Machine Learning Engineer NanoDegree

Project Report

Name: Rishabh Sobti
Date: 10 March, 2018

I. DEFINITION

Project Overview

This project aims at inclusion of machine learning technology into the field of medical diagnosis. One very important part of lung and thorax assessment is chest X-Rays. The proper study of chest X-Rays can help doctors diagnose several diseases like effusion, infiltration, masses or nodules in the lungs, etc. As beneficial as chest X-rays are, they are also challenging to read and infer properly. A lot of work is already going on, in the field of computer-aided detection and diagnosis, but achievement of a satisfactory level of accuracy in such a delicate field is a dream yet to be achieved. A promising example in the field is using CNNs for detection of Tuberculosis (source: CNN for TB).

With the increasing amount of chest X-ray data openly available at our disposal, deep learning techniques can be very useful to study trends and aid doctors in reading chest X-rays. I come from a country where 5.2 million medical errors are happening annually, most of which are due to wrong diagnosis, so, I feel motivated to work in the given field. It should be noted, that ideology is not to eliminate the involvement of doctors, rather to help them make the right diagnosis.

The dataset hosted by the National Institutes of Health – Clinical Center (official website - https://clinicalcenter.nih.gov), is the most suited in order to achieve the desired goal. It is a collection of 112120 chest X-ray scans, from 30805 unique patients, each of 1024X1024 resolution, which are labeled into 15 classes. The dataset is openly available for download and use at "https://nihcc.app.box.com/v/ChestXray-NIHCC/folder/36938765345".

But, due to lack of computing capacity at my disposal and the time limit on this project, I have used a drastically and randomly reduced version of this complete dataset, which is contributed at and reviewed by kaggle.com. The link for the sampled dataset that I have used is https://www.kaggle.com/nih-chest-xrays/sample . The dataset can be obtained using this link, after going through a free signup. To make the downloading process even easier, I've uploaded the same dataset at this google drive link – https://drive.google.com/open?id=1IuCB5Etj-yOVrVtvM6Y8rybJKn4LCX3d . The labels on these images are provided in a separate .csv file that accompanies the dataset.

Problem Statement

The goal is to create a system which can classify input X-ray images into the 15 classes (14 diseases and one class for "no findings"), the details of which are discussed in the next section of this report. The tasks that are needed to be performed are:

1. Downloading and preprocessing the needed dataset, possibly reducing the resolution of images.
2. Designing a convolution neural network, which can train on the given data, and give better results than the benchmark model (vanilla neural network).
3. Making the model optimal, by adjusting parameters, that leads to a higher accuracy.

In logical terms, it is an image classification problem, and since the dataset, which will be used is labeled, it is both quantifiable and measurable, with help of metrics like F-score.

Evaluation Metrics

Since, we need to make a model which correctly predicts the disease associated with the chest X-ray, and the dataset we have is imbalanced, we need to consider F-score for the model, which gives a fair idea about the model if the dataset is sparse. The F-score can be defined as the harmonic mean of two other metrics, which are recall and precision.
Mathematically, for each class, we can calculate recall and precision as:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

Where TP: True positive or the samples of this class which are predicted correctly

FN: False negative or the samples which belong to this particular class, but predicted incorrectly

FP: False Positive or the samples which do not belong to this class, but are predicted to be in this class

And, for each class, F-1 score or simply, F-score is:

$$F - score = 2 * \frac{1}{\frac{1}{Recall} + \frac{1}{Precision}}$$

The F-score of all the classes are then averaged in a weighted fashion, to get F-score of the model.
In other terms, the F-score can be written as:

