Pneumonia Detection by observing Lung Opacity from the Chest X-Ray Images

Interim Report

Content

INTRODUCTION

1.1 What is Pneumonia?

Pneumonia is an infection in one or both lungs. Bacteria, viruses, and fungi cause it. The infection causes inflammation in the air sacs in your lungs, which are called alveoli.

Pneumonia accounts for over 15% of all deaths of children under 5 years old internationally. In 2017, 920,000 children under the age of 5 died from the disease. It requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital signs and laboratory exams.

1.2 What Does a Normal Image Look Like?

This is an illustration of the chest anatomy with the lungs highlighted

It is observed that there is a mass of tissue surrounding the lungs and between the lungs. These areas contain skin, muscles, fat, bones, and also the heart and big blood vessels. That translates into a lot of information on the chest radiograph that is not useful for detecting the lung opacity.

1.3 Chest Radiographs Basics

In the process of taking the image, an X-ray passes through the body and reaches a detector on the other side. Tissues with sparse material, such as lungs which are full of air, do not absorb the X-rays and appear black in the image. Dense tissues such as bones absorb the X-rays and appear white in the image. In short -

- Black = Air

- White = Bone
- Grey = Tissue or Fluid

The left side of the subject is on the right side of the screen by convention. It can also be observed that there is a small L at the top of the right corner. In a normal image we see the lungs as black, but they have different projections on them - mainly the rib cage bones, main airways, blood vessels and the heart.

PROBLEM STATEMENT, DATA AND FINDINGS

2.1 Problem statement

Pneumonia usually manifests as an area or areas of increased opacity on CXR. However, the diagnosis of pneumonia on CXR is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, or post-radiation or surgical changes. Outside of the lungs, fluid in the pleural space (pleural effusion) also appears as increased opacity on CXR.

So lung opacities do not literally point to pneumonia, but there can be many other issues or abnormalities as well. When available, comparison of CXRs of the patient taken at different time points and correlation with clinical symptoms and history are helpful in making the diagnosis.

CXRs are the most commonly performed diagnostic imaging study. A number of factors such as positioning of the patient and depth of inspiration can alter the appearance of the CXR, complicating interpretation further. In addition, clinicians are faced with reading high volumes of images every shift.

2.2 Business Domain Value

Automating Pneumonia screening in chest radiographs, providing affected area details through bounding box. Assist physicians to make better clinical decisions or even replace human judgement in certain functional areas of healthcare (eg, radiology).

With the help of AI techniques and also by relevant clinical support, we can unlock clinically relevant information hidden in the massive amount of data, which in turn can assist clinical decision making.

2.3 Objective of the work

- To build a model to identify whether CXR images have lung opacity or not.
- To build an algorithm to automatically locate lung opacities on chest radiographs.

2.4 About the given Data Set

2.4.1 The initial set of data

Medical images are stored in special format called DICOM files (*.dcm). They contain a combination of header, metadata as well as underlaying raw image arrays for pixel data.

The data and datasets are downloaded from https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data

The data and the dataset contains:

1) **stage_2_detailed_class_info.csv** – contains attributes patientId and class information
2) **stage_2_train_labels.csv** – contains attributes patientId, x, y, width, height and Target
3) **stage_2_train_images** – 22684 images in .dcm format, to be used for training the model
4) **stage_2_test_images** – 3000 images in .dcm format.

2.4.2 Details about the two given csv files

Sl.No	File Name	Total no. of information's (rows)	No. of Attritubes	Attributes
1	stage_2_detailed_class_info.csv	30227	2	patientId, Class
2	stage_2_train_labels.csv	30227	6	patientId, x, y, width, height, Target

2.4.3 Details about the given images

Training Images	:	26,684 images and supported by two csv files
Testing Images	:	3,000 images
Observation	:	All the images are named with the patient-id

- **The testing images doesn't contain any information. So the training images should be splitted into training data and validation data in the ratio of 70:30**

