COMPUTER VISION

PNEUMONIA DETECTION CHALLENGE

Final Report Submission for Post Graduate Program in

Artificial Intelligence and Machine Learning (PGP - AIML)

By

Nov 21B CV1 Group 1

1. Ramesh Gopinath
2. Thrilok Madduru
3. Arabinda
4. Amol Pimpale
5. Kishore Bhagat
6. Amit Kumar Singhall

MENTOR: DR. AAMIT LAKHANI

Submitted date (Nov 25, 2022)

Table of Contents

1.0 Summary of problem statement, data and findings

1.1. Problem Statement Summary:

In various healthcare applications, CNN is widely used in assisting the medical professionals while treating the patients for better decision making during the treatment of various diseases. One of such disease is pneumonia which can be detected through X-ray by detecting lungs inflammation which generates immense amount of data. In case there is a presence of opacity in the lungs, then it is classified as **"**Lung Opacity**"** and if there is no opacity then it is classified as "Normal". We have new rows in the data that are classified under "Not Normal No Lung Opacity" which says there is abnormality in the lungs even though there is no opacity. We have a set of medical images which are stored in DCM format which contains metadata and image arrays for pixel data.

1.2. Data Sets:

Stage_2_train_images.zip	-	Contains 26684 training images in .dcm format
Stage_2_test_images.zip	-	Contains 3000 test images in .dcm format
Stage_2_train_labels.csv	-	CSV file containing training labels x,y,width,height,target
Stage_2_detailed_class_info.csv	-	Contains class info of training images like whether lung opacity is normal, Lung Opacity, No Lung Opacity / Not Normal
Stage_2_sample_submission.csv	-	Contains 3000 patient id's which are to be predicted using the Trained model.

* train and test images contains different metadata stored along with the images for a patient.

1.3. Objective:

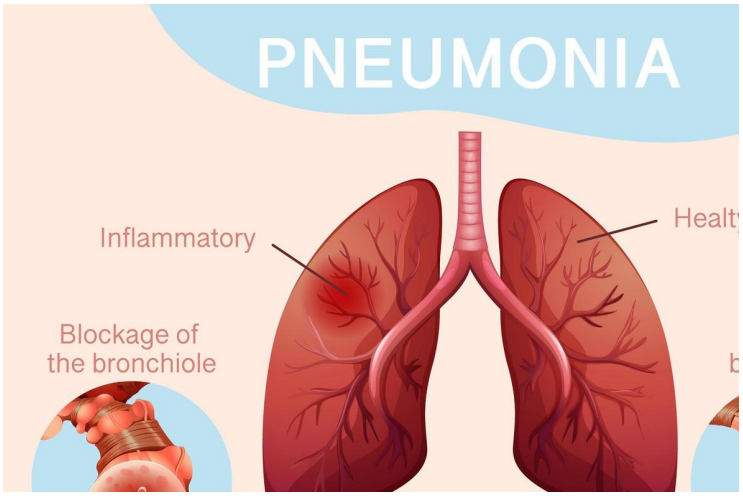
Design a deep learning-based algorithm for detecting pneumonia. We are making use of Deep Learning algorithms like Convolutional Neural Network models to properly predict the training images. In the later part, we will make use of transfer learning to train the model on given dataset and predict the test images.

1.4 Brief Introduction

1.4.1 What is Pneumonia disease?

Pneumonia is an infection that affects one or both lungs. It causes the air sacs, or alveoli, of the lungs to fill up with fluid or pus. Bacteria, viruses, or fungi may cause pneumonia. Symptoms can range from mild to serious and may include a cough with or without mucus (a slimy substance), fever, chills, and troubled breathing. How serious pneumonia is, depends on patients age, overall health, and what caused the infection.

To diagnose pneumonia, healthcare provider will review patients medical history, perform a physical exam, and order diagnostic tests such as a chest X-ray. This information can help determine what type of pneumonia the patient has.

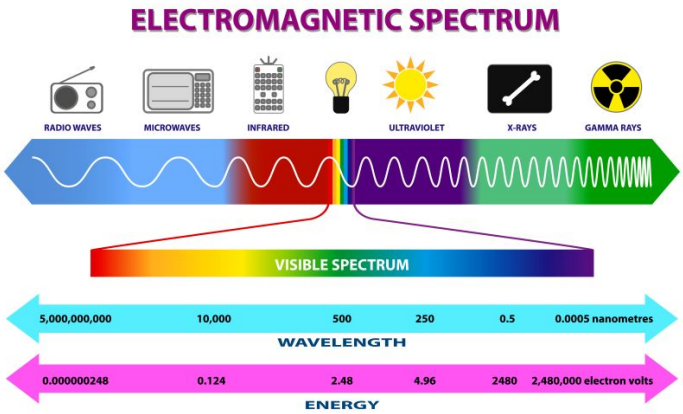


Pic Credits: [Poster for Pneumonia with Human Lungs 1109633 Vector Art at Vecteezy]

1.4.2 What are medical x-rays?

X-rays are a form of electromagnetic radiation, similar to visible light. Unlike light, however, x-rays have higher energy and can pass through most objects, including the body. Medical x-rays are used to generate images of tissues and structures inside the body. If x-rays traveling through the body also pass through an x-ray detector on the other side of the patient, an image will be formed that represents the "shadows" formed by the objects inside of the body.

One type of x-ray detector is photographic film, but there are many other types of detectors that are used to produce digital images. The x-ray images that result from this process are called radio graphs.



Pic Credits: [https://www.nibib.nih.gov/science-education/science-topics/x-rays]

1.4.3 AP and PA view of x ray:

1.4.3.1 Posterior-Anterior (PA) projection:

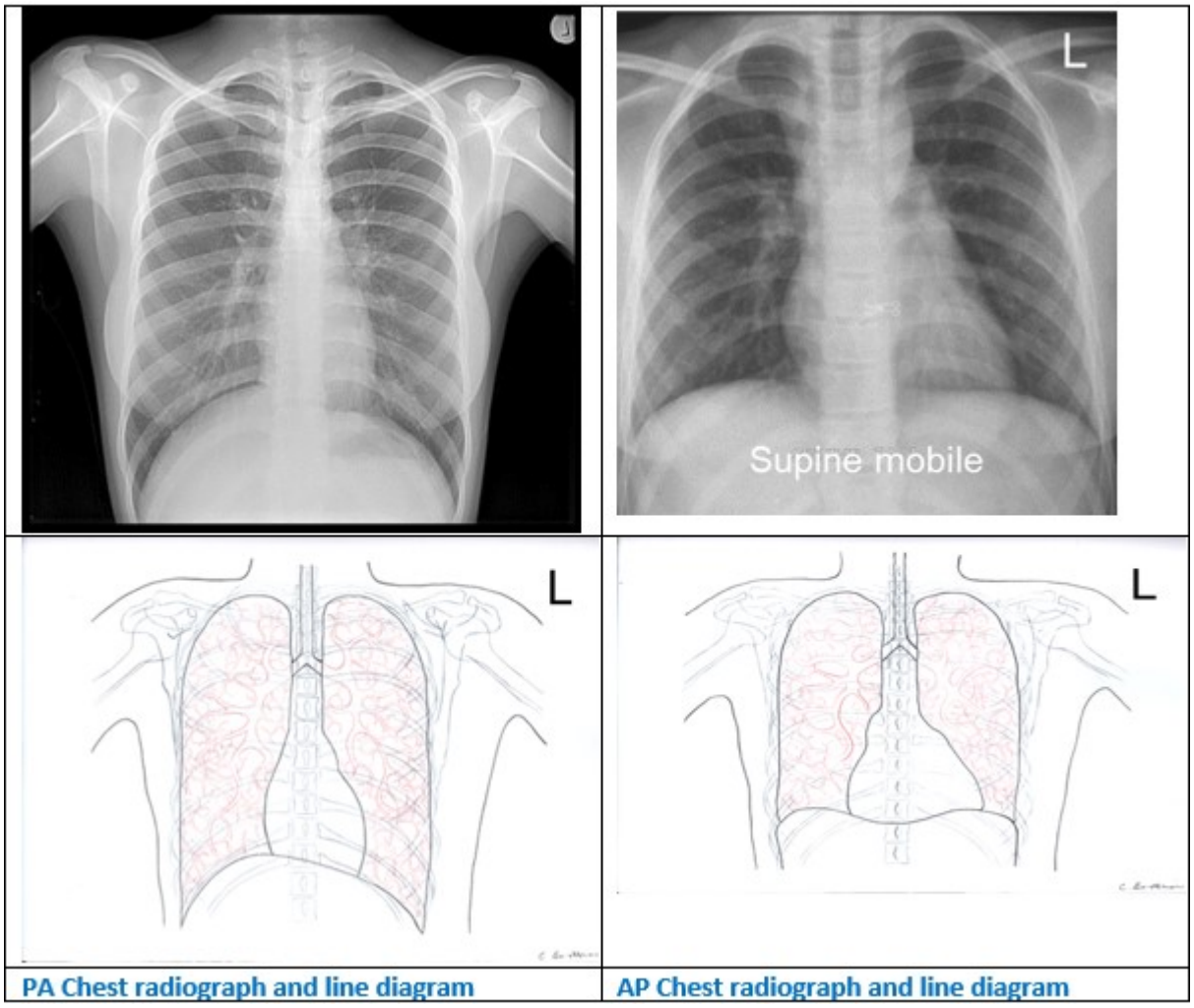
The standard chest radiograph is acquired with the patient standing up, and with the X-ray beam passing through the patient from Posterior to Anterior (PA).

The chest X-ray image produced is viewed as if looking at the patient from the front, face-to-face. The heart is on the right side of the image as you look at it.

1.4.3.2 Anterior-Posterior (AP) projection:

Sometimes it is not possible for radiographers to acquire a PA chest X-ray. This is usually because the patient is too unwell to stand.

The chest X-ray image is still viewed as if looking at the patient face-to-face.



Pic Credits: https://www.elearning.isrrt.org

1.4.3.3 AP v PA projection:

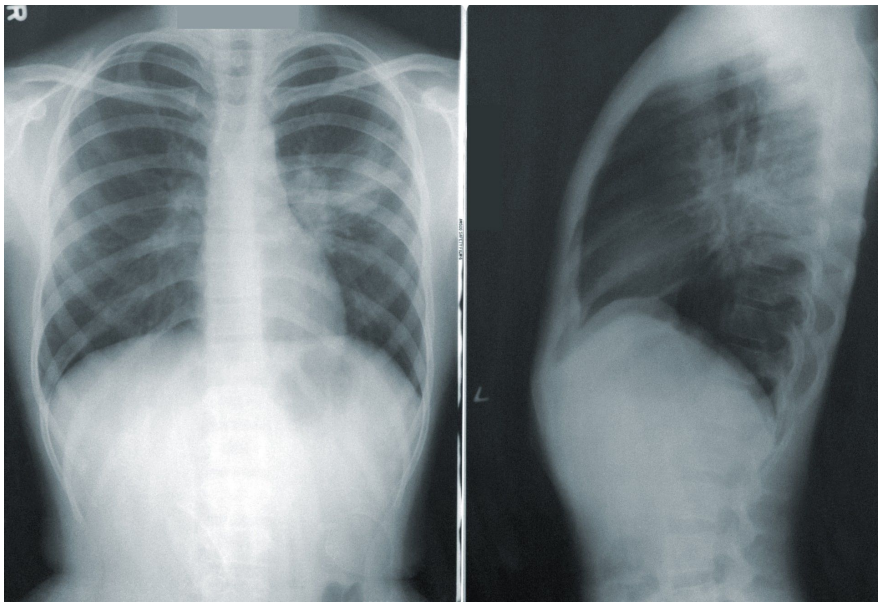
In AP Projection, Heart size is exaggerated because the heart is relatively farther from the detector, and also because the X-ray beam is more divergent as the source is nearer the patient.

In PA Projection, heart size is nearer to the real size, as the heart is relatively nearer the detector. Magnification of the heart is also minimised by use of a narrower beam, produced by the increased distance between the source and the patient.

1.4.4 What is DICOM?

DICOM (Digital Imaging and Communications in Medicine) is a standard format that enables medical professionals to view, store, and share medical images irrespective of their geographic location or the devices they use, as long as those devices support the format. DICOM images need to be viewed through specific software called DICOM viewers that can read and display the format. The images, along with the

corresponding patient data, are often stored in a large database called the Picture Archiving and Communication System (PACS). The purpose of a DICOM application is to store information in the PACS about the imaging examination, along with patient details, and then when required, to view and interpret (and possibly edit) medical images that are retrieved from the PACS. DICOM images are unique in the fact that they contain patient information in addition to the image data.



Pic Credits: https://towardsdatascience.com/how-to-use-fastai-to-evaluate-dicom-medical-files-738d7f7bc14d

1.5 METADATA SUMMARY

1.5.1 Metadata from dicom images:

COLUMN NAME	BRIEF INFORMATION
PATIENT ID	Primary identifier for the patient
X	Co-ordinate of image on X - Axis
Y	Co-ordinate of image on Y - Axis
WIDTH	Width of the image
HEIGHT	Height of the image
TARGET	1 - Pneumonia / 0 - No Pneumonia
CLASS	Normal, Lung Opacity, Not Normal / No Lung Opacity
SPECIFIC CHARACTER SET	ISO_IR 100, Character set description
SOP CLASS UID	'1.2.840.10008.5.1.4.1.1.7', The Secondary Capture (SC) Image Information Object Definition (IOD) specifies images that are converted from a non-DICOM format to a modality independent DICOM format.
SOP INSTANCE UID	Uniquely identifies the SOP Instance.
STUDY DATE	Date of the study started
STUDY TIME	Time of the study started
ACCESSION NUMBER	A RIS generated number that identifies the order for Th study
MODALITY	Type of equipment that originally acquired the data used to create the images in this Series.
CONVERSION TYPE	Describes the type of image conversion
REFERRING PHYSICIAN NAME	Name of the patients referring physician
SERIES DESCRIPTION	Description of the series - Posterior - Anteriror (PA) or Anterior - Posterior
PATIENT NAME	Full name of the patient
PATIENT BIRTH DATE	Birth date of the patient
PATIENT SEX	Sex of the patient. (Male / Female)
PATIENT AGE	Age of the patient
BODY PART EXAMINED	Description of the part of body examined
VIEW POSITION	Radiographic view of the image
STUDY INSTANCE UID	Unique identifier assigned to the study
SERIES INSTANCE UID	Unique identifier assigned to the series
STUDY ID	Equipment generated study identifier
SERIES NUMBER	Number to identify this series
INSTANCE NUMBER	Number that identifies this image
PATIENT ORIENTATION	Patient direction of the rows and columns of image
SAMPLES PER PIXEL	Number of samples in an image
PHOTOMETRIC INTERPRETATION	Specifies intended interpretation of the pixel data.
ROWS	No of rows in the Image
COLUMNS	No of columns in the image
MULTI PIXEL SPACING	Space between multiple pixels of an image
PIXEL SPACING / PIXEL SPACING 1	physical distance between the centers of each Two -dimensional pixel, specified by two numeric values
BITS ALLOCATED	No of bits allotted to each pixel sample
BITS STORED	No of bits stored for each pixel sample
HIGH BIT	Most significant bit for pixel sample data

PIXEL REPRESENTATION	Data representation of pixel samples
LOSSY IMAGE COMPRESSION	Specifies whether an image has undergone any image compression
LOSSY IMAGE COMPRESSION METHOD	Lossy Compression Method applied on image
FNAME	Name of each image file
IMG_MIN	Minimum pixel value in the image
IMG_MAX	Maximum pixel value in the image
IMG_MEAN	Average pixel value in the image
IMG_STD	Standard deviation of pixel values in the image
IMG_PCT_WINDOW	Image saved in Macintosh PICT format

1.6 EDA & VISUALIZATIONS

1.6.1 What is EDA?

EDA - Exploratory Data Analysis is an important activity applied to investigate the data in depth and learn different data characteristics, summarize key inputs, and better understand the features of the data often with visual means and find useful patterns in the data.

1.6.1.1 What are the steps involved?

1.6.1.1.1 Data Collection:

This is the essential process where we find and load the data into system. In the capstone project we are provided with below data:

i) 26684 Train images in Dicom Format.
ii) 3000 Test images in Dicom Format.
iii) Class information of training images.
iv) Label information of training images.
v) Sample submission file for test images.

We have unzipped the images file and saved them in respective folders. Class and Label information was read into a Pandas Data Frame. Apart from the image files and data csv files provided, we have metadata in the dicom images, which is extracted and stored in the Data Frame. After loading all the required data, we displayed basic information of the data loaded using various methods like .info(), head(), .tail()…etc.