$$F - score = 2 * \frac{TP}{2 * TP + FP + FN}$$

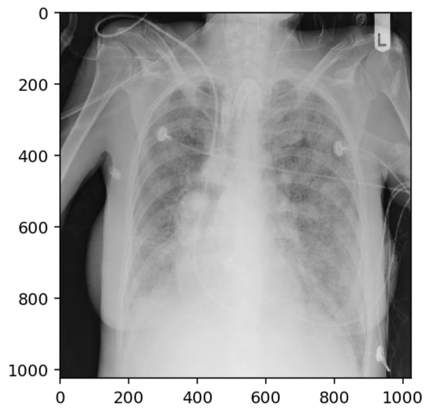
II. ANALYSIS

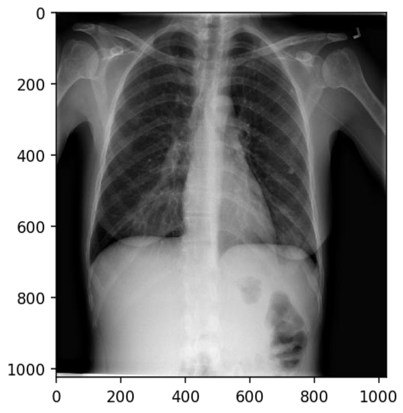
Dataset

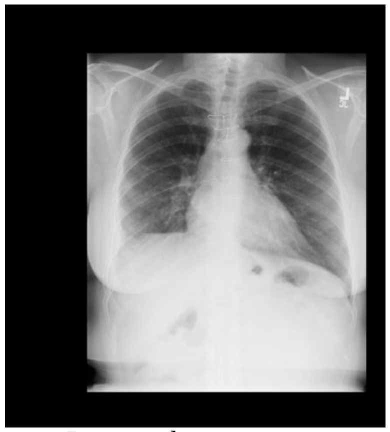
The link for the sampled dataset that we would be using is https://www.kaggle.com/nih-chest-xrays/sample . The dataset can be obtained using this link, after going through a free signup. To make the downloading process even easier, we've uploaded the same dataset at this google drive link – https://drive.google.com/open?id=1IuCB5Etj-yOVrVtvM6Y8rybJKn4LCX3d . The labels on these images are provided in a separate .csv file that accompanies the dataset. The 15 disease labels used in this dataset, as mined using Natural Language Processing on the original medical reports are:

1. Hernia - 13 images
2. Pneumonia - 62 images
3. Fibrosis - 84 images
4. Edema - 118 images
5. Emphysema - 127 images
6. Cardiomegaly - 141 images
7. Pleural_Thickening - 176 images
8. Consolidation - 226 images
9. Pneumothorax - 271 images
10. Mass - 284 images
11. Nodule - 313 images
12. Atelectasis - 508 images
13. Effusion - 644 images
14. Infiltration - 967 images
15. No Finding - 3044 images

The size of this reduced dataset is ~2GB, in comparison to 45.6GB dataset originally there, on the NIH-CC. It contains a total of 5606 images of the resolution 1024X1024. The resolution of images will be reduced to 256X256 before starting training on the CNN. Following are some samples from the dataset we have used:





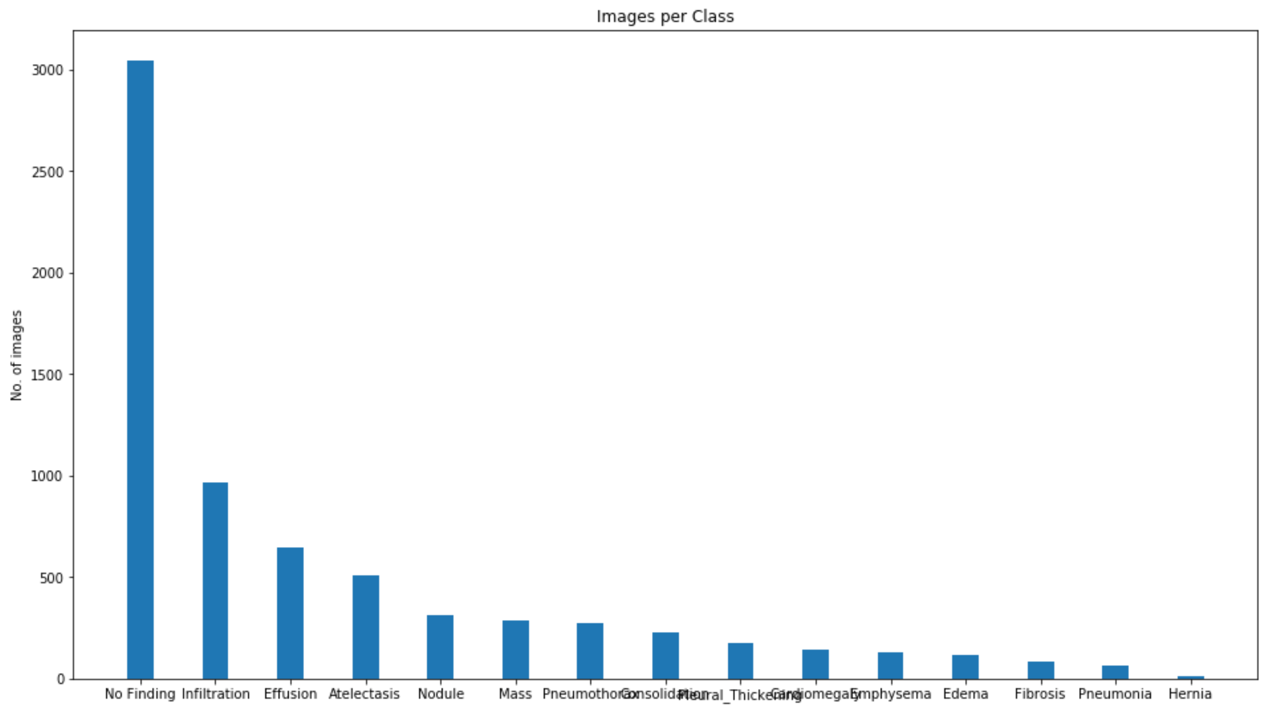


Fibrosis and Infiltration Nodule Pneumothorax

As it is clear from the number of images belonging to each class, the dataset we are using is **highly imbalanced**. This fact has to be kept in mind, while choosing the right metric and training loss function. Moreover, this problem is a **multi-output problem**, as clear from the first sample given above, that means, that a patient can have more than one disease at a time, and our system should be able to spot them all accurately.