2.5 Findings from the given dataset

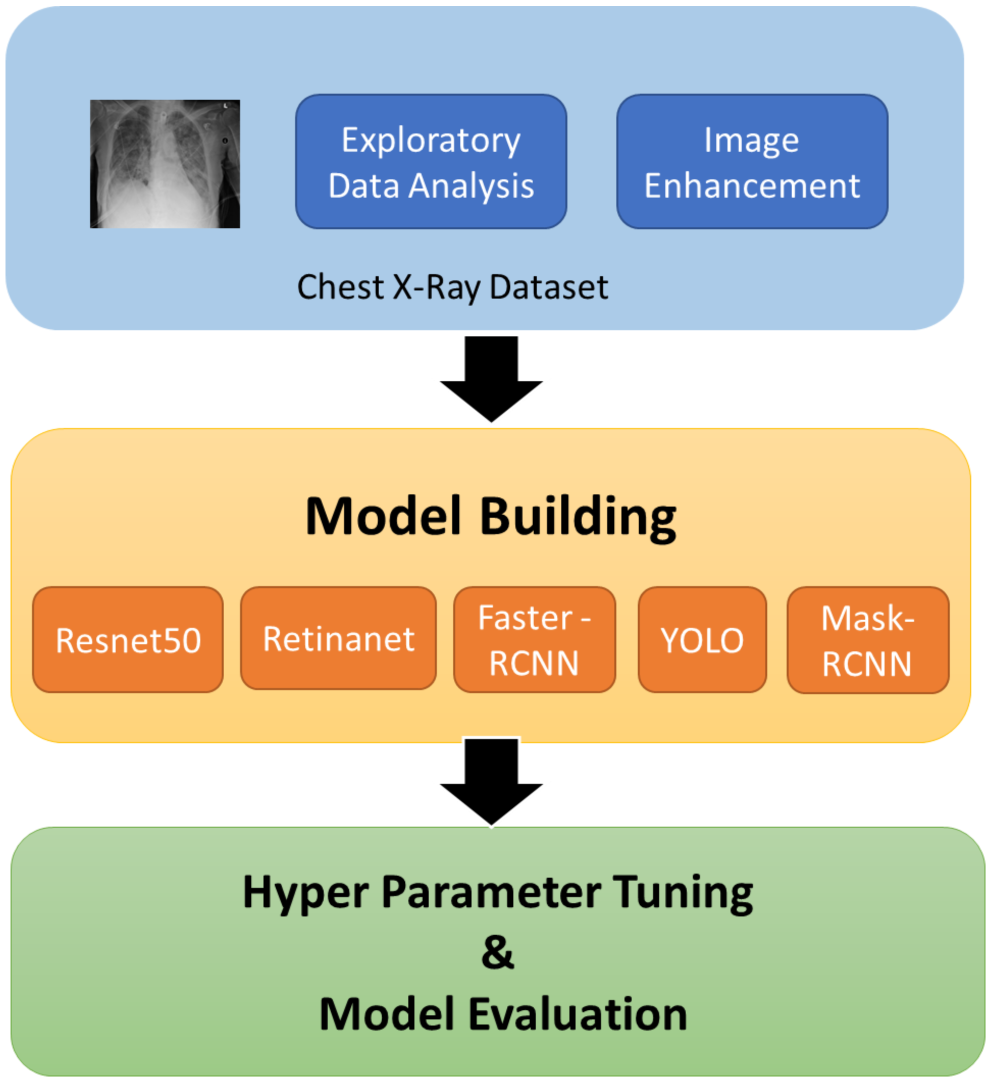
Differences observed between training images and no. of information's given in the .csv files

• No of training images	:	26,684
• No. of rows given in csv file (No. of patientId)	:	30,227
• Differences Observed	:	3,543