1.6.1.1.2 Data Cleaning:

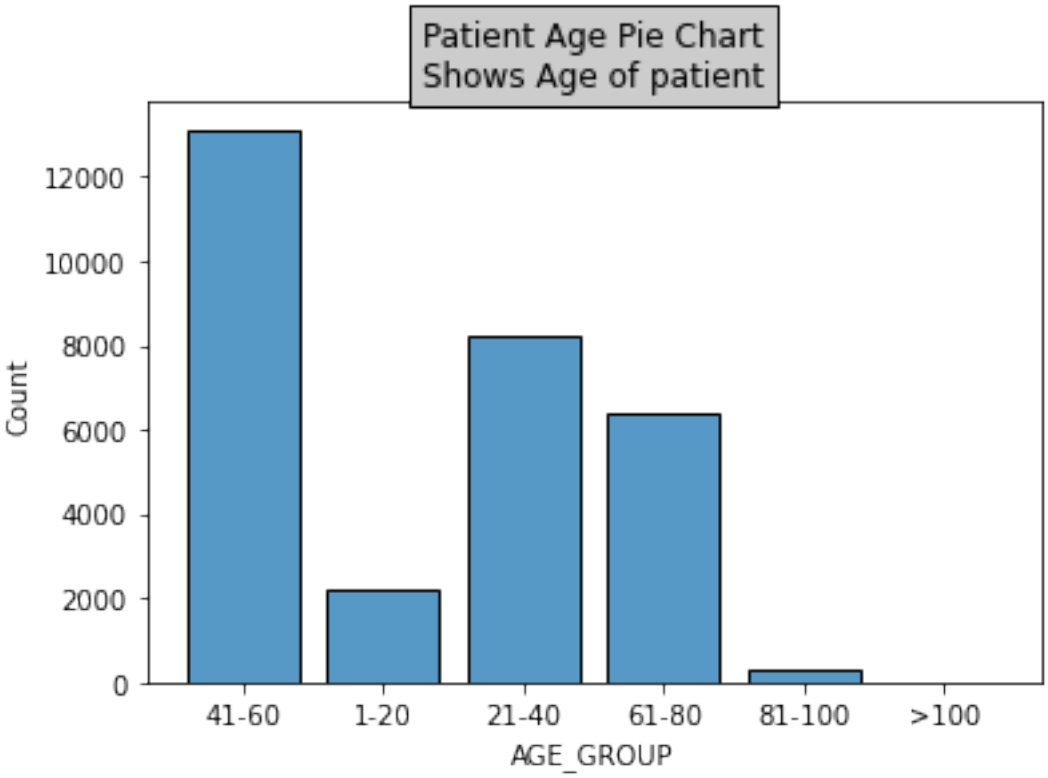
Here we look for various unwanted variables and values, getting rid of the irregularities, check for missing values and outliers and take necessary actions like dropping the rows / columns having unwanted / missing data. Deal with outliers and reduce the number of outliers in the data, replacing the missing values with significant values.

In the metadata of images extracted, we are having 25 columns where there is only single value and 5 out of those are having NaN value which doesn't have meaningful information that affects model performance and dropped. Hence we haven't considered these features while doing Uni-variate, Bi-variate analysis.

There seems to be some issues while collecting Age information, there are quite a few age reported abov e 100 which might not be true, there was one number even reported as 155. I guess

those are mistakes hence converting Age into bins of

- 01 - 20
- 21 - 40
- 41 - 60
- 61 - 80
- 81 - 100
- > 100



when observed there are large number of patients between 41-60 followed by 21-40, 61-80 and relatively lesser number of patients between 1 - 20, 81 - 100 and > 100

this observation looks logically correct since there will be very less patients in the largest age category 81-100 and > 100 and in the smallest age group of 1 - 20 (since there are greater probabilities of young people being hale and healthy)

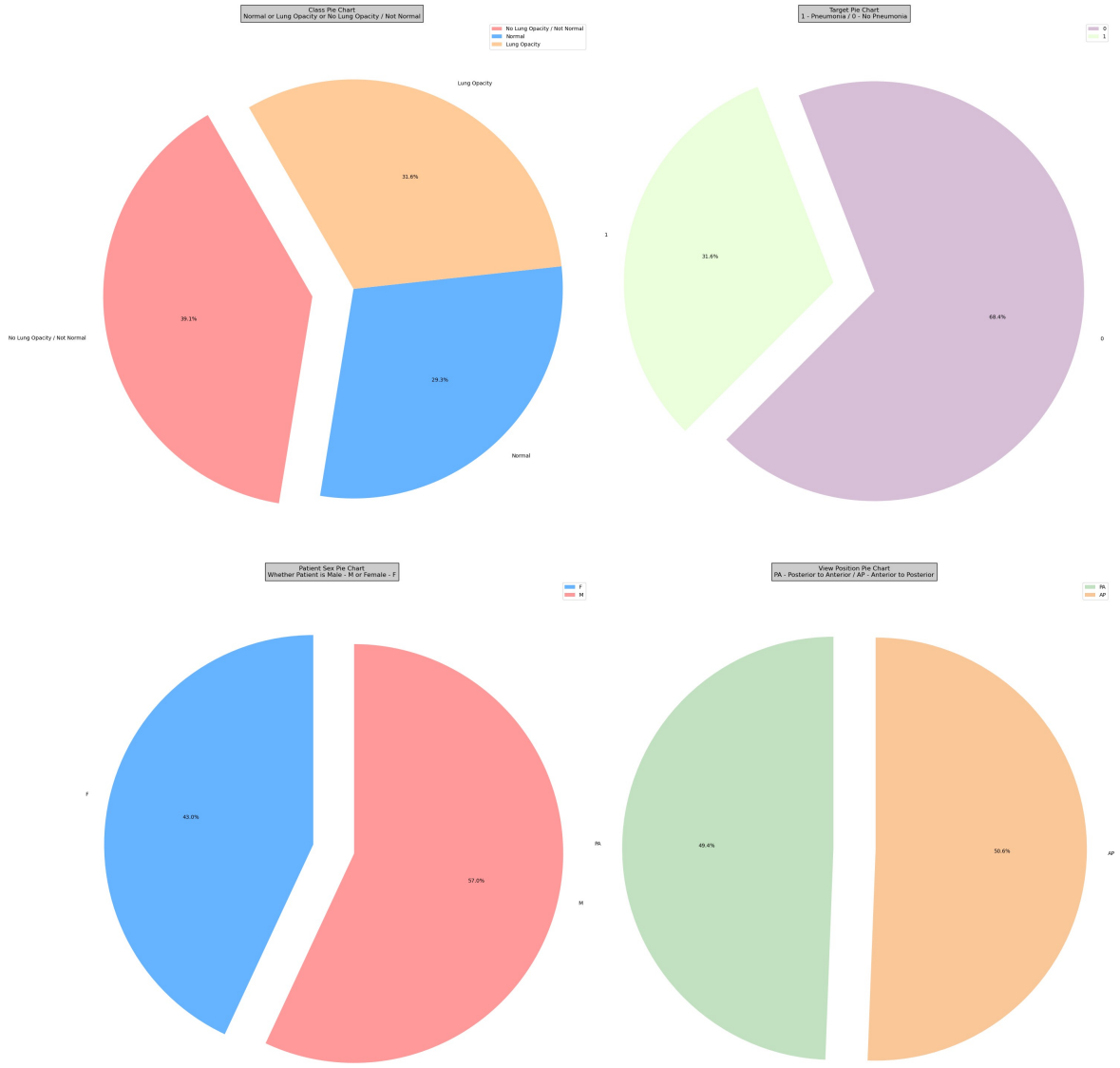
We have mapped the training class and labels information to the images where there 7402 duplicate records in the merged data frame which were dropped from the data frame to get back the original 30227 rows after the merge

a) Uni-variate Analysis:

As the name represents, in this analysis, we analyze the data of just a single feature or column. This can be done by using graphical methods using various visualization techniques or non-graphical methods by finding specific mathematical values in the data. We used graphical approach for this analysis on below features:

A. CLASS - Normal, Lung Opacity, No Lung Opacity / Not Normal
B. TARGET - Either 0 - No Pneumonia / 1 - Pneumonia
C. PATIENT SEX - M - Male / F - Female
D. VIEW POSITION - AP - Anterior to Posterior / PA - Posterior to Anterior

We used pie-charts for the graphical representation of the above features individually.



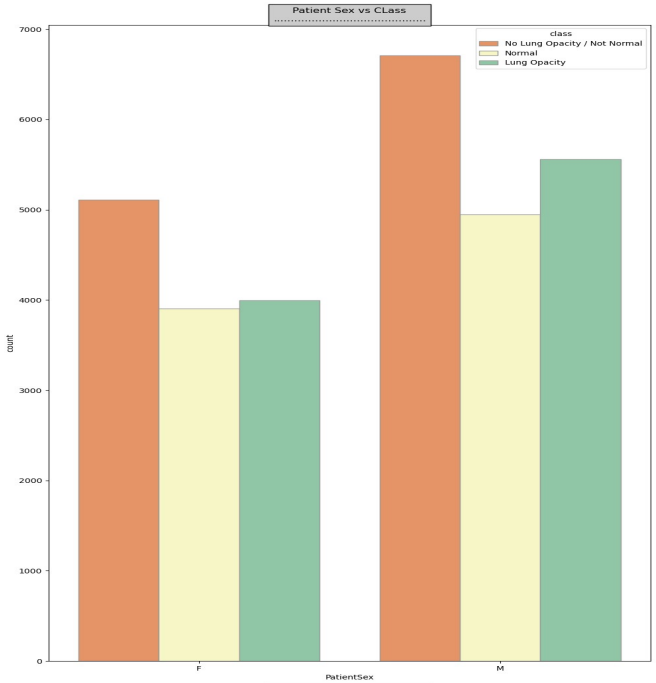
Observations:

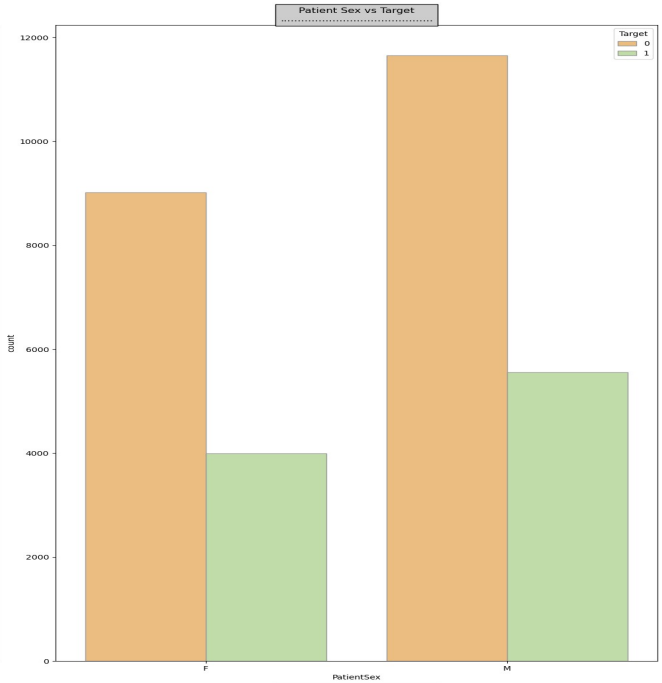
- Pneumonia is about 1/3 of the cases and non-Pneumonia are 2/3 of the cases
- More Males have got themselves checked than Females
- There are almost equal proportions of View Positions

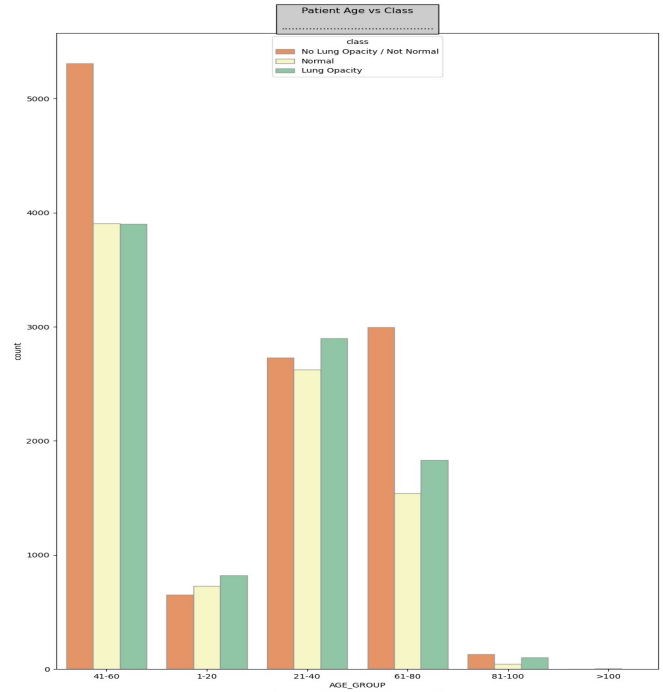
b) Bi-Variate Analysis:

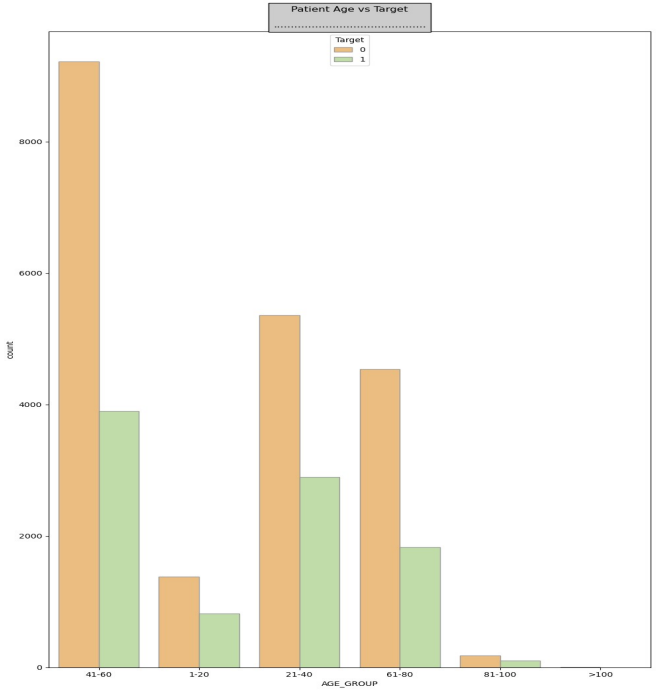
Here we use two features and compare them. In this we can find how one feature affect the other feature. Here we are doing below Bi-Variate analysis:

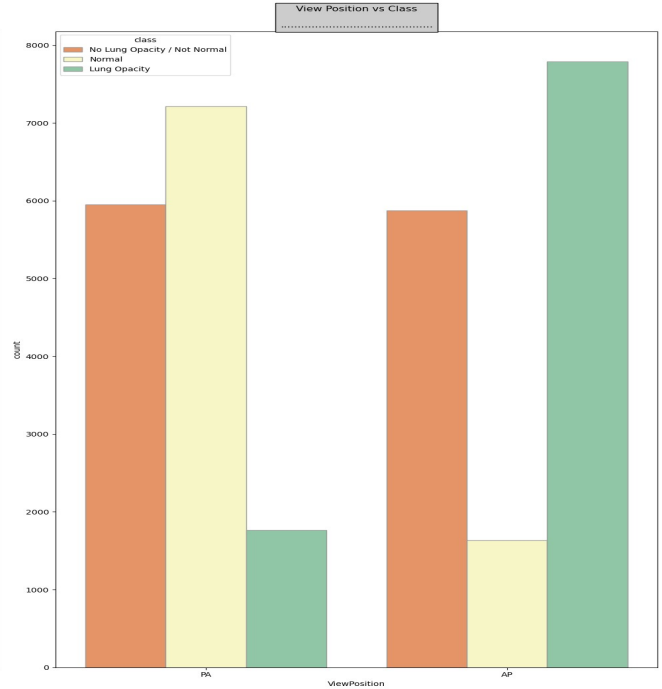
A. Patient Sex Vs Class
B. Patient Sex Vs Target
C. Patient Age Vs Class
D. Patient Age Vs Target
E. View Position Vs Class
F. View Position Vs Target
G. Class Vs Target
H. Centers of Lung Opacity Rectangles over Rectangle

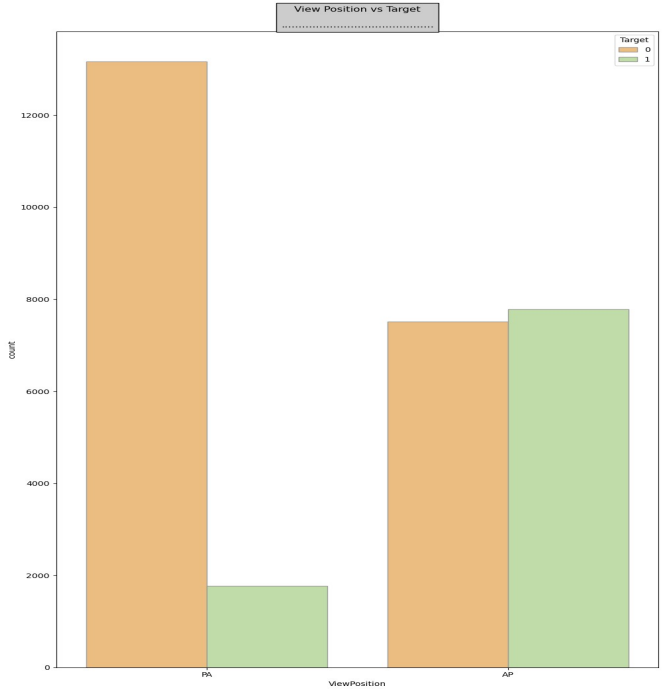

Patient Sex vs CLass
class
No Lung Opacity / Not Normal
Normal
Lung Opacity
count
PatientSex
F
M


Patient Sex vs Target
Target
0
1
count
PatientSex
F
M


Patient Age vs Class
class
No Lung Opacity / Not Normal
Normal
Lung Opacity
count
AGE_GROUP
41-60
1-20
21-40
61-80
81-100
>100


Patient Age vs Target
Target
0
1
count
AGE_GROUP
41-60
1-20
21-40
61-80
81-100
>100


View Position vs Class
class
No Lung Opacity / Not Normal
Normal
Lung Opacity
count
ViewPosition
PA
AP


View Position vs Target
Target
0
1
count
ViewPosition
PA
AP

Observations:

1. Sex Vs Class

Since there are Male patients than Female patients as observed earlier the proportion of Classes a lso seem to be almost similar between the genders

2. Sex Vs Target

Since there are more Male patients, the number of Pneumonia cases are also higher in them than in the Female patients. But more or less the proportions seem to be almost similar between the genders

3. Age group Vs Class

As observed earlier since the number of patients are very high in the 41-60 age category, the classes are also very high in that category. Normal cases and Lung opacity ca ses are almost the same in the 41-60 category with the Not normal cases being very high

Since 21-40 age category patients are next highest after 41-60, the number of classes are also high there. Interestingly all the 3 classes are almost similar in this age category

61-80 age has higher Not Normal cases than 21-40 Not normal cases

4. Age group Vs Target

Age group 41-60 has the highest Pneumonia cases followed by 21-40 then 61-80 and then 1-20