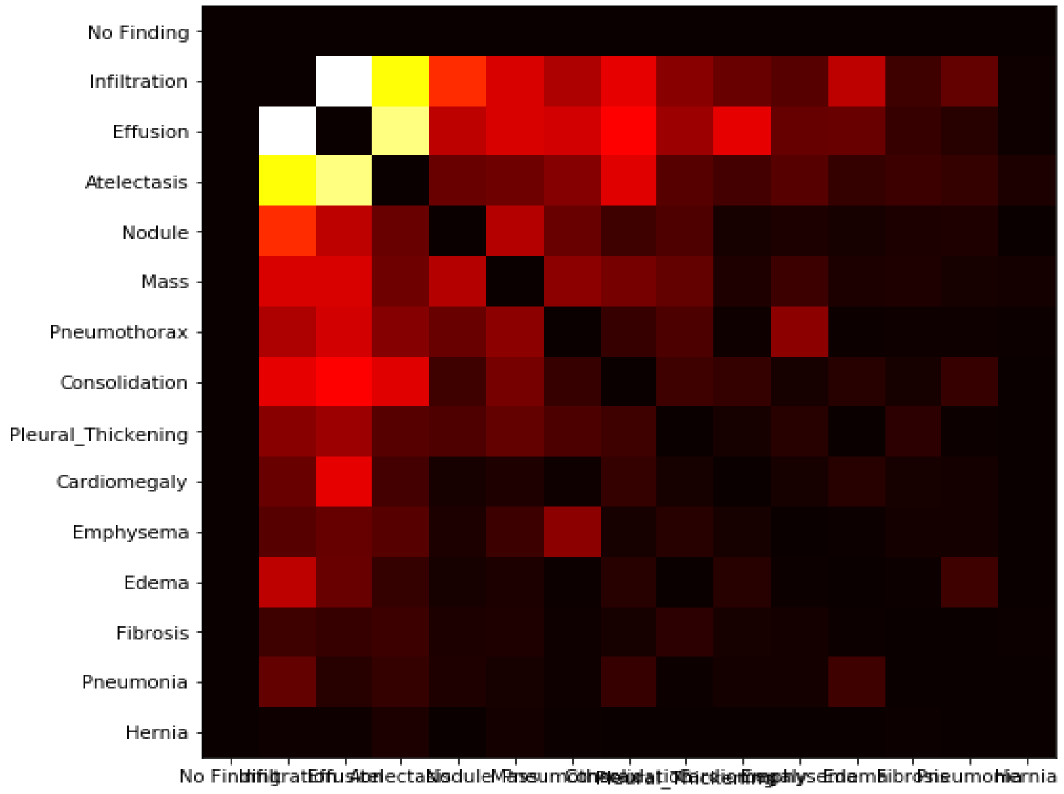
Exploratory Visualization

The following plot shows a bar graph of how the sample dataset is divided into the different classes.



The given visualization clearly helps us to explore that the skewed nature of this bar plot indicates that the dataset is imbalanced, and special care of this fact has to be taken, while deciding on algorithms and techniques to use. Moreover, it also shows that there are too few images for diseases at the far right of the plot, i.e., Fibrosis, Hernia, and Pneumonia. So, we should expect a lower F-score especially for these classes.

The given heat map, attempts to catch the co-relation among different classes:



This heat map helps us visualize the fact that there is a high amount of co-existence between Infiltration, Effusion and Atelectasis. Though, this information can be used in the refinement of the model, yet, in the scope of this project, it will not be taken into account.

Algorithms and Techniques

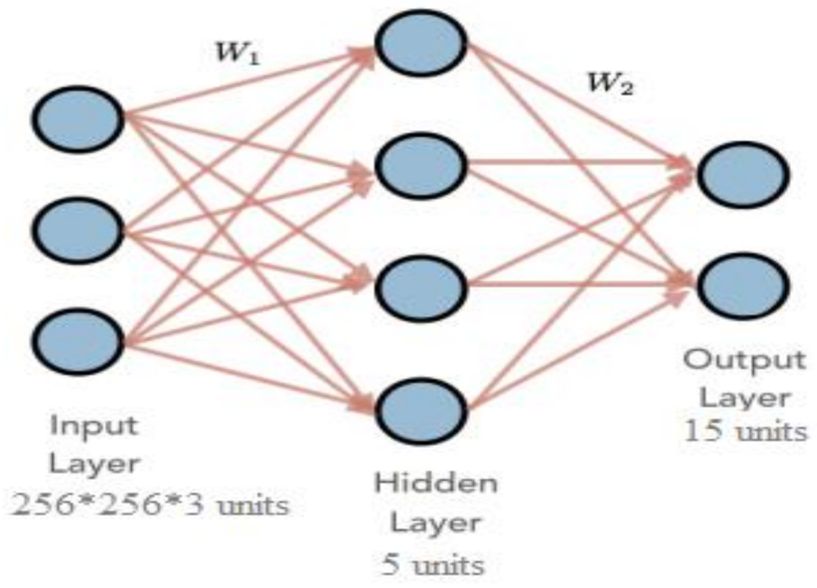
Convolution Neural Network (CNN) will be used to model and solve the given problem. This state of the art neural network is the most efficient algorithm for classifying images and natural language texts. The given algorithm will proceed in the following way:

- **Data Preprocessing**: The features will first be reduced to a 256X256X3 matrix, which is a low resolution RGB version of the original chest X-ray image. Then the features will be normalized and the corresponding labels from the .csv file will be one-hot encoded. The normalization helps to uniformly spread the feature in a restricted range. This preprocessed data will then be divided into training, validation and testing datasets and saved to the disk for future use.