SUMMARY OF THE APPROACH

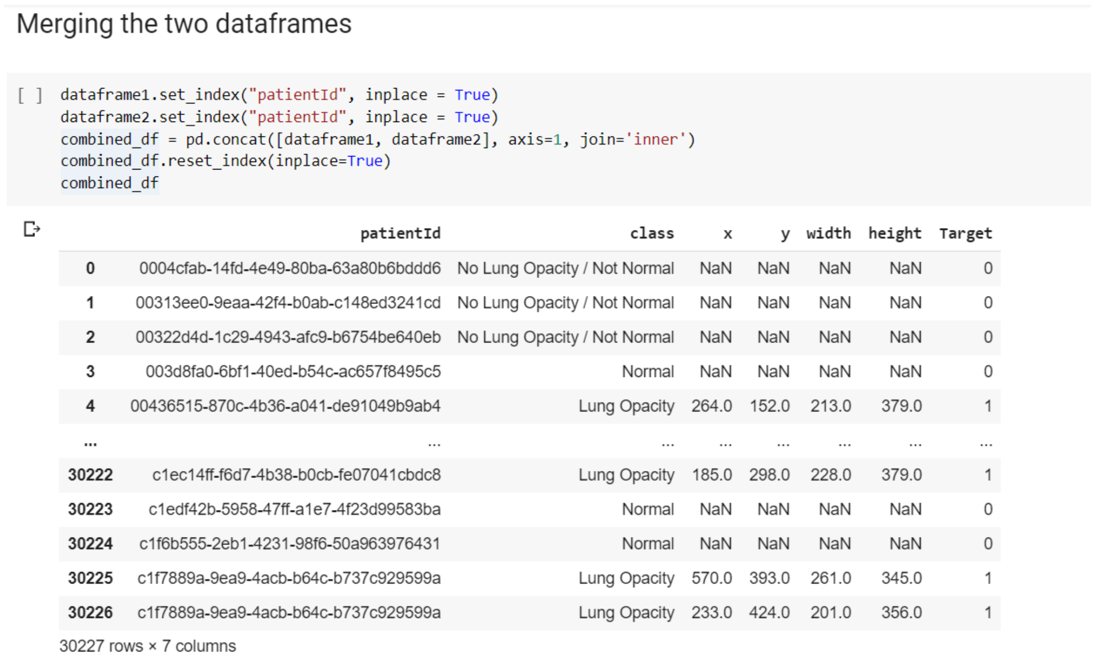
3.1 Workflow

- Exploratory Data Analysis
- Image Pre-processing
- Model Building
- Hyper parameter tuning and Model Evaluation



EXPLORATORY DATA ANALYSIS

4.1 Findings from the given two csv files after concatenating



Merging the two dataframes

```
[ ] dataframe1.set_index("patientId", inplace = True)
    dataframe2.set_index("patientId", inplace = True)
    combined_df = pd.concat([dataframe1, dataframe2], axis=1, join='inner')
    combined_df.reset_index(inplace=True)
    combined_df
```

```
                                   patientId                        class      x      y  width  height  Target
0      0004cfab-14fd-4e49-80ba-63a80b6bddd6  No Lung Opacity / Not Normal    NaN    NaN    NaN     NaN       0
1      00313ee0-9eaa-42f4-b0ab-c148ed3241cd  No Lung Opacity / Not Normal    NaN    NaN    NaN     NaN       0
2      00322d4d-1c29-4943-afc9-b6754be640eb  No Lung Opacity / Not Normal    NaN    NaN    NaN     NaN       0
3      003d8fa0-6bf1-40ed-b54c-ac657f8495c5                        Normal    NaN    NaN    NaN     NaN       0
4      00436515-870c-4b36-a041-de91049b9ab4                  Lung Opacity  264.0  152.0  213.0   379.0       1
...                                     ...                           ...    ...    ...    ...     ...     ...
30222  c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8                  Lung Opacity  185.0  298.0  228.0   379.0       1
30223  c1edf42b-5958-47ff-a1e7-4f23d99583ba                        Normal    NaN    NaN    NaN     NaN       0
30224  c1f6b555-2eb1-4231-98f6-50a963976431                        Normal    NaN    NaN    NaN     NaN       0
30225  c1f7889a-9ea9-4acb-b64c-b737c929599a                  Lung Opacity  570.0  393.0  261.0   345.0       1
30226  c1f7889a-9ea9-4acb-b64c-b737c929599a                  Lung Opacity  233.0  424.0  201.0   356.0       1
30227 rows × 7 columns
```