5. View Position Vs Class

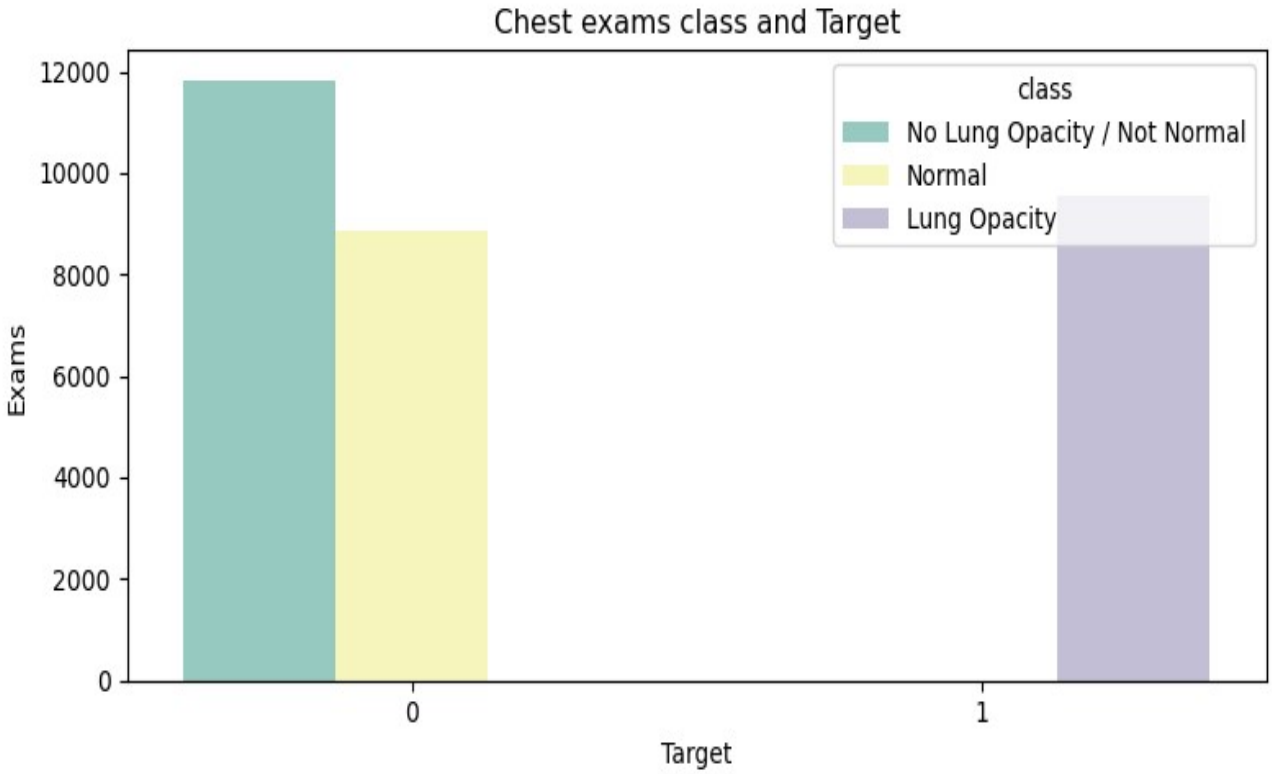
PA - Posterior to Anterior means X-ray source passes through the back of the patient with the chest facing the Film

AP - Anterior to Posterior means X-ray source passes through the front of the patient with the back facing the Film

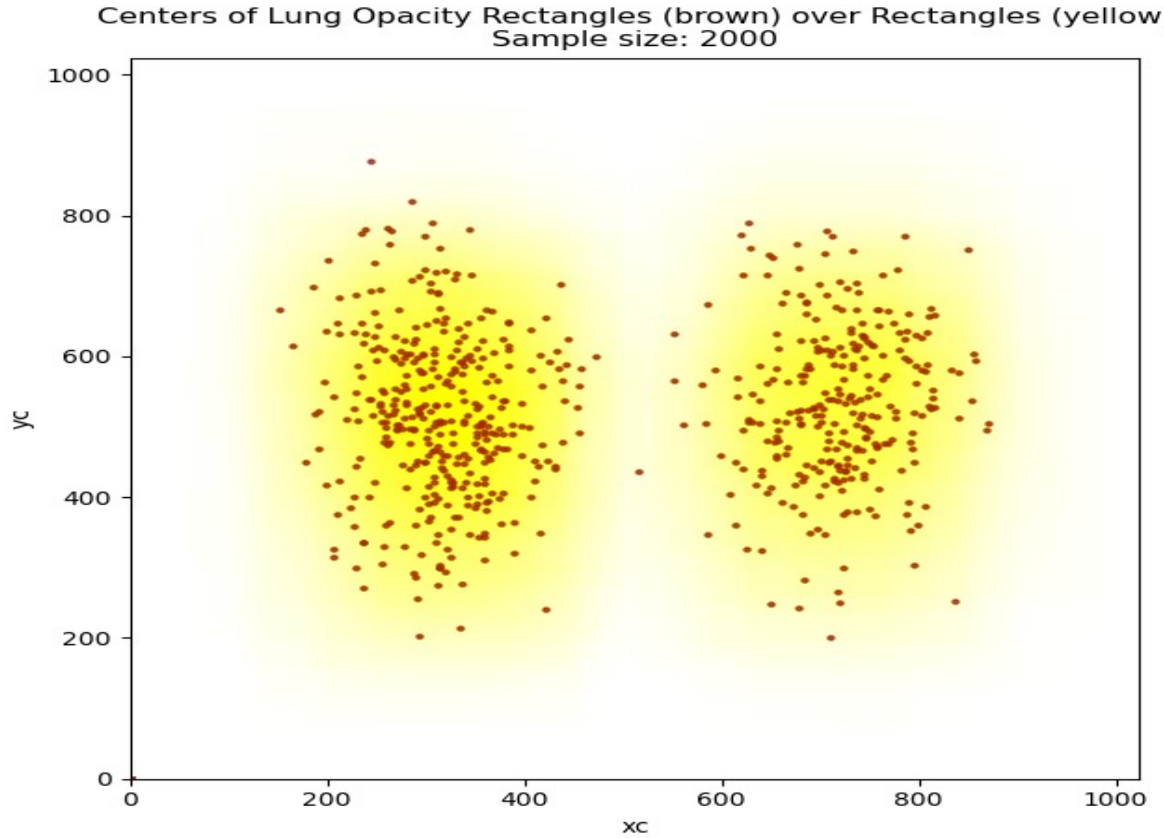
More Normal Classes are found in PA than in AP view
Nor normal / No Lung opacity classes are almost similar in both PA and AP views

6. View Position Vs Target

Huge number of Pneumonia cases found in AP views than in PA views



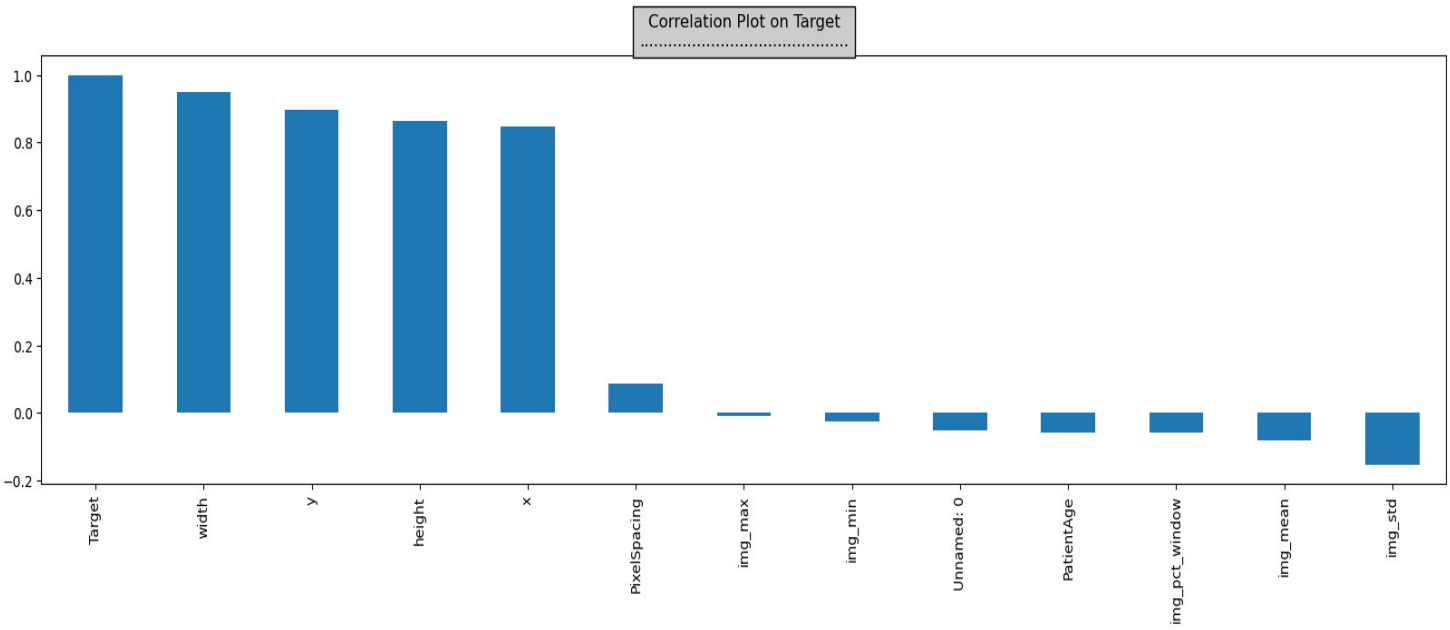
Observation: It is quite obvious that Pneumonia (Lung Opacity) shows up only in the Target = 1 and not in the Target = 0

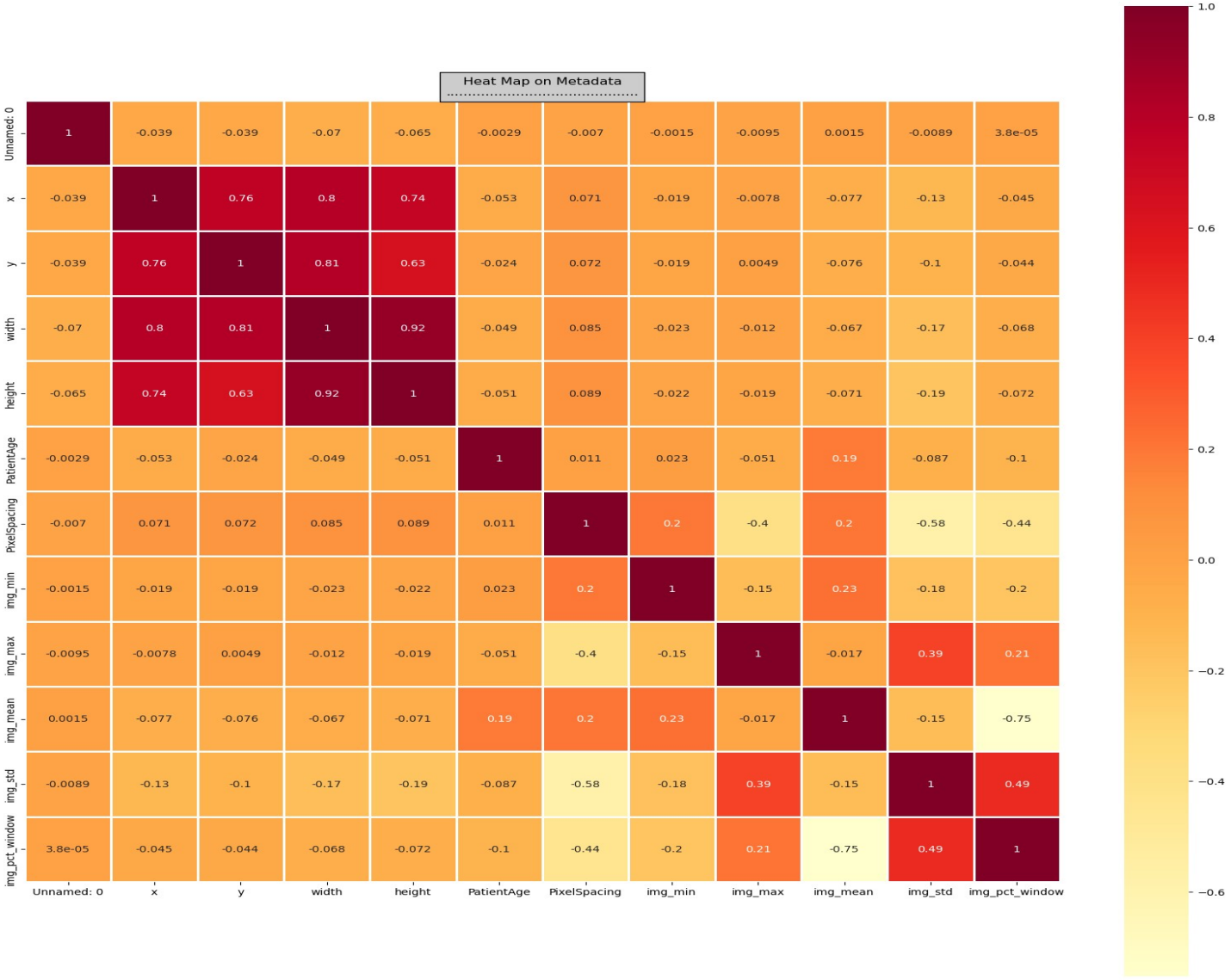


Observation: I guess this graph picks up a sample size of 2000 Pneumonia X-rays and shows the centres of Lung Opacity. This graph could possibly useful to visually see if there any patterns which suggest a huge number of Lesions / Inflammations / Lung Opacities observed in a certain part of the Chest. But in this particular graph it looks more or less equally spread out with slightly an increase in density on the left side maybe. **These kind of graphs could probably come to use during Pandemics and Outbreaks.**

c) Multi Variate Analysis:

In this analysis, we use more than 2 features from the data set and reveal the relationship among several features simultaneously. We have used pair plot, correlation plot, heat map and box plot to visualize the multi variables.






Box Plot on Metadata
30000
25000
20000
15000
10000
5000
0
Unnamed: 0
x
y
width
height
Target
PatientAge
PixelSpacing
img_min
img_max
img_mean
img_std
img_pct_window

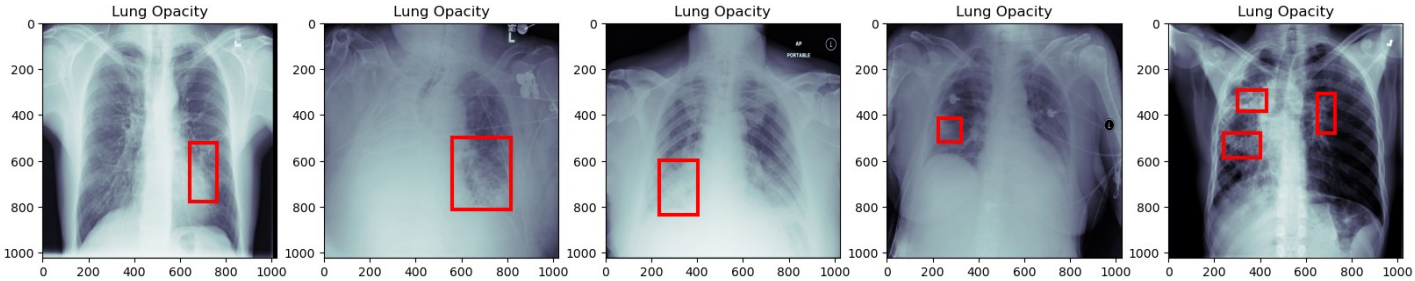
Image Pre-Processing:

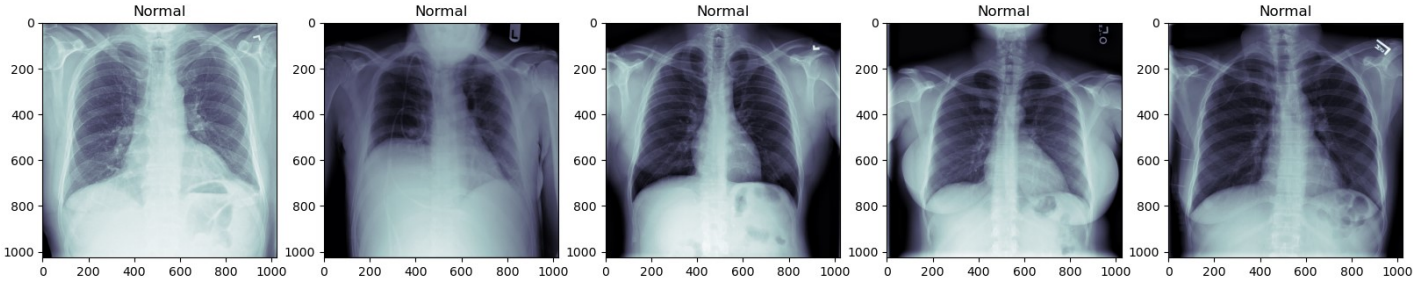
Data pre-processing / Cleaning is the crucial step in machine learning where we will deal with outliers, treating the missing values and remove any unwanted or noisy data. When we are dealing with data such as images, then we need to take care of additional techniques like dealing with pixel brightness - increase or decrease the size, brightness, transforming the geometry of the images, filtering the images, segmentation of the images, saturation / re-saturation.

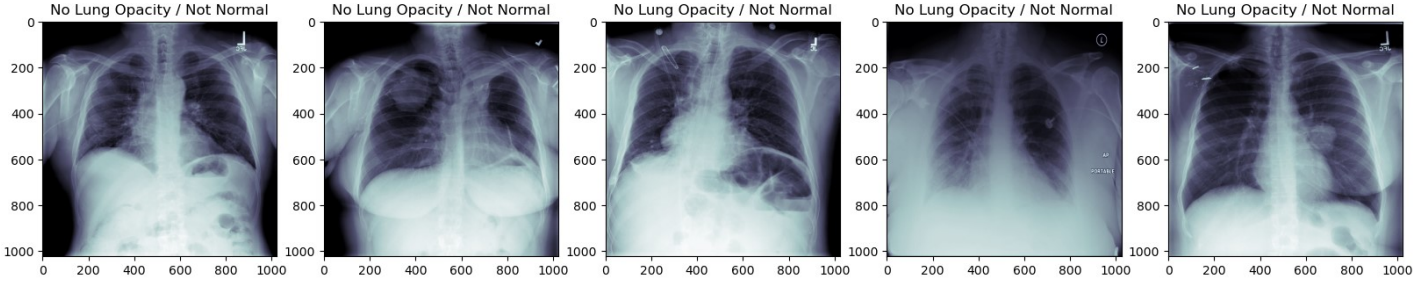
In the current project, we are provided with 26684 training images and 3000 testing images which are dicom images.