- **Network Design:** The network is built from multiple convolution layers, max-pooling layers, and fully connected layers. The exact implementation of these layers is provided in the next section. Here we will look at the characteristics and need for these layers:
    - Convolution layers: These layers are key for any CNN. These layers are generated by moving fixed size two dimensional filters over each component of the given image (the RGB components). This leads to generation of a new layer with 'k' width where 'k' is number of filters. This layer helps in recognition of similarities in images, which might lead to the basis of their classification.
    - Max-pooling layers: The most important role of max-pooling layers is to reduce the dimensions of image without altering the width 'k'. The max-pool filter picks the maximum feature value from each section.
    - Fully Connected layers: These layers are added towards the end of the network, and are similar o the fully connected layers of a simple neural network. They are used to accurately map the different similarities together, which were picked up by the convolution layers.
- **Training:** As we have seen that the dataset is highly imbalanced, hence the loss function which needs to be minimized and the optimization algorithm have to be chosen wisely. Moreover, we should not use softmax activation in the final output layer, because multiple outcomes are possible, hence sigmoid will work better, which keeps all the outputs independent from each other. So, tensorflow's weighted_cross_entropy_with_logits() seems to be the best fit for our data and requirements. It is similar to sigmoid cross entropy with the only difference that it gives a weight factor to the positive labels. The optimization algorithm that we will be using is Adam optimizer, which is well equipped to adjust learning rates dynamically. The training length (epochs) and batch size will be kept at a value of 5 and 150 respectively, because of the limiting factor of the resources at disposal.

Benchmark Model

The model that I have used for this project is a simple "Vanilla Neural Network" trained on the same preprocessed dataset. Some characteristics and results of the benchmark model are as follows:



- As clear from the representation above, the vanilla neural network is a basic neural network with only one layer, which has been used as a benchmark.
- The image is first flattened and then all the inputs, independent of each other are fed to this neural network.
- The loss function used is weighted sigmoid cross entropy and optimizer is Adam's optimizer.
- **The neural network was run for 10 epochs, and the results converged to an F-score value, on the validation set, of 0.369 and the loss, on training set, of 0.95**
- These values will be considered as base values while refining the parameters of the CNN that we have constructed.

III. METHODOLOGY

Data Preprocessing

The preprocessing of the given data is important, so that the images and the text data can be converted to a format, which can be fed directly into the CNN we have constructed. The following steps are carried out in the preprocessing step:

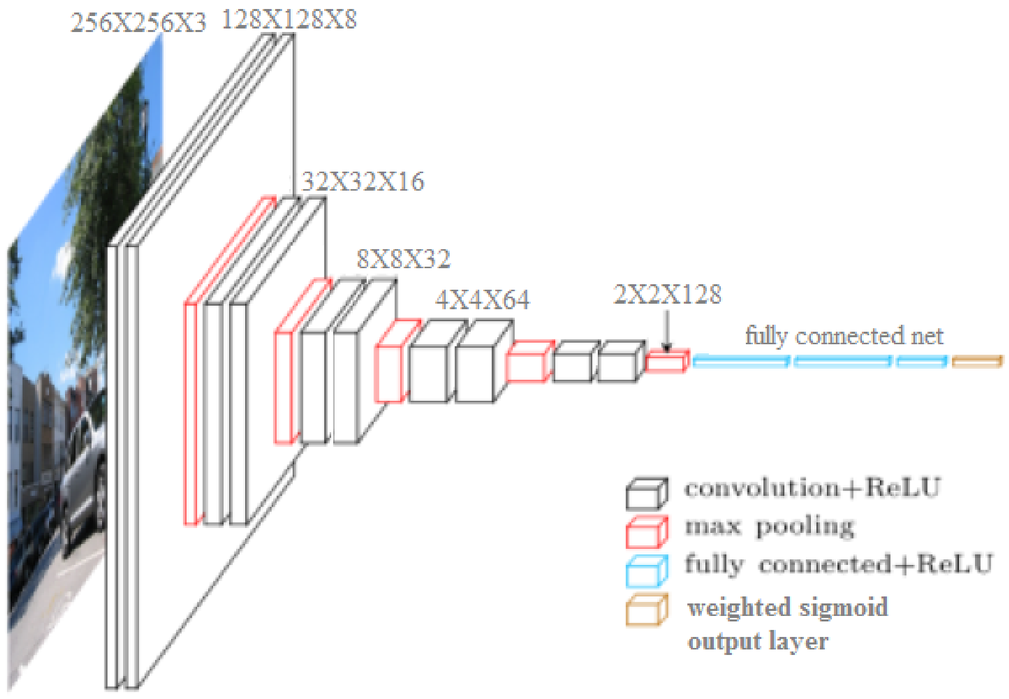
- First, the image is reduced to a size of 256X256X256 from the initial 1024X1024X3 available to us. This is done using openCV.