- **The dataframe contains 30227 informations**
- **Attributes: class, x, y, width, height and Target**

4.1.1 Information about the missing values

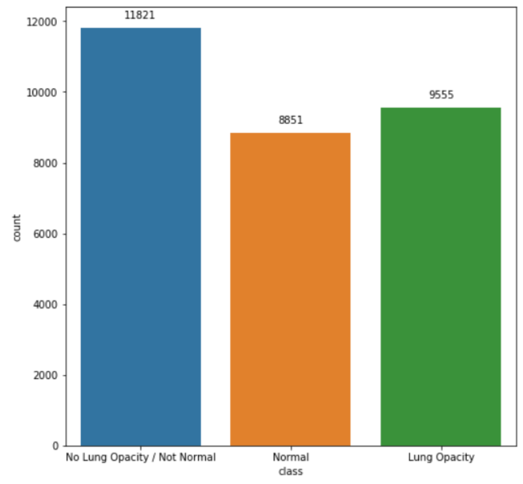
Attributes	Non-Null Count	Null Values count	Dtype
patientId	**30227**	**0**	**object**
class	**30227**	**0**	**object**
x	**9555**	**20672**	**float64**
y	**9555**	**20672**	**float64**
width	**9555**	**20672**	**float64**
height	**9555**	**20672**	**float64**
Target	**30227**	**0**	**int64**

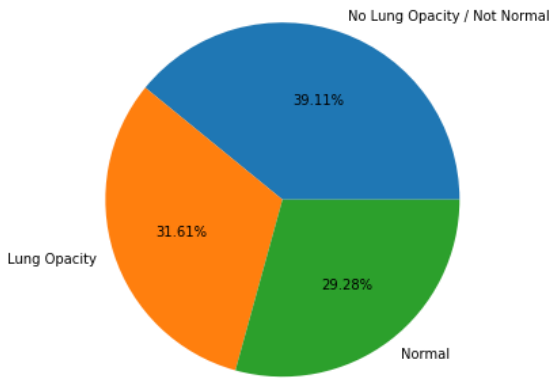
- **No missing values observed in patientId, class and Target**
- **Equal no. of missing values observed in the columns x, y, width and height**

4.1.2 Exploring the attribute "class"

- Three different classes of images were observed
    - No Lung Opacity/ Not Normal – 11821 information
    - Lung Opacity – 9555 information
    - Normal – 8851 information
- All the three classes have almost equal distributions

```
No Lung Opacity / Not Normal    11821
Lung Opacity                     9555
Normal                           8851
Name: class, dtype: int64
```

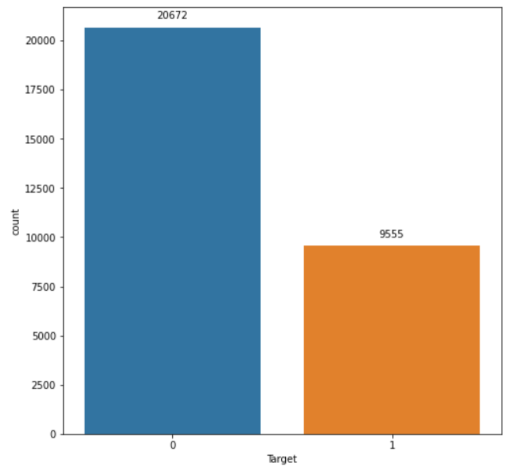


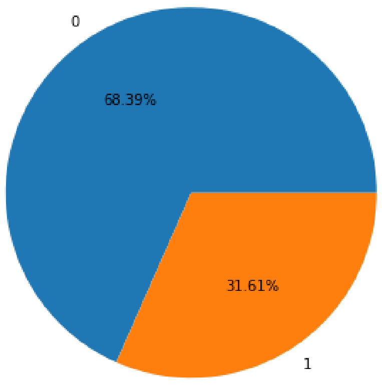


4.1.3 Exploring the attribute "Target"

- Two different targets of images were observed
    - Target= 0 –20672 information
    - Target=1 – 9555 information
- The given information is more biased towards Target=0

```
0    20672
1     9555
Name: Target, dtype: int64
```