We are extracting the image zip files provided into respective folders. For ease of reference, we have segregated the images into different folders based on their class. A new data frame is created containing the image file data like file name, path, class and target details. Below are steps taken to pre-process the image.

A. Convert the images into gray scale
B. Resize the image into width: 224 and height = 224
C. Display the images with the bounding box rectangles.







d) CNN Model

There are 26684 Images which need to be fitted into a CNN model and Trained. Since picking up all the 26684 Dicom images, converting them into Pixel_arrays of 224 x 224 x 3 (RGB layers) would mean a huge of amount of Data for Training, Validation and Testing.

We decided to hand pick 6000 sample Images with the right proportion of 3000 Pneumonia

Images (Target = 1) and 3000 Not Pneumonia Images (Target = 0) to ensure there is no Target Bias while training and Validating the Model

Used from **pydicom.pixel_data_handlers.util import apply_color_lut** libraries and **rgb = apply_color_lut(img_3, palette='PET')**function to get the RGB layers of dimension 224 x 224

So the final X array shape is 6000 x 224 x 224 x 3

Y variable was the Targets (0 & 1) with shape **6000 x 1**

Converted y to_categorical [1,0] array so the shape changed to **6000 x 2**

Split X into X_train and X_test in 80% and 20% ratio with **stratify=y** (to ensure we maintain the y proportion) this resulted in the following

X_train (4800, 224, 224, 3) – Master Training dataset
y_train (4800, 2) – Master Training dataset
X_test (1200, 224, 224, 3) – Test dataset
y_test (1200, 2) – Test dataset

we further split the X_train & y_train into X_train1 – Training dataset and X_val and y_val (Validation dataset) with the same 80% and 20% ratio and again with stratify = "y_train" and got the following

X_train1 (3840, 224, 224, 3)
y_train1 (3840, 2)
X_val (960, 224, 224, 3)
y_val (960, 2)

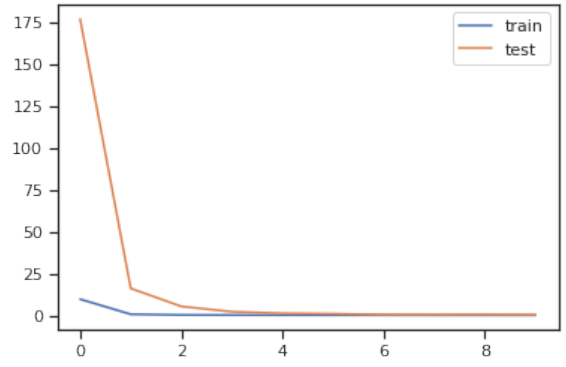
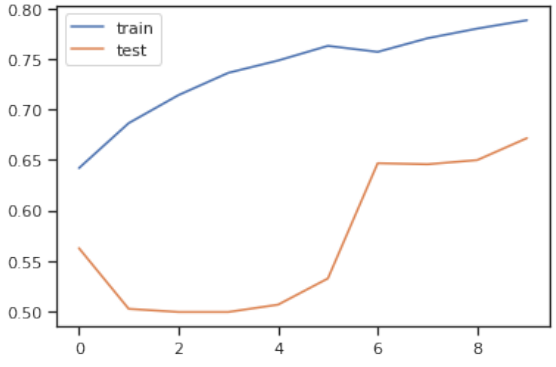
And Built 4 CNN models.

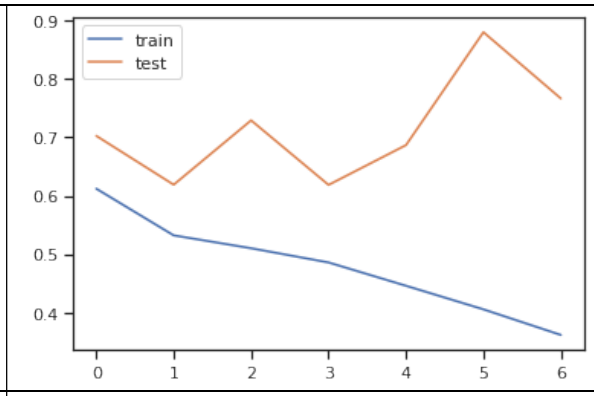
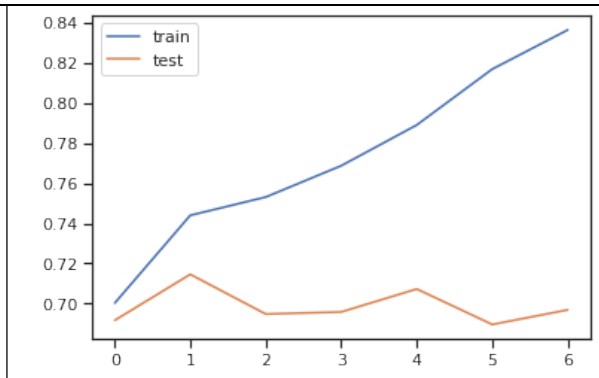
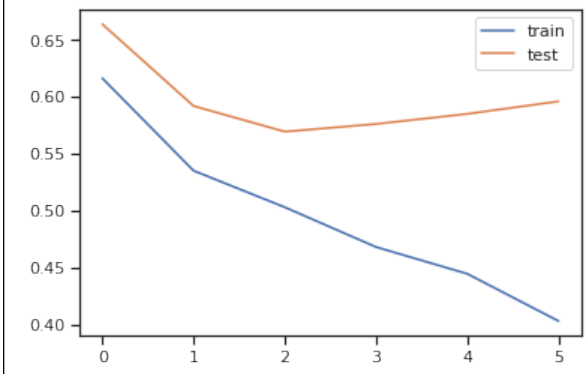
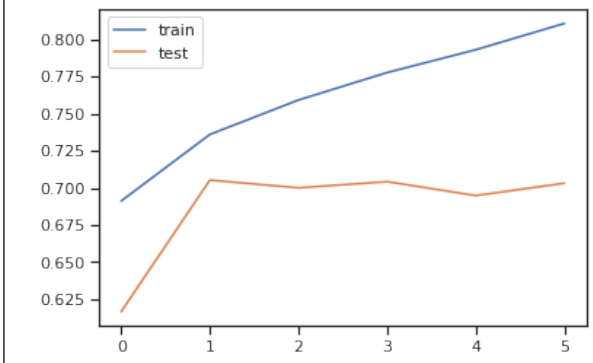
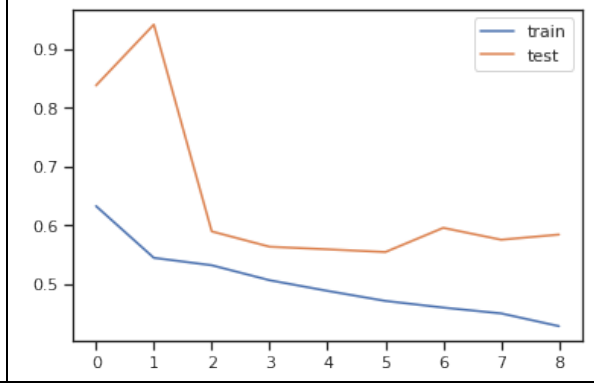
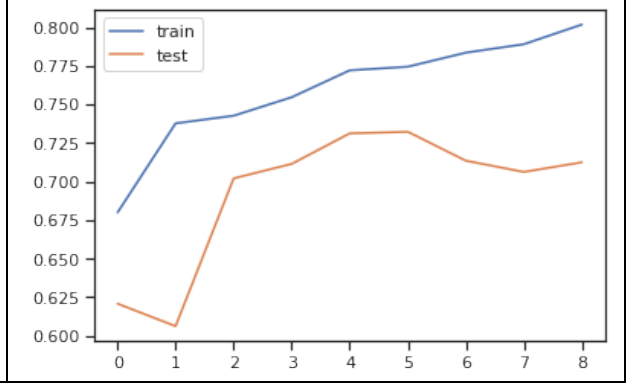
After building and compiling the model, we have to train the model by fitting the designed model to training and validation data set. We used batch size of 128 with 10 epochs. Using below plot figures, we can compare the loss and accuracy across training and validation data. This gives us the loss values and metrics for the model. It gives us the how well the model is generalizing the similar data on which it is trained. When the model is well fitted, it produces more accurate outcomes on the test data / new instances. In below table let us compare the loss and accuracy graphs for each of the 4 models trained. It shows how the loss and accuracies are varied across various epochs.

Once the model is trained on fitting the training and validation data set, we evaluated and predicted for test data to see how the 4 models are performed. This can be achieved through comparing the performance metrics like precision, recall, f1-score, accuracy.

Classification Report Comparison				
	precision	**recall**	**f1-score**	**support**
0	0.73	0.62	0.67	600
1	0.67	0.77	0.72	600
accuracy			0.69	1200
macro avg	0.70	0.69	0.69	1200
weighted avg	0.70	0.69	0.69	1200
	precision	**recall**	**f1-score**	**support**
0	0.69	0.71	0.70	600
1	0.70	0.68	0.69	600
accuracy			0.70	1200
macro avg	0.70	0.70	0.70	1200
weighted avg	0.70	0.70	0.70	1200
	precision	**recall**	**f1-score**	**support**
0	0.73	0.66	0.69	600
1	0.69	0.75	0.72	600
accuracy			0.70	1200
macro avg	0.71	0.71	0.70	1200
weighted avg	0.71	0.70	0.70	1200
	precision	**recall**	**f1-score**	**support**
0	0.74	0.69	0.71	600
1	0.71	0.76	0.73	600
accuracy			0.72	1200
macro avg	0.73	0.72	0.72	1200
weighted avg	0.73	0.72	0.72	1200

Model	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
Model	79%	67%	69%
Model 2	83%	70%	69%
Model 2a	81%	70%	70%
Model 3	80%	73%	72%

	Validation Loss Graph	Validation Accuracy Graph
Model		

Model2		
Model_2a		
Model3		

Improvisation Plan

All the Models seem to be high bias and high variance but when compared to the bias the variances are relatively lower.

There is almost 17-20% of bias in the models while the variances are 7 – 12%

The Validation accuracies are almost matching with the Final Testing accuracies and there isn't much difference there, not sure if it is due to the similar volume of data used for Validation set and Test set.

This performance can be improved by tweaking some parameters, adding / deleting layers, using other activation functions and so on. Below are some steps that can be taken to improve the model performance:

1. **Increase the size of datasets used to train the model** - The more the data, the better the chances for model to understand the patterns of the dataset in classifying them. We have considered only a sample of 6000 images for the Milestone 1.

2. **Image Augmenting:** All the Images might not be in the same exact shape, there could be some tilting, stretching and other orientational issues. We need to ensure the algorithm is Trained on all these so that when it encounters such alignment issues model can accordingly take note of it.

3. **Overfitting of the model** - When the model is really performing well against the Training data but underperforming on the Validation datasets or test data set, then the model is said to over fit. Our models seem to be having a high bias so first of all needs to be improved on the Training dataset but slightly under performing on the Validation and Test data sets. we can probably use more dropout to overcome the problem. Dropout helps in randomly switching off the neurons and reduces the complexity of the architecture.

4. **Lowering the learning rate** - Lowering the learning rate can help in finding an epoch where the model is performing better before over fitting. We have tried a few learning rates but probably there are some more to be explored

5. **Keras Tuner:** We could use Keras Tuner to figure out the best number of Convolutional layers, Neurons, kernels, Dense layers and Neurons along with the best optimizer, best learning rate etc and finally use that to get the best accuracies.

6. **Transfer Learning** - Transfer learning is improving of learning in a new task by transfer of knowledge from a related task that was already trained on similar tasks, in this case image classification.

2.0 Overview of the final process

We were supposed to work on the following

1. Fine tune previously built CNN models for Pneumonia Classification, fit them against our data and test for Accuracies.
2. Use Pre-Trained models as Transfer learning for Pneumonia Classification, fit them against our data and test for Accuracies.
3. Use Pre-trained models as Transfer learning for Object Detection of Lung Opacity bounding boxes.

3.0 Step-by-step walk through the solution

3.1 Datasets

3.1.1 Dicom images to Pixel array

Selecting a sample of 6000 Dicom Images and Converting them into a Huge X(Pixel array) of 6000 x 224 x 224 x 3

There are 26684 Images which need to be fitted into a CNN model and Trained. Since picking up all the 26684 Dicom images, converting them into Pixel_arrays of 224 x 224 x 3 (RGB layers) would mean a huge of amount of Data for Training, Validation and Testing.

We decided to hand pick 6000 sample Images with the right proportion of 3000 Pneumonia Images (Target = 1) and 3000 Not Pneumonia Images (Target = 0) to ensure there is no Target Class Bias while training and Validating the Model

Used from pydicom.pixel_data_handlers.util import apply_color_lut libraries and
rgb = apply_color_lut(img_3, palette='PET') function to get the RGB layers of dimension 224 x 224

So the final X array shape is 6000 x 224 x 224 x 3

Y variable was the Targets (0 & 1) with shape 6000 x 1

Converted y to_categorical [1,0] array so the shape changed to 6000 x 2

Split X into X_train and X_test in 80% and 20% ratio with stratify=y (to ensure we maintain the y proportion) this resulted in the following

X_train (4800, 224, 224, 3) – Master Training dataset
y_train (4800, 2) – Master Training dataset
X_test (1200, 224, 224, 3) – Test dataset
y_test (1200, 2) – Test dataset - Target

We further split the X_train & y_train into X_train1 – Training dataset and X_val and y_val (Validation dataset) with the same 80% and 20% ratio and again with stratify = "y_train" and got the following

X_train1 (3840, 224, 224, 3) – Training set
y_train1 (3840, 2) – Training set - Target
X_val (960, 224, 224, 3) – Validation set
y_val (960, 2) – Validation set – Target

3.1.2 Image Data augmentation using JPG images

Image Data Generator with the same set of 6000 Images (3000 Pneumonia and 3000 Non-Pneumonia images) converted to JPG images

The same Image dataframe with 3000 Pneumonia and 3000 non-Pneumonia images were used for this
We train-test split the 6000 Images dataframe with stratify on Target so that appropriate ratio of Targets will be used both in Train as well as in Validation

80% of Training resulted in 4800 rows with 2400 Pneumonia and 2400 Non-pneumonia images
20% of Validation resulted in 1200 rows with 600 Pneumonia and 600 non-Pneumonia images

These Dicom images were separated out from stage_2_train_images and loaded into Train_6k folders and Val_6k folders
Next all these 4800 Train Dicom images and 1200 Validation Dicom images were converted to Jpg files and stored in Train_jpg_6k and Val_jpg_6k folders
All the Training Images were then augmented on the following basis

```
rotation_range = 20,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True,
fill_mode = 'nearest'
```

STEP_SIZE_TRAIN=round_up(train_generator.n/train_generator.batch_size)
STEP_SIZE_VALID=round_up(valid_generator.n/valid_generator.batch_size)

Batch_size was 32, so Train_batch was setup at 150 (4800/32)
While Validation_batch was setup at 38 (1200/32)

3.2 Fine Tuning

As a part of Fine tuning we tried the following

a) Experimented by Increasing the Convolutional layers
b) Experimented by Increasing Neurons in the Convolutional layers
c) Experimented by changing the Kernel_size - 7,5,3
d) Experimented with the strides(1,1), (2,2)
e) Added padding = 'same'
f) Experimented with kernel_regularizer & activity_regularizer
g) Added Dropout layers in between
h) Experimented Adam optimizer - learning rate, beta_1, beta_2, epsilon=1e-07
i) Experimented with SGD optimizer - learning_rate, momentum
j) Experimented with RMSProp optimizer - learning_rate, momentum, epsilon.
k) Experimented with Batch sizes and Epochs

Observation:

All of the above experimentation were carried on with 2 different datasets as mentioned above

i) Huge pixel array created from 6000 Dicom images
ii) Image Data Generator with same 6000 Jpg images

Both the approaches gave us a max of 72-73% accuracy and did not improve much from the Base CNN models at all.

Various approaches

In order to get the desired performance from the model, we need to make small adjustments to the base model so that the model adapts to the training better and improvise on its learning and achieves more accuracy.

How to achieve desired output from fine-tuning?

In our project, we are using CNN model for image classification and we have performed all the below steps as part of fine-tuning.