- Then, the labels in text format are retrieved from the .csv file, for each corresponding image. These labels, if more than one, are separated and then their index numbers are associated with the corresponding image features.
- The features which are now 256X256X3 in number for each sample is scaled or normalized to a range of [0,1] from the default color range of [0,255] for each color component.
- The labels are one hot encoded, so that we an array of 15 elements for each input feature, in which 1 represents the classes it belongs to.
- After these two preprocessing techniques, the data is divided into 3 segments: 70% data in training, 10% data in validation and 20% data in testing.
- This data is stored to disks with '.p' extension, and will be retrieved later.

It should be noted that, after preprocessing, the data is saved to disk in 13 training files, 1 validation file and 2 testing files. This further segmentation is done, so that the data can be loaded file wise, or in parts, if partial training or testing is needed.

Implementation

The implementation of the convolution network is done according to the following architecture and is explained below the diagram:



- The figures and flow of the above given architecture is exactly what have been used in the implementation in the jupyter notebook provided along with this report.

- The initial layers of network, which the data passes through, are convolution and max-pooling layers.
- The convolution layers are stacked in pairs and after each pair; there is a max-pooling layer. This fashion is repeated 5 times.
- In the second phase of network, the final output from the last max-pool layer is first flattened and then fed into a normal feed-forward fully connected neural network layers with rectified linear activations.
- In the last fully connected layer, i.e., the output layer, the total number of units is 15, one for each class present in our data.

Apart from this basic architecture, the usage and need of some intermittent functionalities is as follows:

- Usage of ReLu: ReLu activations give an output max(0,input). Hence, it doesn't let the gradient vanish while training using back-propagating, which is often the problem with sigmoid or tanh while using CNNs.
- Loss function: The loss function we are using is weighted sigmoid cross entropy, which is well suited for us, because we are using highly imbalanced training dataset. This loss function calculates the sigmoid cross positive, but gives a weight factor to the positives. This helps us to tell the network that it should not focus on the large number of true negatives it is getting. It should also take into account, the much smaller set of positives. I have kept the weighing factor at a value of 5.
- Optimization algorithm: The Adam optimizer is used to minimize the loss obtained. This algorithm dynamically varies the leaning factor and momentum to train the network.

The implementation of taking measurements and testing is as follows:

- While training the neural network, I have kept a look at only the F-score to see variations in training across different epochs. This F-score (defined in a previous section) takes into account a harmonic mean of precision and recall, both of which are more essential to us than accuracy, owing to the lower numbers of positives than negatives in our data.
- While testing, first both the testing files are loaded and then the precision, recall and F-score for both the vanilla neural network and convolution neural network are calculated and compared.

Refining

There were two particular areas which were considered and redefined along the course of this project, and hence lead to the final results given in the next section. The initial and final implementations along with the reasons of refinement are given as follows:

- **The choice of loss function** – Initially, I just concentrated on the fact that the output is multi-labeled and that each of the output activations should be independent of each other, hence I used sigmoid cross entropy, instead of softmax cross entropy used for data with single output label per feature. **This lead to the F-score running to 0.0 suddenly after a few epochs, while the training loss was still non zero and decreasing.** This was happening because the network, after a few epochs started to learn that one way to decrease the amount of training loss, is to give more and more negatives as output. This would result in lower loss because there will be less number of false positives. Increasing number of negative predictions, means decreasing number of positive predictions, and hence, the F-score, which takes into account true positives, instead of true negatives, was running to zero.

