have a plethora of options to choose from for this purpose. However, we determined that the most suitable loss function for this type of binary classification would be Binary Cross Entropy.

Optimizer - Adam

While experimenting with Optimizers (SGD, RMSProp and Adam), we observed more consistent results with Adam than the other alternatives. A possible explanation for this can be attributed to an extremely

low learning rate, which in turn would have caused fewer oscillations around the local minima. Another reason for choosing Adam over SGD is that it's more tunable than SGD, and thus can be tweaked to converge at much faster rates. Scientific literature suggests that RMSProp tends to be more effective while dealing with sparsely connected networks, or RNNs, both of which are contrary to our architecture (CNNs are densely connected). We chose a learning rate of 0.001

Parameter Initialization - Kaiming Normal

Random weight initialization can potentially lead to either the vanishing gradient problem if the gradients are too small and tending to zero, or the gradients could explode and might compromise the model training. For this matter, we chose to initialize our Convolutional and FC layers using the Kaiming normal function. This type of initialization heuristic is dependent and most effective when the layer's activation function uses ReLu. It prevents the inner nodes from getting a zero output (which would be the case if we randomly initialize and then use ReLU because probability suggests that half of the nodes would then output a zero value). However, it is important to state that even without the kaiming normal initialization, our model was still training considerably well, and applying this initialization only added a slight amount of performance to it.

Learning Rate Scheduler - StepLR

One of the tools employed to improve model training that was taught in class was that of the learning rate scheduler. Learning rate schedulers seek to adjust the learning rate during training according to a predefined schedule. This is beneficial as not applying such a scheduler may lead to slow convergence of the model. Furthermore, the learning rate scheduler aids in preventing oscillations while training the model. This ensures that we do not go back and forth a local minima. As such, we employed a learning rate scheduler(with the RMSprop optimizer) with a step size of 13(performed best while experimenting) and a gamma value of 0.9.

Weight Decay/Regularization

An additional technique that we decided to implement for our model was that of weight decay. Given the relatively few sample images that we had, overfitting may be a problem. As such, it is important for us to implement techniques that will aid in generalising the model. Weight decay works by adding an extra term to the objective function, this, in turn, ensures that the model does not overfit the training data. The weight decay that we chose to use was 1e-4.

Model with Augmentation and Preprocessing

Model with augmentation, preprocesing but wihtout LR scheduler and regularsation

```
In [26]: torch.cuda.empty_cache()
         !python ./train_binary.py --gpu --epochs 25 --batchsize 32 --upsample True
Epoch: 23/25 @ 2021-03-21 07:31:20.055433
 Training Loss: 0.394 -  Validation Loss: 0.306 -  Validation Accuracy:
88.976
Epoch: 24/25 @ 2021-03-21 07:31:26.684504
 Training Loss: 0.388 -  Validation Loss: 0.318 -  Validation Accuracy:
87.402
Epoch: 25/25 @ 2021-03-21 07:31:33.481616
 Training Loss: 0.378 -  Validation Loss: 0.298 -  Validation Accuracy:
88.976



Test Accuracy of model 2:
tensor([[111.,  28.],
        [ 29., 213.]])
Test set accuracy:  85.03937007874016 %
Figure(1200x600)
plot saved as model_classifier2_graph.png
```

Image 5: Model without LR scheduler and regularisation

Model with LR Scheduler, Regularsiation, Augmentation and Preprocessing

Model with augmentation, preprocesing but wihtout LR scheduler and regularsation

```
In [29]: torch.cuda.empty_cache()
         !python ./train_binary.py --gpu --epochs 25 --batchsize 32 --upsample True
Epoch: 23/25 @ 2021-03-21 07:38:07.212427
 Training Loss: 0.413 -  Validation Loss: 0.353 -  Validation Accuracy:
85.039
Epoch: 24/25 @ 2021-03-21 07:38:13.831534
 Training Loss: 0.396 -  Validation Loss: 0.315 -  Validation Accuracy:
88.189
Epoch: 25/25 @ 2021-03-21 07:38:20.340793
 Training Loss: 0.395 -  Validation Loss: 0.358 -  Validation Accuracy:
84.777



Test Accuracy of model 2:
tensor([[111.,  28.],
        [ 23., 219.]])
Test set accuracy:  86.61417322834646 %
Figure(1200x600)
plot saved as model_classifier2_graph.png
```

Image 6: Model with LR scheduler and regularisation

Applying a learning rate scheduler and weight decay did indeed improve the performance of the model, in line with the literature review. As can be seen from the images above, the performance did indeed increase by around 1.6%.