1. Increasing number of images to 10000 from 6000
2. Convert the images to JPEG format
3. Reduce the image size to 128x128. (Both JPEG & Dicom images)

Loss & Accuracy in Base models:

MODEL_NAME	LOSS	ACCURACY
model	67.16	71.25
model_2	70.9	68.17
model_2a	74.99	67.75
model_3	59.18	69.58

3.2.1 Increasing number of images to 10000 from 6000:

We have increased the number of images by almost 67% to 10000 from 6000. If there are more number of images are fed to train the model, we expect that the model has more combination of data to learn and train from more images and thus understand the patterns in test data thus increasing the model accuracy. But it is observed that only model_2a has shown more than 10% of increase in the testing accuracy as compared to the base model. Below are the Loss & Accuracy after tuning:

MODEL_NAME	LOSS	ACCURACY
model	53.7	74.1
model_2	76.91	65.65
model_2a	55.8	74.5
model_3	52.84	72.95

Model parameters for Model_2a:
Total params: 8,398,050
Trainable params: 8,397,090
Non-trainable params: 960

3.2.2 Convert the images to JPEG format:

Here we have changed the images from dicom format to jpeg format. Here dicom images are compressed into jpeg format where there is chance of data in the original images are lost. But since the images are compressed, the jpeg images takes less memory and speeds up the training. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	75.89	69
model_2	59.7	70.5
model_2a	61.51	68.67
model_3	53.46	73.83

Model parameters for Model_3:
Total params: 4,402,780
Trainable params: 4,402,524
Non-trainable params: 256

3.2.3 Reduce the images to 128x128:

A. **Using JPEG IMAGES**

Here we reduced the jpeg images to 128x128. Reducing the size of image can lead to faster training due to its less memory occupancy. We observed that models are trained much faster when the image size is reduced. We could see there is sligt improvement in model, 2, 2_a performance over 224x224 jpeg image files and also against the base models. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	53.58	73.25
model_2	61.66	71.83
model_2a	58.06	70.58
model_3	56.6	72.25

Model Parameters for Model:
Total params: 5,184,898
Trainable params: 5,184,706
Non-trainable params: 192

B. **Using DICOM images**

We reduced the dicom images size to 128x128 in this step. Earlier we used 224x224 in model training, but by reducing the size further to 128x128, models are trained faster. Also we can observe that as the models are getting trained and trained, there is reduction in the loss produced by the models in fine tuning as compared to base models. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	55.07	73.83
model_2	61.08	67.92
model_2a	54.65	73.08
model_3	56	73.25

Model Parameters for Model:
Total params: 5,184,898
Trainable params: 5,184,706
Non-trainable params: 192

3.2.4 Modify optimizer parameters and changing loss function:

Here we added beta and epsilon to the Adam optimizer and reduced the learning rate. Changed the loss function from binary_crossentropy to categorical_crossentropy. Since the learning rate is reduced, we expect the models to train very slow.Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	59.08	71.42
model_2	56.94	70.5
model_2a	55.57	71.83
model_3	57.46	69.83

Model parameters for model_2a:
Total params: 4,402,780

Trainable params: 4,402,524
Non-trainable params: 256

3.2.5 Reduce the number of images per batch:

By reducing the number of images per batch, we expect there is less load per batch and models can understand the images much better. By this we can either expect bump in model performance in either ways - one model understands the data in images much better and increase accuracy or due to less images fed per batch, it may not understand certain patterns in the images and thus reduces in accuracy even more. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	63.82	69.17
model_2	69.31	50
model_2a	64.93	50
model_3	69.33	50

Model Parameters for Model:
Total params: 5,184,898
Trainable params: 5,184,706
Non-trainable params: 192

3.2.6 Increase the number of images per batch:

On increasing the number of images per batch, model has more information at a time to learn and train. But we can see model_2 & 3 has very less accuracy as compared to base models and remaining 2 models shown slight improvement. Also note that models being trained for a longer period continuously which might lead to leakage of information. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	61.09	71.5
model_2	69.32	50
model_2a	56.26	71.58
model_3	69.32	50

Model parameters for Model_3:
Total params: 4,402,780
Trainable params: 4,402,524
Non-trainable params: 256

3.2.7 Add new layers / Update existing layers in the model:

In the earlier steps, we have not played with model layers but only introduced more data, also jpeg images, changed loss function, optimizer, reduced the image size to 128x128. But in all the scenarios there is not much of a rise in the model accuracy. We have added more convolutional layers to the model to improve the perfrmance. We are also going to modify the layers by adding kernel_regularizer, strides and padding and compare the model performance. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	24851053	68.67
model_2	10705	50.67
model_2a	8354902	69.92
model_3	58	70.67

Model parameters for model_3:
Total params: 41,809,140
Trainable params: 41,808,180
Non-trainable params: 960

3.2.8 Using RMSProp Optimizer function:

We are changing the optimizer function from Adam to RMSProp in this case. Below are the Loss & Accuracy after training:

MODEL_NAME	LOSS	ACCURACY
model	28070.41	50
model_2	135.95	50.75
model_2a	770954.59	50
model_3	62.97	70.83

Model parameters for model_3:
Total params: 41,809,140
Trainable params: 41,808,180
Non-trainable params: 960

3.2.9 Using Image Data Generator:

Image data generators allows you to generate the batches of images where only small set of image data is used to train the model in batches since a model cannot fill more number of images in a batch while training. It also helps in data augmentation where adds noise to the data when we are having only limited data. We have created the image data generator using appropriate arguments and used it to generate train and test flow from the folders where the images are stored.

```
train_datagen = ImageDataGenerator(preprocessing_function = preprocess_input,
                rotation_range = 20,
                width_shift_range = 0.2,
                height_shift_range = 0.2,
                shear_range=0.2,
                zoom_range=0.2)

valid_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)
Test_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)
```

- ✧ **height_shift_range:** Shifts the image along the height dimension. It supports various inputs. For float the image shall be shifted by fraction of total height, if < 1, or pixels if >= 1.
- ✧ **rotation_range**: Int. Degree range for random rotations.
- ✧ **width_shift_range**: Shifts the image along the width dimension.

- ✧ **Shear_range**: 'Shear' means that the image will be distorted along an axis, mostly to create or rectify the perception angles.
- ✧ **Zoom_range**: This method randomly zooms the image either by zooming in or it adds some pixels around the image to enlarge the image.
- ✧ **class_mode**: One of "categorical", "binary", "sparse", "input", or None. Default: "categorical". This determines the type of label that is returned by the generator.
  'categorical' is for multiple classes. Labels are one hot encoded.
  'binary' is for two classes.
  'sparse' returns 1D integer labels
  'input' this is useful for auto-encoders where you require the input image to be the label
  'None' will cause the generator to return no label
- ✧ **Interpolation**: Interpolation method used to resample the image if the target size is different from that of the loaded image.
- ✧ **target_size**: Tuple of integers (height, width), default: (256, 256). The dimensions to which all images found will be resized
- ✧ **directory**: string, path to the target directory. It should contain one subdirectory per class.
- ✧ **preprocessing_function**: function that will be applied on each input. The function will run after the image is resized and augmented.

Below are the Loss and Accuracy after using ImageDataGenerator:

MODEL_NAME	LOSS	ACCURACY
model	29586746.88	63.33
model_2	1453439.45	50
model_2a	14527342.19	61.42
model_3	60.62	68.67

Model parameters for model_3:
Total params: 41,809,140
Trainable params: 41,808,180
Non-trainable params: 960

FINE TUNING METHODS	model		model_2		model_2a		model_3	
	LOSS	ACCURACY	LOSS	ACCURACY	LOSS	ACCURACY	LOSS	ACCURACY
BASE MODEL	67	71.25	71	68.17	75	67.75	59	69.58
10000 IMAGES	54	74.1	77	65.65	56	74.5	53	72.95
JPEG FORMAT	**76**	**69**	**60**	**70.5**	**62**	**68.67**	**53**	**73.83**
JPEG 128X128	54	73.25	62	71.83	58	70.58	57	72.25
DICOM 128X128	**55**	**73.83**	**61**	**67.92**	**55**	**73.08**	**56**	**73.25**
LEARNING RATE, OPTIMIZER	59	71.42	57	70.5	56	71.83	57	69.83
REDUCE BATCH SIZE	64	69.17	69	50	65	50	69	50
INCREASE BATCH SIZE	61	71.5	69	50	56	71.58	69	50
ADD MODEL LAYERS	24851053	68.67	10705	50.67	8354902	69.92	58	70.67
RMSPROPOPTIMIZER	28070.41	50	135.95	50.75	770954.59	50	62.97	70.83
IMAGE DATA GENERATOR	29586746.88	63.33	1453439	50	14527342	61.42	60.62	68.67

3.3 Pre-Trained Model

Used the following Pre-trained models

a) VGG16
b) Resnet50
c) InceptionV3
d) Densenet121 with Imagenet weights
e) Densenet121 with Chexnet weights

Observation:

All of the above experimentation were carried on with 2 different datasets as mentioned above

iii) Huge pixel array created from 6000 Dicom images
iv) Image Data Generator with same 6000 Jpg images

There wasn't much difference in accuracies between both the datasets and the best accuracy observed was with **Densenet121 with Chexnet weights – 80%**

Object Detection with Bounding box

Data used:

4500 Pneumonia Images (Dicom Images) were used for this. Each image has its own bounding boxes. Converted images into Pixel_arrays of 224 x 224 x 3 (RGB layers).

Used rgb = np.stack([img_3]*3,axis=2) to get the RGB layers of dimension 224 x 224

So the final X array shape is **4500 x 224 x 224 x 3**

Y variable were the 4 bounding boxes x,y,width, height with shape **4500 x 4**

Split X into X_train and X_test in 80% and 20% ratio, this resulted in the following

X_train shape (3600, 224, 224, 3) – Master Training set
y train shape (3600, 4) – Master Target set
x_test shape (900, 224, 224, 3) – Final Testing set
y_test shape (900, 4) – Final Target set

We further split the X_train & y_train into X_train1 – Training dataset and X_val and y_val (Validation dataset) with the same 80% and 20% ratio and got the following

X_train1 shape (2880, 224, 224, 3) – Training set
y train1 shape (2880, 4) – Training Target set
X_val shape (720, 224, 224, 3) – Validation set
y_val shape (720, 4) – Validation Target set

The X array was further reshaped to 128 x 128 x 3 since we our Models used for Object detection were using 128 x 128 x 3 as input shape

```
X_train2 = np.resize(X_train1,(X_train1.shape[0],128,128,3))
X_val2 = np.resize(X_val,(X_val.shape[0],128,128,3))
X_test2 = np.resize(X_test,(X_test.shape[0],128,128,3))
```

And since the Predictor variables were being down scaled to 128 from 1024, we also had to downscale the Y (Target) bounding boxes also to 128 x 128 from 1024 x 1024 numbers

```
y_train2 = (y_train1 * 128) / 1024
y_val2 = (y_val * 128) / 1024
y_test2 = (y_test * 128) / 1024
```

So finally, the shapes of Predictor variables and Target variables were

X_train2 shape (2880, 128, 128, 3) – Training set
y train2 shape (2880, 4) – Training Target set (Downscaled to 128 x 128)
X_val2 shape (720, 128, 128, 3) – Validation set
y_val2 shape (720, 4) – Validation Target set (Downscaled to 128 x 128)
x_test2 shape (900, 128, 128, 3) – Final Testing set
y_test2 shape (900, 4) – Final Target set (Downscaled to 128 x 128)

Used 2 the following models to do the Object detection and to identify the bounding boxes

3.3.1 Mobilenet

Fitting the model against the Training and Validation datasets

```
Epoch 1/20
144/144 [==============================] - 21s 55ms/step - loss: 1822.3097 - IoU_2: 0.2758 - val_loss: 1754.7238 - val_IoU_2: 0.2640
Epoch 2/20
144/144 [==============================] - 8s 57ms/step - loss: 1442.2554 - IoU_2: 0.2073 - val_loss: 1286.6444 - val_IoU_2: 0.2310
Epoch 3/20
144/144 [==============================] - 7s 52ms/step - loss: 1035.5963 - IoU_2: 0.1585 - val_loss: 961.0732 - val_IoU_2: 0.1413
Epoch 4/20
144/144 [==============================] - 8s 53ms/step - loss: 788.3230 - IoU_2: 0.1598 - val_loss: 757.6865 - val_IoU_2: 0.1776
Epoch 5/20
144/144 [==============================] - 8s 54ms/step - loss: 631.9854 - IoU_2: 0.1838 - val_loss: 625.2685 - val_IoU_2: 0.1836
Epoch 6/20
144/144 [==============================] - 8s 55ms/step - loss: 532.6617 - IoU_2: 0.1834 - val_loss: 539.4155 - val_IoU_2: 0.1741
Epoch 7/20
144/144 [==============================] - 8s 56ms/step - loss: 470.9486 - IoU_2: 0.1749 - val_loss: 485.6057 - val_IoU_2: 0.1645
Epoch 8/20
144/144 [==============================] - 9s 62ms/step - loss: 433.7806 - IoU_2: 0.1610 - val_loss: 451.6979 - val_IoU_2: 0.1521
```

Evaluating against the Validation Set

23/23 [==============================] - 2s 66ms/step - loss: 467.6636 - IoU_2: 0.1795
[467.66357421875, 0.17945216596126556]

Testing some Images to see how the Ground Truth boxes and Predicted boxes are displayed.
Did not get a great IOU% hence Predicted bounding boxes (in YELLOW) is not fitting right over the Ground Truth boxes (in RED)

3.3.2 Densenet121 with Chexnet weights (brucechou1983_CheXNet_Keras_0.3.0_weights.h5)

Ran the same 4500 Lung Opacity Images dataset against this and got almost similar results

Epoch 1/100
86/86 [==============================] - 12s 102ms/step - loss: 125834.2656 - IoU_2: 0.2901 - val_loss: 118614.2344 - val_IoU_2: 0.2570
Epoch 2/100
86/86 [==============================] - 7s 80ms/step - loss: 119315.1484 - IoU_2: 0.2548 - val_loss: 113342.5391 - val_IoU_2: 0.2296
Epoch 3/100
86/86 [==============================] - 5s 62ms/step - loss: 114534.2656 - IoU_2: 0.2445 - val_loss: 109054.4141 - val_IoU_2: 0.2026
Epoch 4/100
86/86 [==============================] - 5s 56ms/step - loss: 110474.1875 - IoU_2: 0.2411 - val_loss: 105287.7344 - val_IoU_2: 0.1670
Epoch 5/100
86/86 [==============================] - 6s 64ms/step - loss: 106838.0859 - IoU_2: 0.2189 - val_loss: 101861.5391 - val_IoU_2: 0.1417
Epoch 6/100
86/86 [==============================] - 6s 66ms/step - loss: 103500.9219 - IoU_2: 0.2235 - val_loss: 98688.0859 - val_IoU_2: 0.1736
Epoch 7/100
86/86 [==============================] - 5s 58ms/step - loss: 100382.0312 - IoU_2: 0.2121 - val_loss: 95704.3281 - val_IoU_2: 0.1616
Epoch 8/100
86/86 [==============================] - 5s 56ms/step - loss: 97441.5156 - IoU_2: 0.2057 - val_loss: 92888.4609 - val_IoU_2: 0.1618
Epoch 9/100
86/86 [==============================] - 5s 57ms/step - loss: 94650.6094 - IoU_2: 0.1941 - val_loss: 90198.6016 - val_IoU_2: 0.2053
Epoch 10/100
86/86 [==============================] - 5s 59ms/step - loss: 91986.6875 - IoU_2: 0.1918 - val_loss: 87619.1641 - val_IoU_2: 0.2104

Evaluating on the Validation set

23/23 [==============================] - 2s 86ms/step - loss: 99378.0312 - IoU_2: 0.2265
[99378.03125, 0.2264869511127472]

3.3.3 Faster RCNN

Faster RCNN using - “faster_rcnn_nas_coco_2018_01_28” from https://download.tensorflow.org/models/object_detection/', created our own labels.pbtxt with just 1 – No Pneumonia and 0 – Pneumonia.

Model input shape

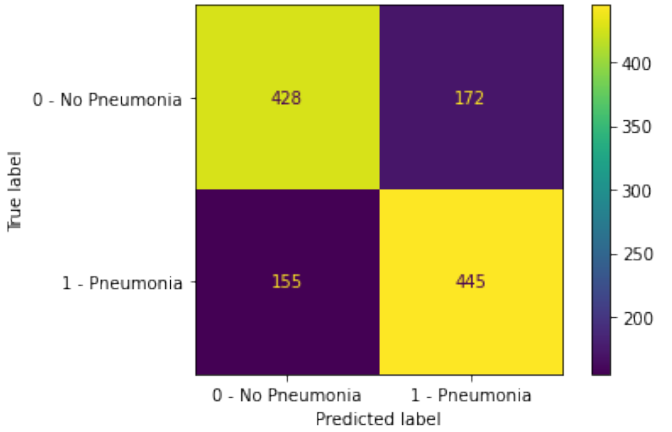
<tf.Tensor 'image_tensor:0' shape=(None, None, None, 3) dtype=uint8>

Model output shapes

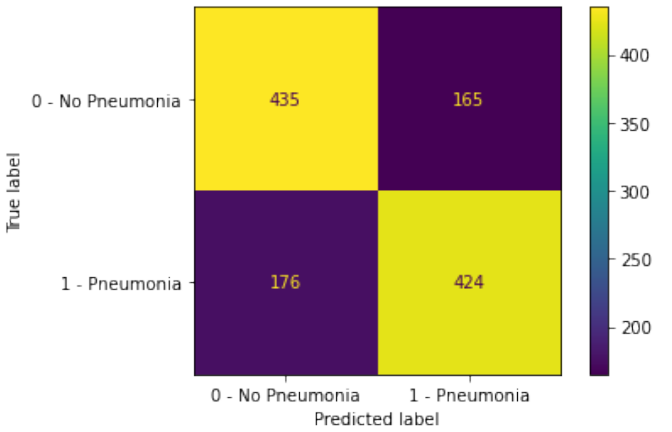
{'detection_scores': TensorShape([None, 100]),
 'detection_boxes': TensorShape([None, 100, 4]),
 'detection_classes': TensorShape([None, 100]),
 'num_detections': TensorShape([None])}

Ran this instantiated model against 40 JPG Images and got the Faster RCNN to identify anchor boxes