How I refined this – I tackled this problem by adding the attribute of weight to the loss function, wherein, the positive predictions were multiplied by a weight (>1), so that they are given more importance than the negative once. This was achieved by using tensorflow's weighted_cross_entropy_with_logits() function.

- **Adding dropouts:** The dropouts are added in the later stages of the network, i.e., the fully connected layers. The dropouts help dropping some weight in a given layer, and accordingly scaling the other weights at the time of training. These not only trains the model to work through interference, it also speeds up the training process, since the number of active neurons is decreased.

IV. RESULTS

Model Evaluation and Validation

The training of the model is done with the data present in training segment of the dataset. And the final parameters for this model were selected because they were the parameters with which the model gave the best results. All the implementations and refinement strategies provided in the previous section fit the problem statement and dataset well, and are providing a satisfactory result.

During the training, the validation was done on a separate validation batch of dataset. This helps us to keep the validating data separate from the testing data, so that the model actually learns, and not just memorize the dataset. Moreover, we know that the trained model is not data sensitive, as it is giving similar results on the completely different testing dataset, as it is giving on the validation data. This result proves that the model is generalized, even for data, which it hasn't seen earlier. The F-score on validation set was 0.53, and on the test set was 0.47 which are close to each other.

The model can be fully evaluated by referring to the architecture given above along with the following facts:

- The shape of each filter in the convolution filters is 5X5.
- The stride of filters in convolution layer pairs was 1 and 2 alternatively.

- The shape of filter in max-pool layer was 3X3, with a uniform stride of 2.
- The probability used for dropping out the neurons in the fully connected layer was 0.5.
- The training runs for 5 epochs.
- The batch size fed for training is 150 samples per batch.

After proper implementation, the result obtained on testing data was:
F-score: 0.47
Precision: 0.53
Recall: 0.42

Justification

The results obtained by the CNN are satisfactory and are compared as below:

- On the testing data, clearly the convolution network model has margin over the benchmark model, if we consider the F-score. The F-score of our model has reached 0.47, which is satisfactory, keeping in mind the limiting factor of the resources at my disposal.
- Moreover, as calculated above, the precision of the model has reached 0.53 (from 0.28 in our benchmark model). This indicates that, if this model predicts a positive, i.e., if it predicts a disease, or no disease, it can be relied upon over 50% of the times.
- Thirdly, the recall is 0.425, which means that if the person has a disease, there is a 42.5% chance that our model will be able to predict it.

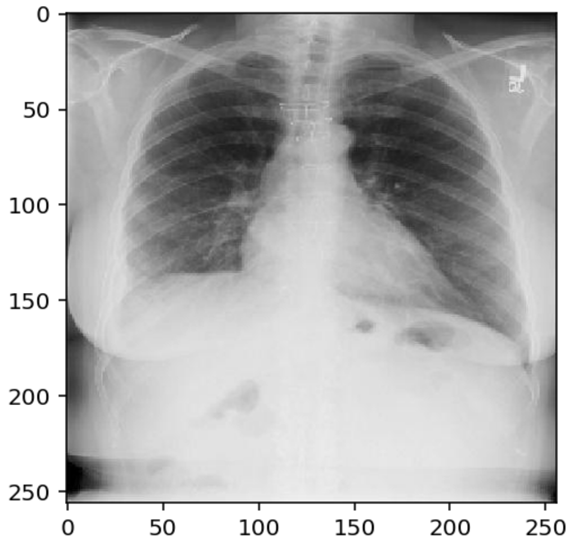
Keeping in mind, that the resolution of images used was just 256X256, and the network was not as deep as it could have been, due to limitation of computing resources, the results obtained are fairly satisfactory.

Though, a medical diagnostic system will surely need more accuracy than posed here, but these figures ensure that a deeper convolution network of the same form, trained over higher resolution images, is capable of generating results, which can be deployed in real world X-ray diagnosis prediction systems.