Performance and Results

To ensure we covered all grounds in determining the better model, we proceeded to compare the performance of both the three-class classifier and binary classifier. From our understanding of neural networks, however, we believed that the multiclass classifier would take additional time in training the model and may not perform as well. This is because it is typically easier for a model to be trained to differentiate between 'normal' cases and 'infected' cases as opposed to differentiating between 'covid' and 'non-covid' status. However as seen from the results below, the performance of the binary classifiers was deemed superior to that of the multiclass classifier. Furthermore, the binary classifiers had a shorter training time as compared to the multiclass classifier and provided greater flexibility for developing models to help differentiate each class.

Multiclass Classifier

Model (Three Class Classifier)	Test Accuracy
Without augmentation & preprocessing	78%
With augmentation & preprocessing	79%
Without LR Scheduler & Regularization	79%
With LR Scheduler & Regularization	82%

Binary Classifiers

The first classifier (to classify 'normal' and 'infected' images) produced an overall accuracy of 71% on the raw dataset, without using any data augmentation or preprocessing. After applying the preprocessing steps and transformations on the images (as discussed in the earlier subsections), the performance increased to 85%. Further results obtained with the addition of weight decay/regularization and learning rate scheduler can be found in the table below.

Model (Classifier 1 - normal/infected)	Test Accuracy
Without augmentation & preprocessing	71%
With augmentation & preprocessing	85%

Without LR Scheduler & Regularization	85%
With LR Scheduler & Regularization	83%

Model (Classifier 2 -covid/non-covid)	**Test Accuracy**
Without augmentation & preprocessing	86%
With augmentation & preprocessing	85%
Without LR Scheduler & Regularization	85%
With LR Scheduler & Regularization	87%

As rightly posed in the problem statement, the overall accuracy is not always the quintessential characteristic for defining the performance of a neural network. Especially, in this case, we need to be extremely careful while using a metric to define the entire performance of a model, largely because of the industry this problem can be attributed to, and how a potential patient's treatment could be based on the result. With this in mind, the primary metric should be that the model's results have the lowest False Positive and False Negative Rate (because we obviously would not want a patient with Covid-19 to be predicted as normal, and vice versa). The secondary metric (although counterintuitive) which is also the objective of this model, is to have a high True Positive and True Negative Rate. The confusion matrix of the binary classifier is, therefore, as follows:

```
Test Accuracy of model 1:
tensor([[132., 102.],
        [  4., 377.]])
Test set accuracy:  82.76422764227642 %
```

Image 7: Binary classifier 1 confusion matrix

```
Test Accuracy of model 2:
tensor([[111.,  28.],
        [ 23., 219.]])
Test set accuracy:  86.61417322834646 %
```

Image 8: Binary classifier 2 confusion matrix

To understand the performance of this classifier, we must elucidate on the confusion matrix above. The diagonal values are the true values, meaning, covid rightly classified as covid (111) and non-covid rightly

classified as non-covid (219). The off-diagonal values are the false positives and false negatives - 23 covid were wrongly classified as non-covid and 28 vice versa. Although this can be considered as a fair accuracy to obtain, there is still room for improvement in terms of increasing the true classification rate. LIkely, one of the main points of improvement would be to try to get inspiration from profound image classification architecture, especially ones that are extensively used in the biomedical industry. An example of this is COVID-NET which was recently tailored just for this purpose - classification of covid. Other areas of improvement would be to experiment more with the hyperparameters and optimizers along with the associated scientific literature for them.

Comparison with Doctor's Analysis

Based on an initial study, we realised that most doctors utilise the same features as the machine does in identifying certain conditions. An example is shown in image 9 below which details how to read an x-ray. As seen from the description for the picture, they utilise brightness and other similar features to identify gas, water and mineral density. This shows the reason why machine learning is useful in the field of radiology as the interpretations are done in a similar manner.