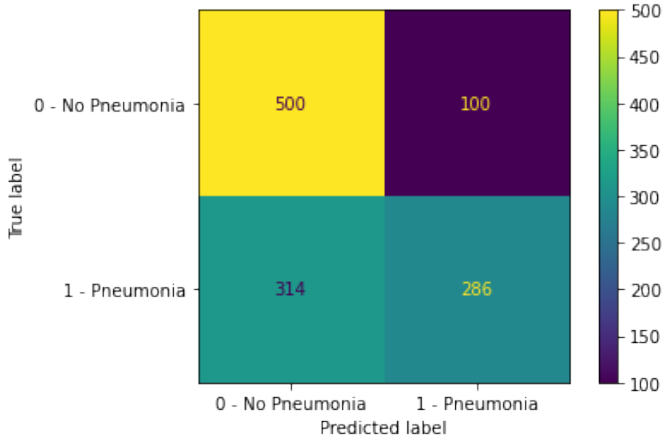
4.0 Model evaluation



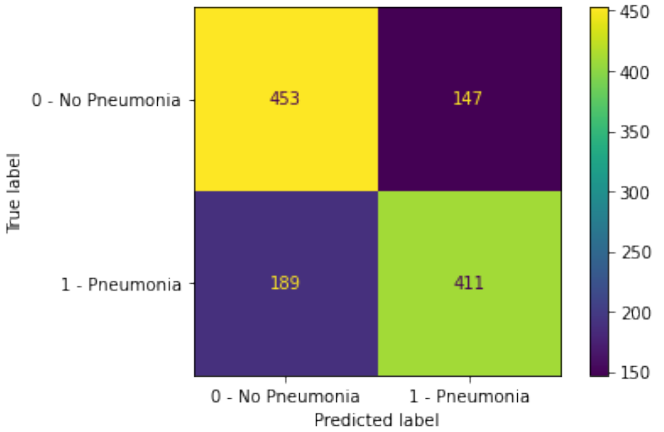
VGG16 best model – Confusion matrix



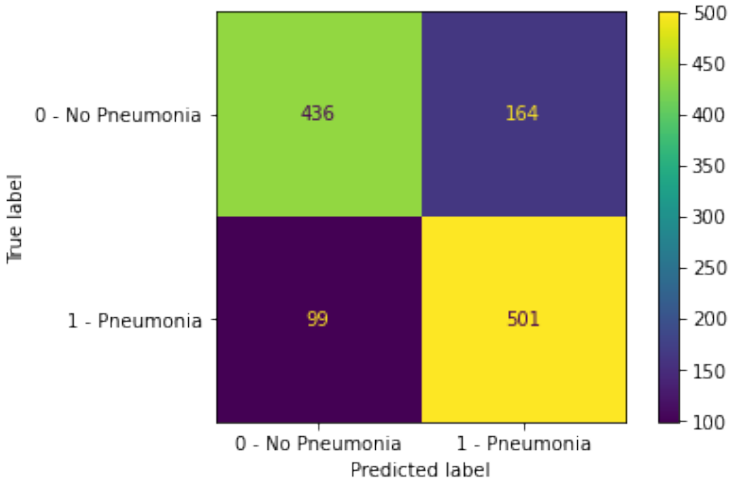
Resnet50 Confusion matrix



InceptionV3 model – Confusion matrix



Densenet121 with Imagenet weights



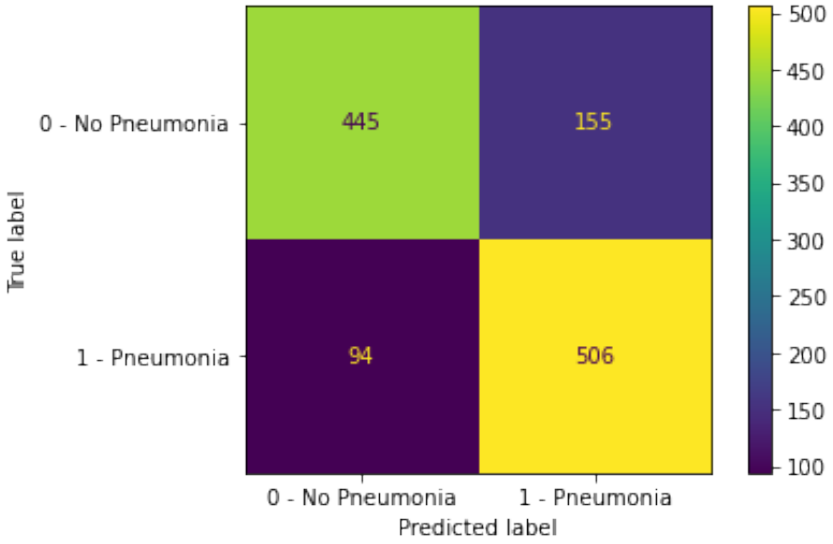
VGG16	Accuracy	0.7275
VGG16	Precision	0.7212
VGG16	Recall	0.7416
VGG16	F1 score	0.7313
VGG16	ROC	0.7275

Resnet50	Accuracy	0.7158
Resnet50	Precision	0.7198
Resnet50	Recall	0.7066
Resnet50	F1 score	0.7132
Resnet50	ROC	0.7158

Inception V3	Accuracy	0.655
Inception V3	Precision	0.7409
Inception V3	Recall	0.4766
Inception V3	F1 score	0.5801
Inception V3	ROC	0.655

Densenet 121 with Imagenet weights	Accuracy	0.7325
Densenet 121 with Imagenet weights	Precision	0.6913
Densenet 121 with Imagenet weights	Recall	0.84
Densenet 121 with Imagenet weights	F1 score	0.7584
Densenet 121 with Imagenet weights	ROC	0.7324

Densenet121 with Chexnet Weights



Densenet121 with Chexnet weights	**Accuracy**	**0.7925**
Densenet121 with Chexnet weights	**Precision**	**0.7655**
Densenet121 with Chexnet weights	**Recall**	**0.8433**
Densenet121 with Chexnet weights	**F1 score**	**0.8025**
Densenet121 with Chexnet weights	**ROC**	**0.7925**

4.1 Object detection using Mobilenet

Epoch 1/20
144/144 [==============================] - 21s 55ms/step - loss: 1822.3097 - IoU_2: 0.2758 - val_loss: 1754.7238 - val_IoU_2: 0.2640
Epoch 2/20
144/144 [==============================] - 8s 57ms/step - loss: 1442.2554 - IoU_2: 0.2073 - val_loss: 1286.6444 - val_IoU_2: 0.2310
Epoch 3/20
144/144 [==============================] - 7s 52ms/step - loss: 1035.5963 - IoU_2: 0.1585 - val_loss: 961.0732 - val_IoU_2: 0.1413
Epoch 4/20
144/144 [==============================] - 8s 53ms/step - loss: 788.3230 - IoU_2: 0.1598 - val_loss: 757.6865 - val_IoU_2: 0.1776
Epoch 5/20
144/144 [==============================] - 8s 54ms/step - loss: 631.9854 - IoU_2: 0.1838 - val_loss: 625.2685 - val_IoU_2: 0.1836
Epoch 6/20
144/144 [==============================] - 8s 55ms/step - loss: 532.6617 - IoU_2: 0.1834 - val_loss: 539.4155 - val_IoU_2: 0.1741
Epoch 7/20
144/144 [==============================] - 8s 56ms/step - loss: 470.9486 - IoU_2: 0.1749 - val_loss: 485.6057 - val_IoU_2: 0.1645
Epoch 8/20
144/144 [==============================] - 9s 62ms/step - loss: 433.7806 - **IoU_2: 0.1610 - val_loss: 451.6979 - val_IoU_2: 0.1521**

Evaluating against the Validation Set

23/23 [==============================] - 2s 66ms/step - loss: 467.6636 - IoU_2: 0.1795 [467.66357421875, **0.17945216596126556**]

4.2 Object detection using Densenet121 with Chexnet weights

Epoch 1/100
86/86 [==============================] - 12s 102ms/step - loss: 125834.2656 - IoU_2: 0.2901 - val_loss: 118614.2344 - val_IoU_2: 0.2570
Epoch 2/100
86/86 [==============================] - 7s 80ms/step - loss: 119315.1484 - IoU_2: 0.2548 - val_loss: 113342.5391 - val_IoU_2: 0.2296
Epoch 3/100
86/86 [==============================] - 5s 62ms/step - loss: 114534.2656 - IoU_2: 0.2445 - val_loss: 109054.4141 - val_IoU_2: 0.2026
Epoch 4/100
86/86 [==============================] - 5s 56ms/step - loss: 110474.1875 - IoU_2: 0.2411 - val_loss: 105287.7344 - val_IoU_2: 0.1670
Epoch 5/100
86/86 [==============================] - 6s 64ms/step - loss: 106838.0859 - IoU_2: 0.2189 - val_loss: 101861.5391 - val_IoU_2: 0.1417
Epoch 6/100
86/86 [==============================] - 6s 66ms/step - loss: 103500.9219 - IoU_2: 0.2235 - val_loss: 98688.0859 - val_IoU_2: 0.1736
Epoch 7/100
86/86 [==============================] - 5s 58ms/step - loss: 100382.0312 - IoU_2: 0.2121 - val_loss: 95704.3281 - val_IoU_2: 0.1616
Epoch 8/100
86/86 [==============================] - 5s 56ms/step - loss: 97441.5156 - IoU_2: 0.2057 - val_loss: 92888.4609 - val_IoU_2: 0.1618
Epoch 9/100
86/86 [==============================] - 5s 57ms/step - loss: 94650.6094 - IoU_2: 0.1941 - val_loss: 90198.6016 - val_IoU_2: 0.2053
Epoch 10/100
86/86 [==============================] - 5s 59ms/step - loss: 91986.6875 - IoU_2: **0.1918 - val_loss: 87619.1641 - val_IoU_2: 0.2104**

Evaluating on the Validation set

23/23 [==============================] - 2s 86ms/step - loss: 99378.0312 - IoU_2: 0.2265 [99378.03125, **0.2264869511127472**]

5.0 Comparison to benchmark

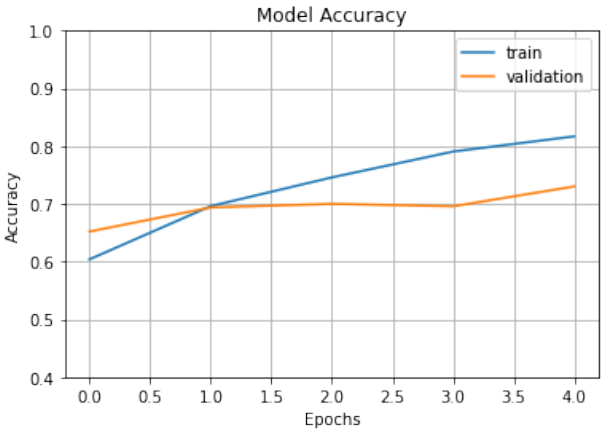
Since se got the best base CNN model accuracy at 72%

a) We expected the best Fine-tuning model to be at least 77% accuracy
b) And expected the Transfer learning pre-trained models to provide at least 85% accuracy

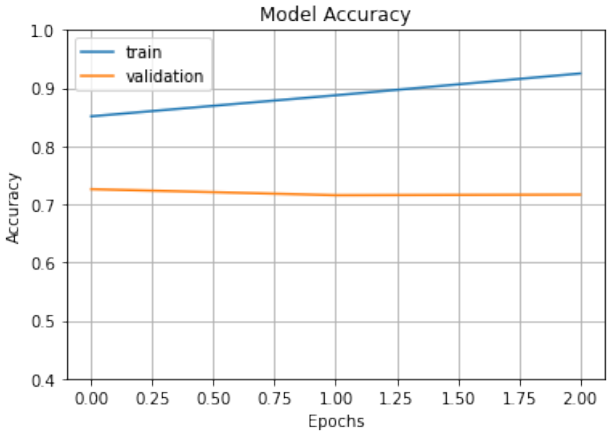
But we were not able to increase the Fine-tuning accuracy beyond 74% and the Pre-trained model accuracy also did not cross 80%.