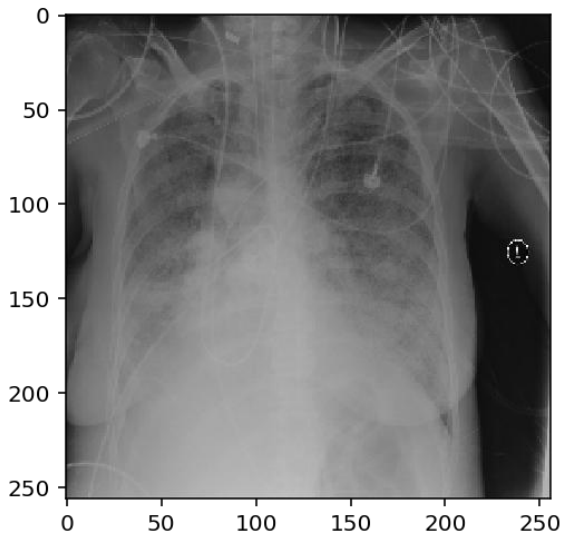
V. CONCLUSION

Free Form Visualization

I will consider free from visualization of the data that is actually being served as an input to our network, i.e., the 256X256X3 image:



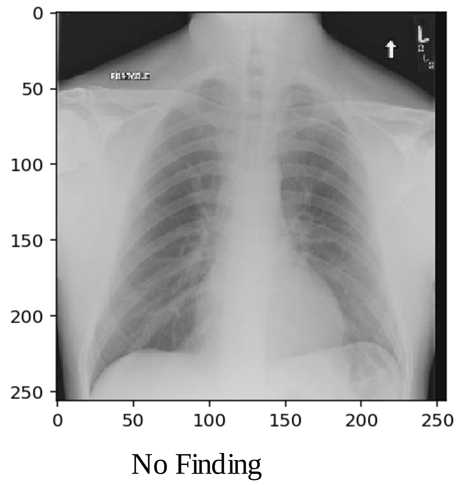
Pneumothorax



Fibrosis and nodule

A look at the reduced resolution images of the input feature space ensures, that, though the images are a little grainy and dark, as compared to the high resolution 1024X1024X3 images, still, most of the features and forms seem to be still preserved.

But, we should also be aware, that after dropping the resolution, some images become more hazy than others, a particular example of which is shown as below, in which nothing is clearly visible.



No Finding

This free-form visualization brings us to the conclusion, that depending upon image to image, the resolution dropping, has different effect on clarity and also the brightness of image.

Reflection

This particular project was as challenging, as it was fun to carry out. The problem was a real-world requirement, and to work in the direction of solving it, was truly motivating in itself. The outline of the process followed is:

- First, a relevant, public dataset was searched for, extracted, visualized and statistics were calculated to get insights into the data.
- Then, the data was normalized, labels were encoded, and the whole set was partitioned into different segments for testing, training, and validation.
- Then, a simple, one-layer vanilla neural net was implemented as benchmark was set.
- Then the architecture of CNN was coded and features were fed to it, and multiple epochs wer carried out.
- The CNN was optimized and refined according to the needs of dataset, and to increase the F-score.
- The CNN was tested on the dataset and final results were obtained.

Some difficulties that were faced along the course of this project are:

- Image size – Information tradeoff: Finding the right image size, which is neither too large for the resources I had, nor too small for the network to learn nothing at all.
- Imbalance in dataset: The highly imbalanced dataset was surely another problem, which dictated major aspects of this project.

In the end, I would just like to add, that the F-score obtained by the model, is satisfactory, and was around what was expected. With further improvements along the same lines, the model can be implemented in a real-world setting.

Improvement

There is scope for a lot of improvement in the existing CNN, the key points of which can be given as follows:

- Including the complete dataset provided by NIH- CC.
- Taking the features at maximum possible resolution.
- Making the CNN deeper, and using high speed GPUs for reduced delays in training and predictions
- Collecting more samples for diseases, in order to balance the dataset as much as possible. This will help us to increase efficiency of prediction, as well as it will help us to catch outliers easily.