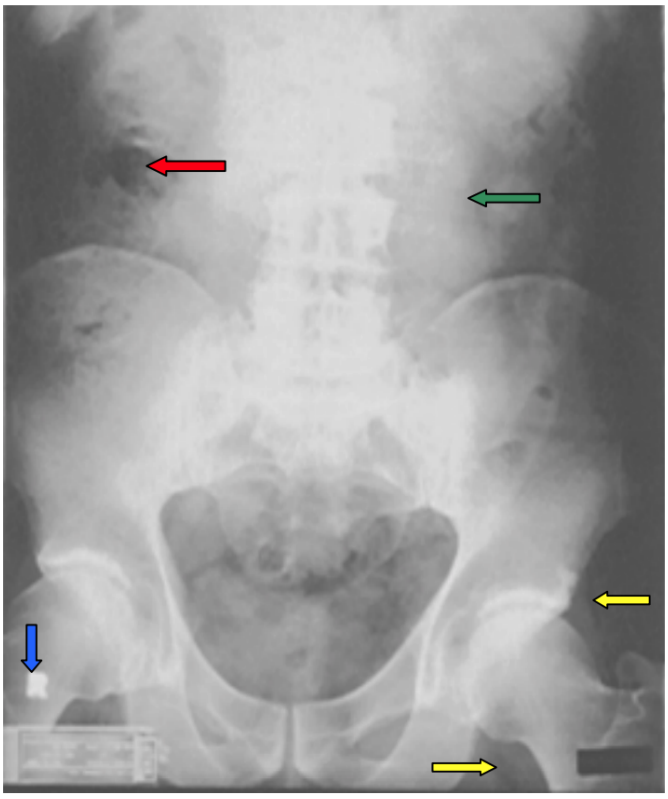

Figure #1(left). **The red arrow points to the black density of gas seen in the right side of the colon. The yellow arrows indicate the slightly lighter density (than gas) of fat in the left hip joint capsule. The blue arrow shows the bright density of metal (mineral) in the "R" of the film marker. Mineral density, not quite as bright as the heavy metal marker, is also noted throughout the bones of the skeleton. The green arrow indicates the water density of the left psoas muscle.**

Image 9: How to read x-rays from a doctor handbook (Hook, n.d.)

1. Turn off extraneous light. Lights overhead or empty adjacent lighted view boxes compromise what can be seen on the radiograph.

2. Use a magnifying glass, especially on small parts such as hands and feet. Magnification is also useful to see fine line detail in lungs and in other areas. The technologist can also magnify the film for you, e.g. one frequently used study is the magnified view of the carpal navicular.

3. Have a hot light available to see overexposed areas.

4. Have a darkroom technologist make light copies of overexposed films. Most darkrooms have that capability and it saves the patient another exposure. However never accept a technically unsatisfactory film in the fear of exposing the patient to too much radiation. To put it in perspective, a single view of the chest exposes the patient to about the same amount of radiation he or she would get by flying from Denver to San Francisco in an airliner.

5. Use rulers and calipers until your eye gets used to normal relationships. For example, in the chest, the heart should be about half the size of the width of the rib cage (C-T ratio).

Image 10: Measures to factor in when reading an x-ray (Hook, n.d.)

The precautions above highlight some of the advantages that radiologists have, mainly the advantage of being able to ensure their surroundings are consistent when interpreting the images. Also, they would be able to utilise external tools such as rulers and callipers on a case by case basis.

Ultimately, given that a model's accuracy tends to fluctuate, it would be best to utilise both doctor's analysis and machine in tandem.

References

1. Banerjee, S. (n.d.). Deep learning for analysis of imbalanced medical image datasets. Retrieved March 20, 2021, from https://devmesh.intel.com/projects/deep-learning-for-analysis-of-imbalanced-medical-image-datasets
2. Blagus, R., & Lusa, L. (2015). Joint use of over- and under-sampling techniques and cross-validation for the development and assessment of prediction models. *BMC Bioinformatics, 16*(1). doi:10.1186/s12859-015-0784-9
3. HOOK, W. F., MD. (n.d.). X-RAY FILM READING MADE EASYWILLIAM F. HOOK, MD. Retrieved from https://med.und.edu/radiology/_files/docs/xray-film-reading-made-easy.pdf
4. Shashank, D. (n.d.). Histogram equalization: Radiology reference article. Retrieved March 21, 2021, from https://radiopaedia.org/articles/histogram-equalisation