6.0 Visualizations

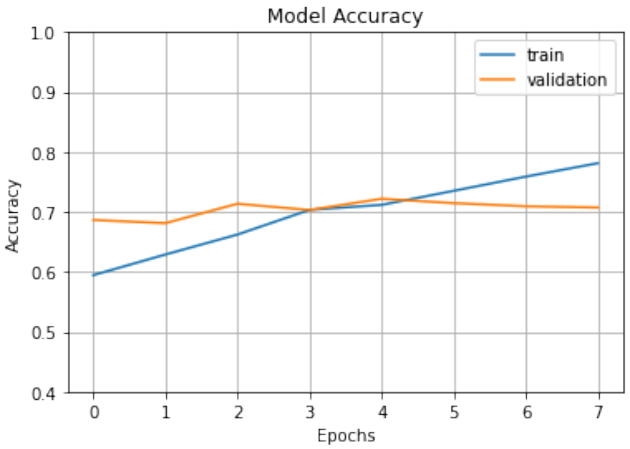
6.1 Fine Tuning Visualizations



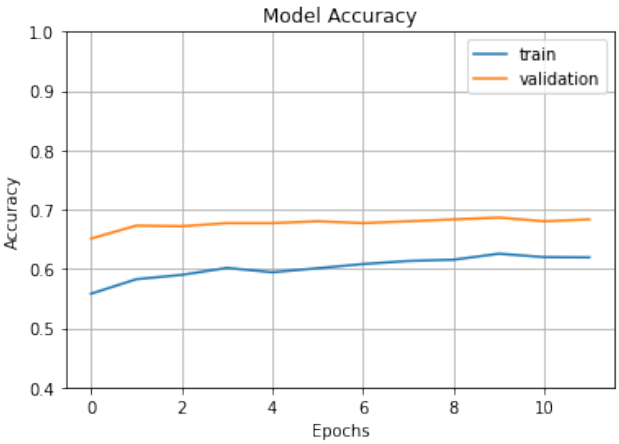
VGG16 model – Best accuracy – 73%



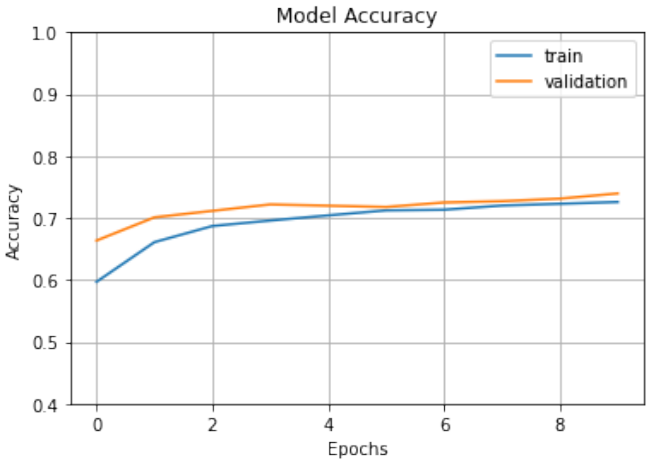
VGG16 model – 71%



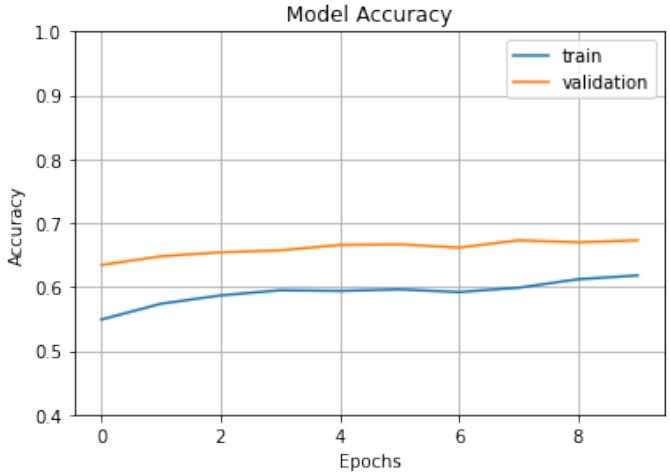
VGG16 model accuracy – 70.5



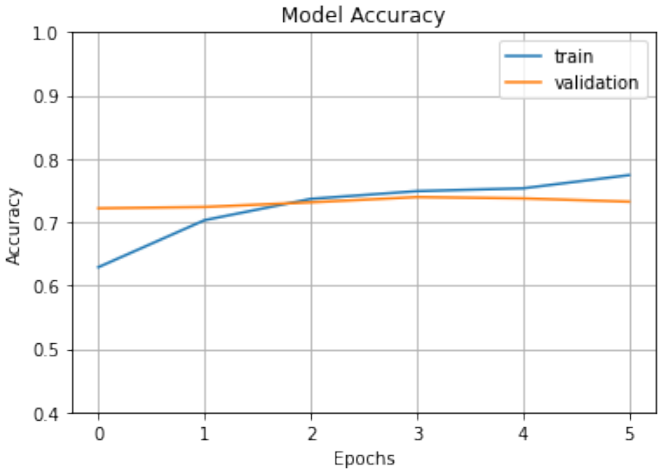
VGG16 model accuracy – 68.%



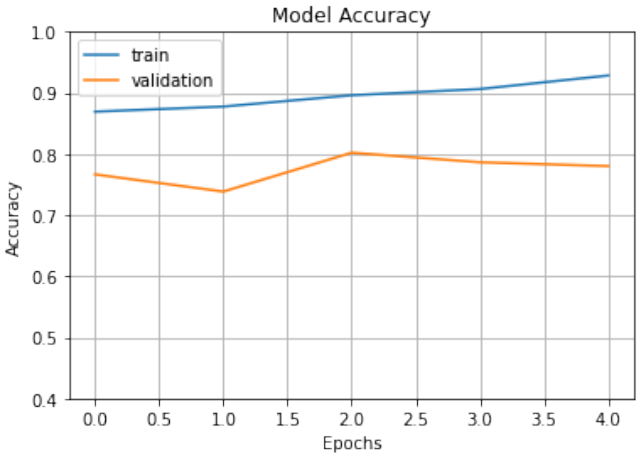
Resnet50 model accuracy – 73.96%

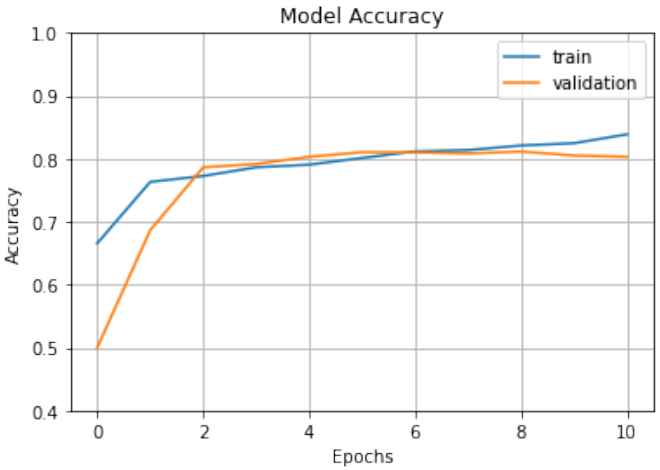


InceptionV3 model accuracy – 67.5%



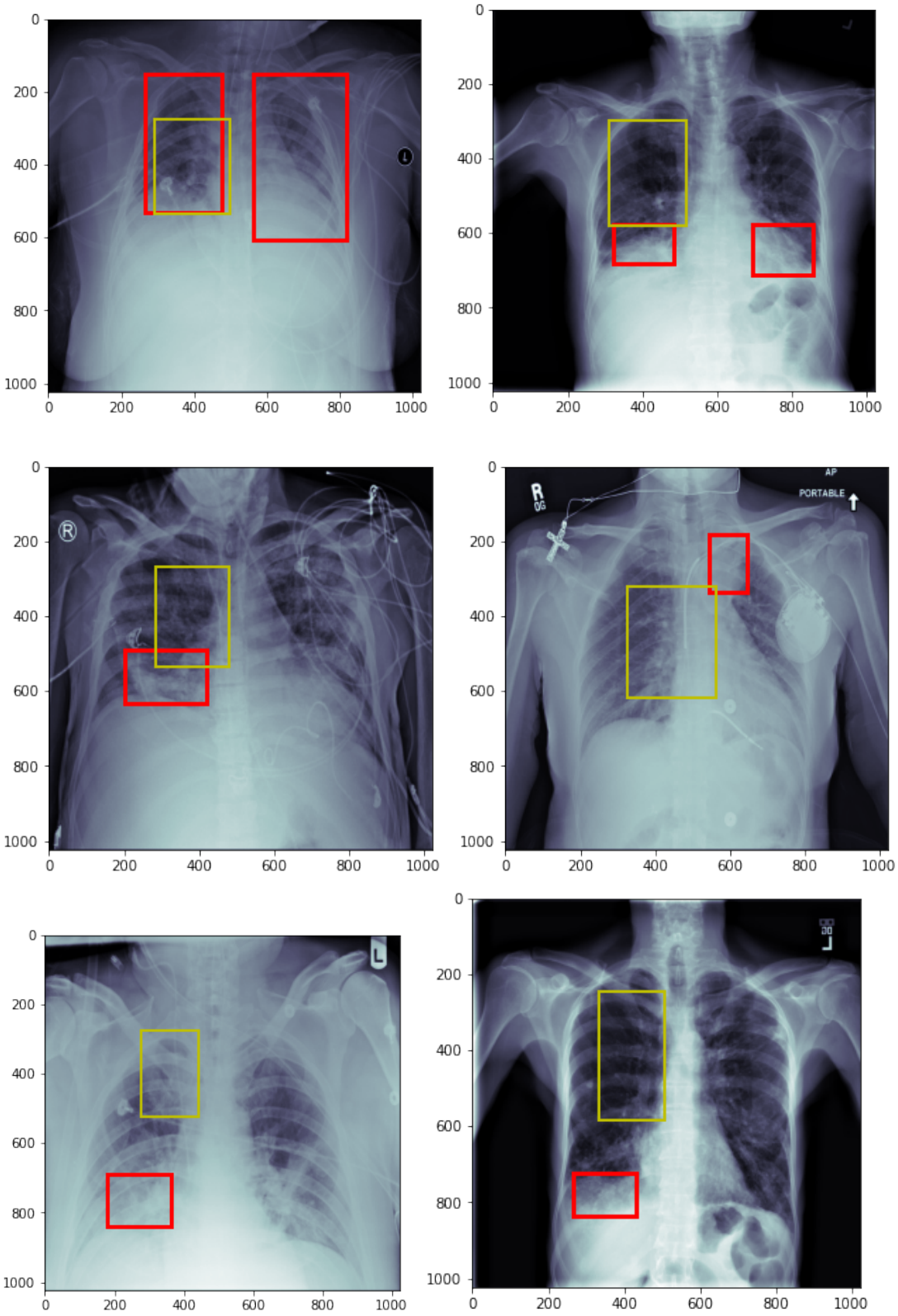
Densenet121 with Imagenet weights – accuracy – 73.9%





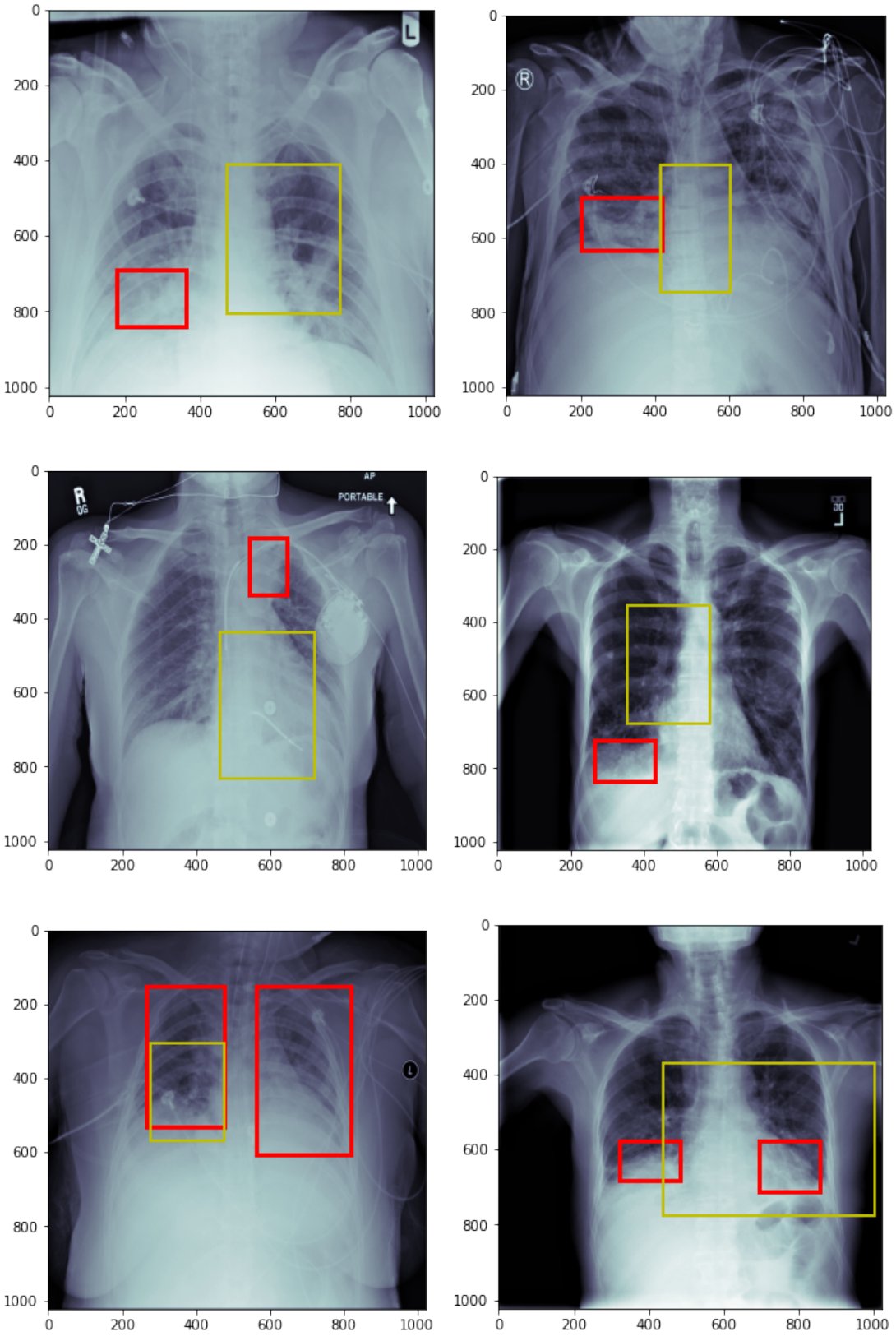
Densenet121 with Chexnet weights accuracy – 78.5% & 80%

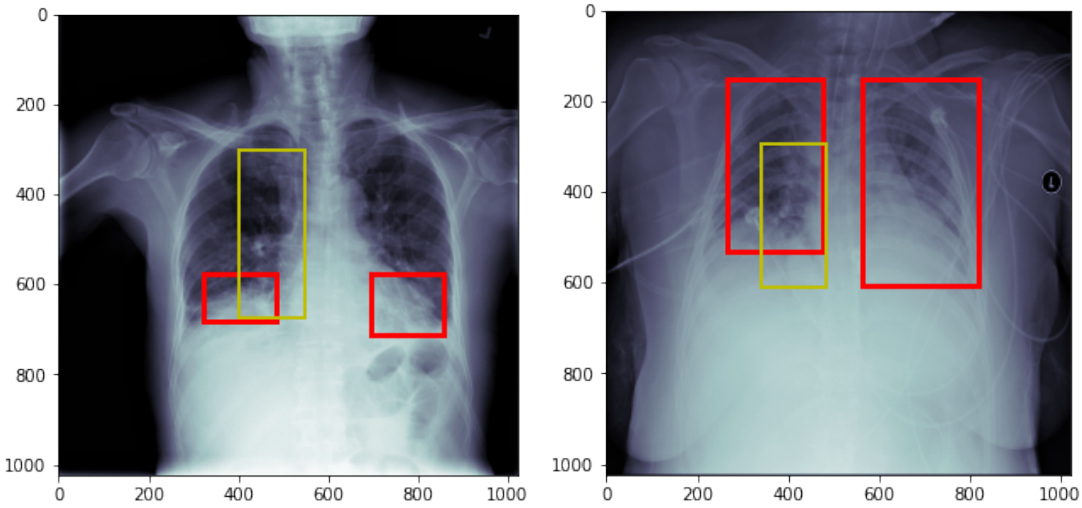
6.2 *Object detection bounding boxes using Mobilenet*



Tried with 1000 Images, 2000 Images and 3000 Images too (Lung Opacity images) and got almost the same results

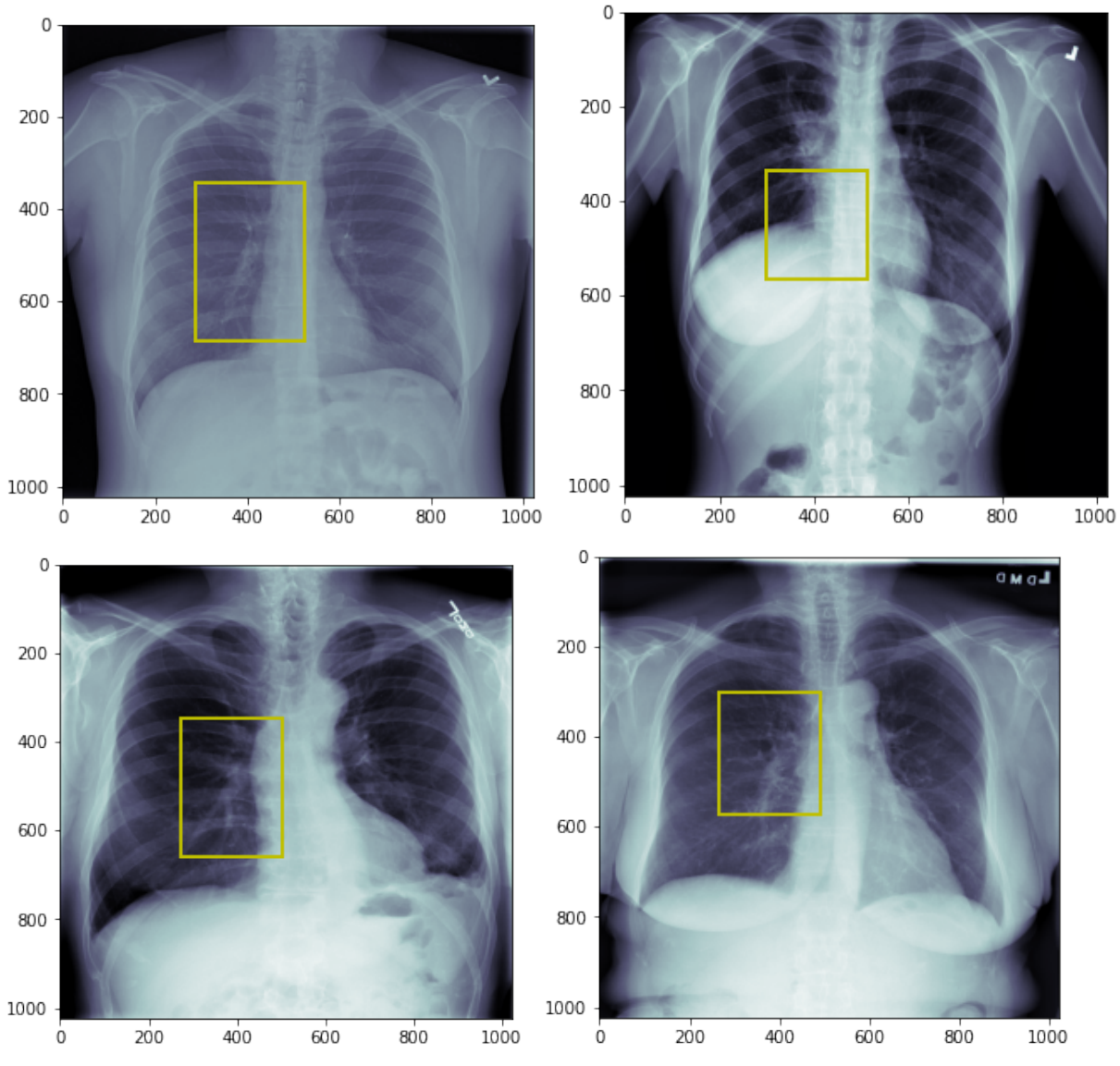
6.3 *Object Detection bounding boxes using Densenet121 with Chexnet Weights*

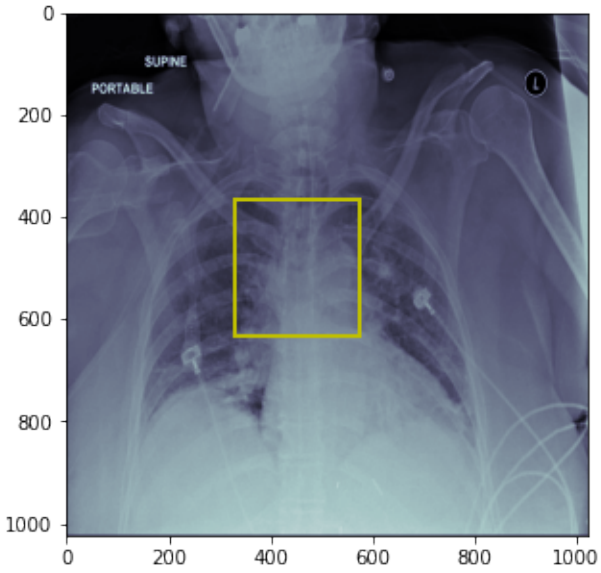




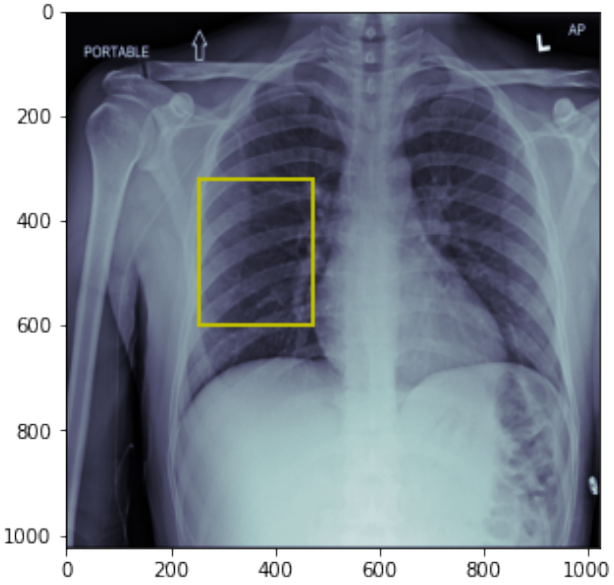
6.4 *Object detection bounding boxes with Random Test Images*

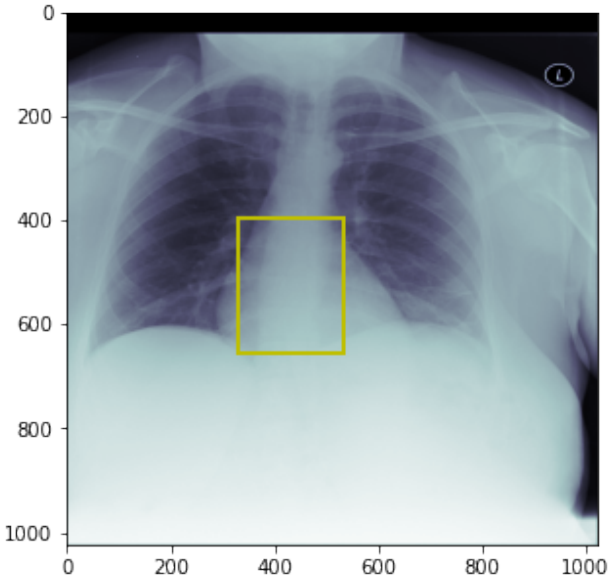
The models were tested against some Random Test images and the predicted bounding boxes are in Yellow color. There is no Red boxes since we don't have any Ground truth information available on these Test Images.

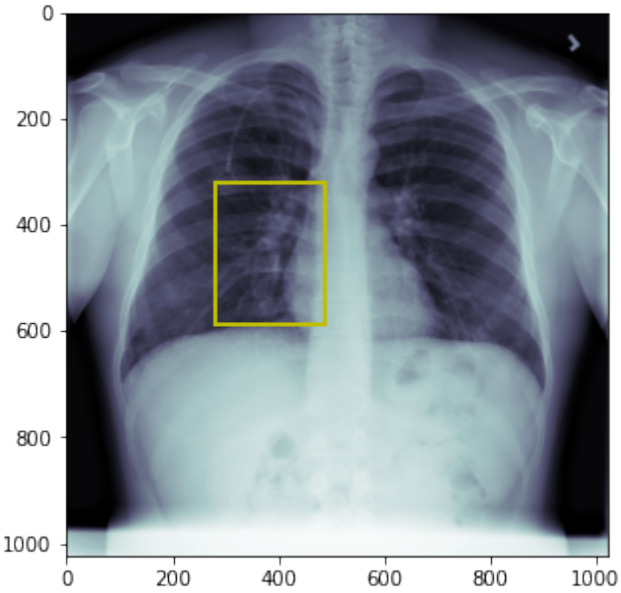


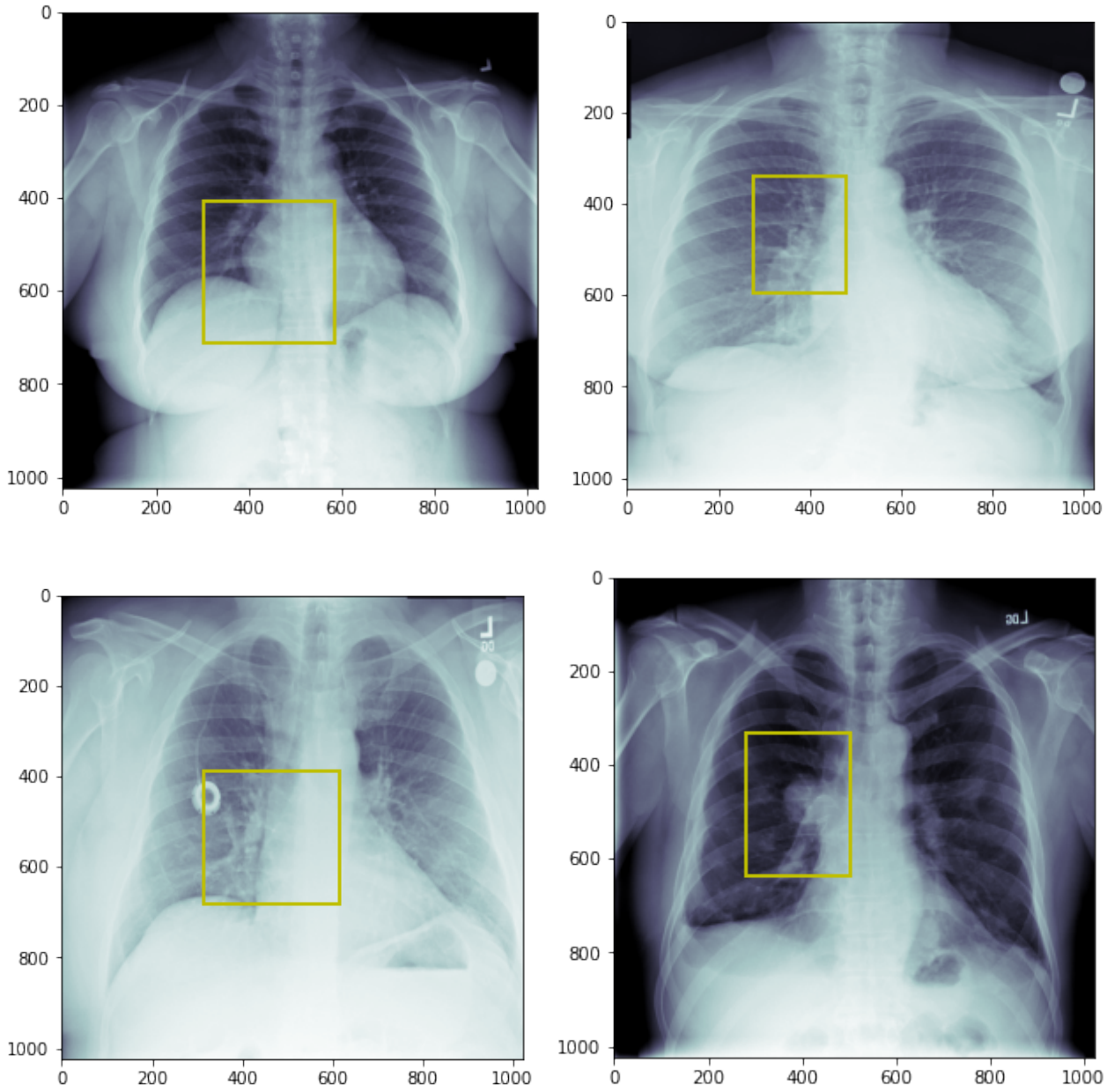

SUPINE
PORTABLE
0
200
400
600
800
1000


0
200
400
600
800
1000


PORTABLE
AP
0
200
400
600
800
1000

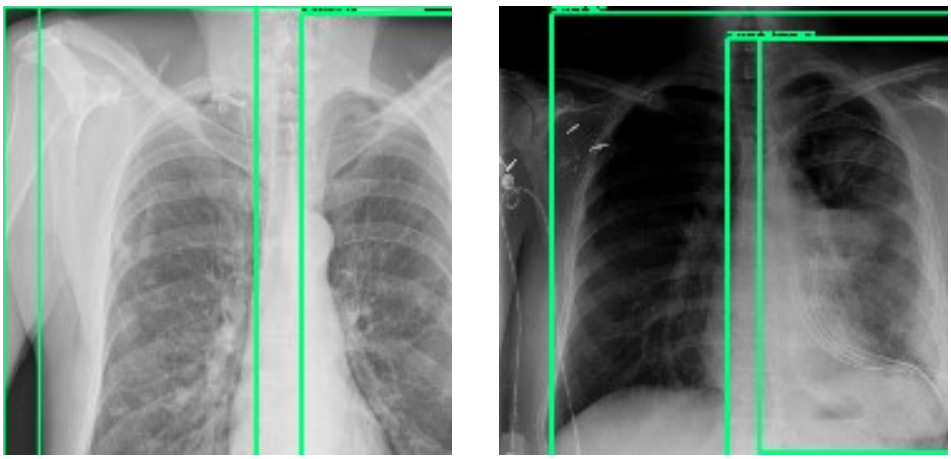

0
200
400
600
800
1000


0
200
400
600
800
1000

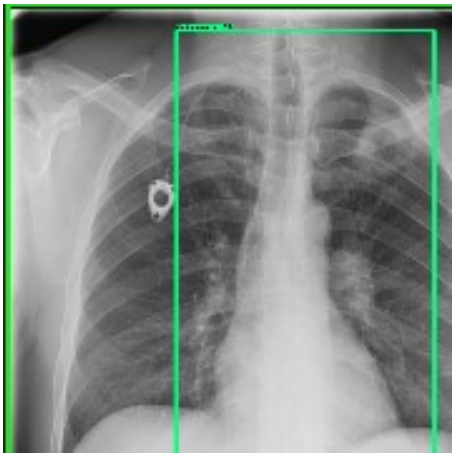


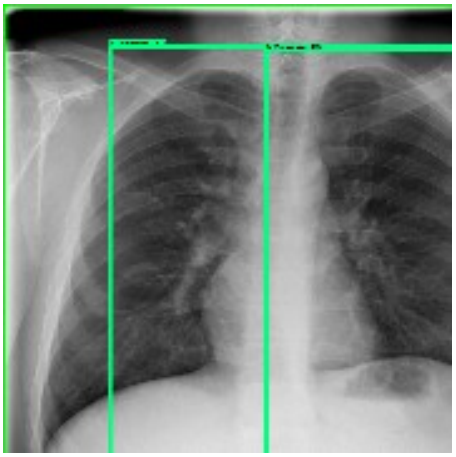
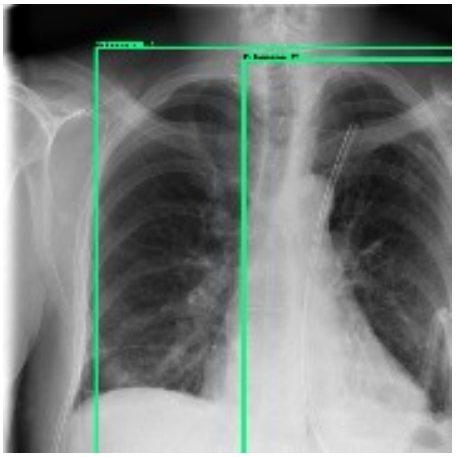
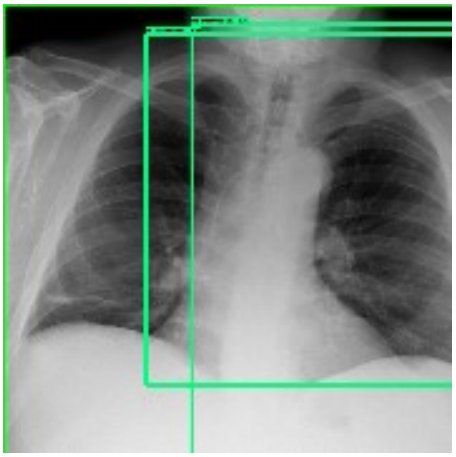
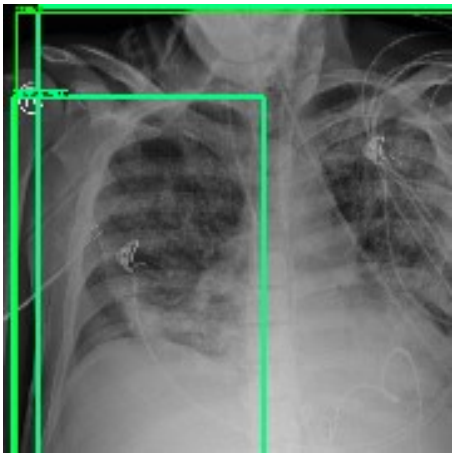
6.5 *Faster_RCNN detected Anchor boxes*

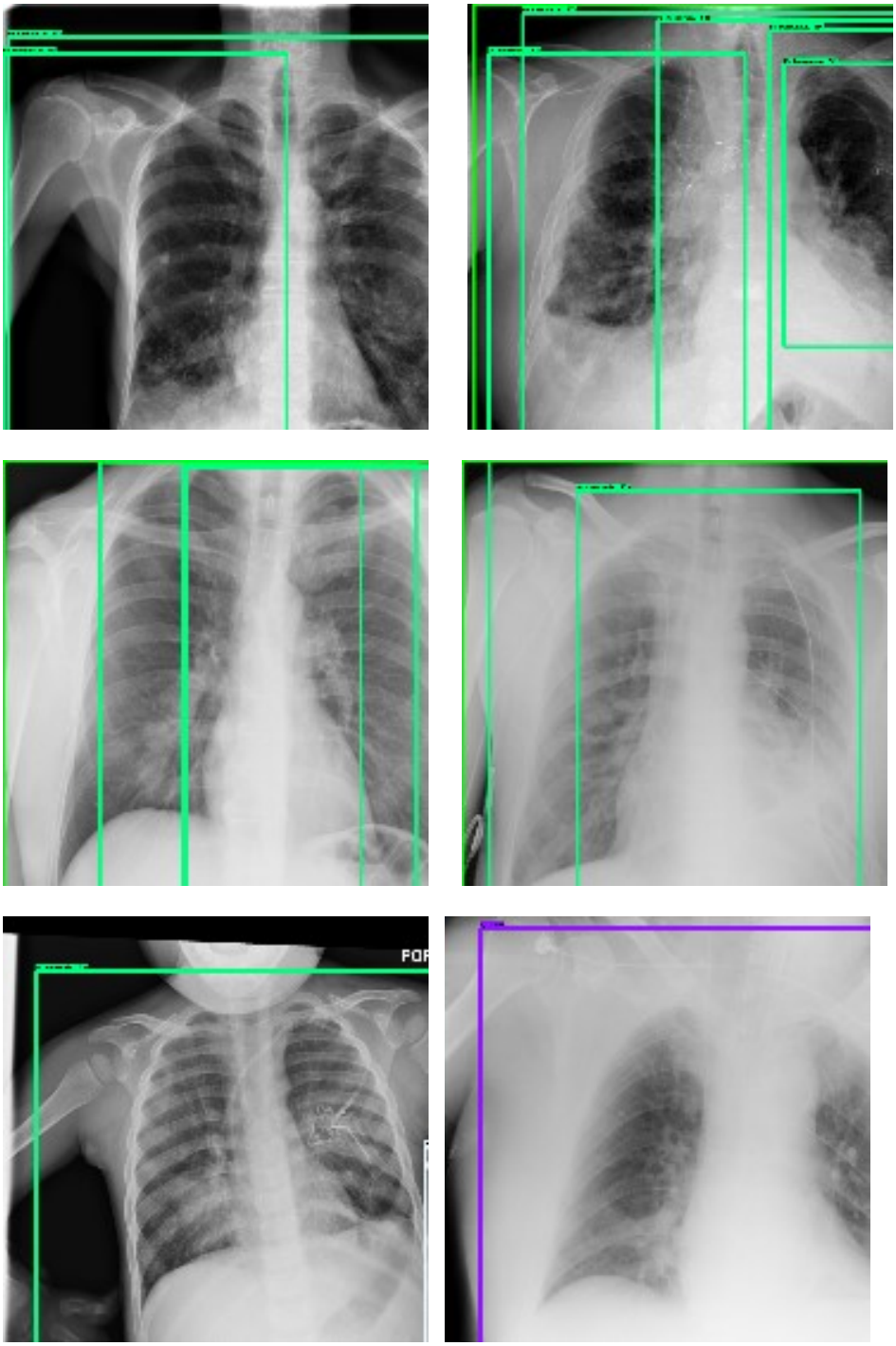
Ran this instantiated model against 40 JPG Images and got the Faster RCNN to identify anchor boxes









7.0 Implications

It is a good idea to use Machine learning / Deep learning algorithms to detect Lung opacities. It will be very beneficial if large number of Digital medical images are collected at a centralized place and run against such Deep learning algorithm. This might help un-earth and predict patterns such as Age group related patterns, gender related patterns, demographic related patterns etc.

These are especially useful during Out-breaks or Pandemics times since there will be too many patients reporting sick at various parts of the state / country and instead of these Digital images being looked at manually by doctors in a de-centralized way it is much better if all these are collected at various primary health centers and sent over to a Centralized location for pattern studies.

During the state of pandemic in recent times, we have observed that virus is taking a newer way to multiply the impact of the virus based on different demographic conditions. In the similar manner,

detecting pneumonia also depends on various demographic conditions, age groups, gender etc. In these scenarios based on symptoms, there is a need to take the X-ray to detect the location of the opacity which confirms whether the patient is suffering from pneumonia or not. In the real time scenarios, AIML has taken its significance to new heights and becoming a boon in all the industries in providing various solutions. Assume a app or software that implements the classification / object detection which helps the patients to have an idea of whether they have pneumonia or not in the emergency cases where doctor is not available on time after getting a X-ray. Based on these, it will make even easier to predecit the opacity blocks on the lungs based on many parameters like age group, sex, demographic conditions thus simplify the predictions during the pandemic scenarios.

8.0 Limitations

We had some computing resource constraints, worked with Google Colab Pro but still had problems in using a huge training dataset, hence had to make do with a restricted number of 6000 Images even though the Training images were about 26684.

If we consider pneumonia or no pneumonia, then there is huge bias towards no pneumonia since the number of images in no pneumonia are 20672 against 6012 pneumonia images which is only 30% of no pneumonia images. Also with the limited system resources, it is always a layback in the model training since it takes more and more time to process the image data which itself is a huge pixel array. Hence we used only 6000 images for creating the model and train. In the later step, we tried to increase the count to 10000 to check for any significant increase in the model performance but as observed there was'nt much of improvement in other models except for model_2a which shown 10% improvement in the accuracy. In the 10000 images 5000 images belong to pneumonia which is about 83.33% of total images and an increase of 67.66% increase over 3000 images from earlier base models. There seems like increasing total images has only increased memory allocation but does not contain more new patterns to learn from the images.

We faced problems in using preprocessing inputs() for the Transfer learning models, found that memory consumption shot up, filled upto the max and would crash Google Colab pro, hence had to ignore the preprocessing input portion.

Probably a preprocessed input might have given us more better accuracies.

Our object detection model ran only against pneumonia / lung opacity images where there are opacity co-ordinates. But due to presence of NaN values in other class of images, object detection model fails to predict the bounding box and model is unable to train.

9.0 Closing Reflection

a. We were able to use a subset of the Training images to design a basic CNN model and create a Classification algorithm to detect X-rays with Pneumonia using about 72% accuracy

b. Further these Basic CNN models were Fine tuned using various Hyper parameters but we were not able to get the accuracies increased beyond 73%

c. But using Pre-trained models as Transfer learning we were able predict Pneumonia classification with a much better 80% accuracy. Densenet121 with Chexnet weights were able to get us to this level.

d. Next we again used Pre-trained models Mobile net and Densenet121 with Chexnet weights to do Object detection of Lung opacity and get bounding boxes.

e. We were able to take Random Training images and predict the object detection bounding boxes along with the pre-provided Ground truth boxes.

f. We were also able to take some Random Testing images from the 3000 images and displayed the predicted bounding boxes on them as Yellow boxes.

g. We were able to run all the 3000 Test images against the Mobile net object detection model and were able to predict opacity co-ordinates.

h. Finally we also ran Faster RCNN and were able to detect Anchor boxes on Randomly selected images.

10. System resources

SYSTEM RESOURCES UTILIZED FOR CAPSTONE PROJECT:
****

PLATFORM - Windows

PLATFORM - RELEASE - 10

PLATFORM - VERSION - 10.0.22621

ARCHITECTURE - AMD64

HOST NAME - MSI

PROCESSOR - Intel64 Family 6 Model 141 Stepping 1, GenuineIntel

RAM - 8 GB

Python Version: 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]

Tensorflow Version: 2.10.0

Keras Version: 2.10.0