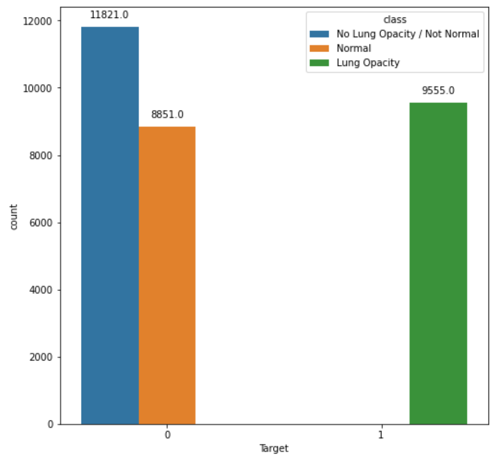




4.1.4 Relationship between "Target" and "class"

- Target = 0 includes two classes " No Lung Opacity/ Not Normal" and "Normal"
- Class "Lung Opacity" is categorized as Target=1

Target : 0 20672 counts		Target: 1 9555 counts
Class: No Lung Opacity/ Not Normal 11821 counts	Class: Normal 8851 counts	Class: Lung Opacity 9555 counts



4.1.5 Analyzing the target attribute separately: Target=0

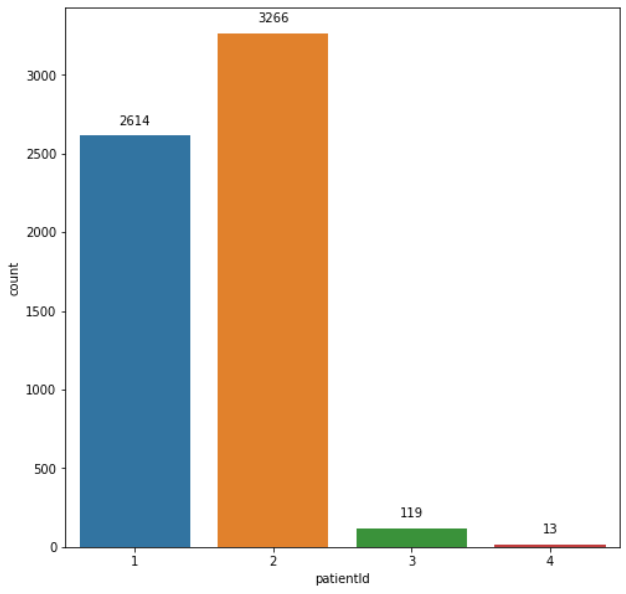
- Total no.of Unique patientId: 20672
- Each patientId contains only one sets of information
- When Target = 0 ➔ Bounding box values are not available [x, y, width, height]

Column	Non-Null Count	Null Values count	Dtype
patientId	20672	0	object
class	20672	0	object
x	0	20672	float64
y	0	20672	float64
width	0	20672	float64
height	0	20672	float64
Target	20672	0	int64

4.1.6 Analyzing the target attribute separately: Target=1

- Total no.of Unique patientId: 6012
- Each patientId contains one and more than one informations (upto 4)
- When Target = 1 ➔ No missing values observed

Column	Non-Null Count	Null Values count	Dtype
patientId	**9555**	**0**	**object**
class	**9555**	**0**	**object**
x	**9555**	**0**	**float64**
y	**9555**	**0**	**float64**
width	**9555**	**0**	**float64**
height	**9555**	**0**	**float64**
Target	**9555**	**0**	**int64**

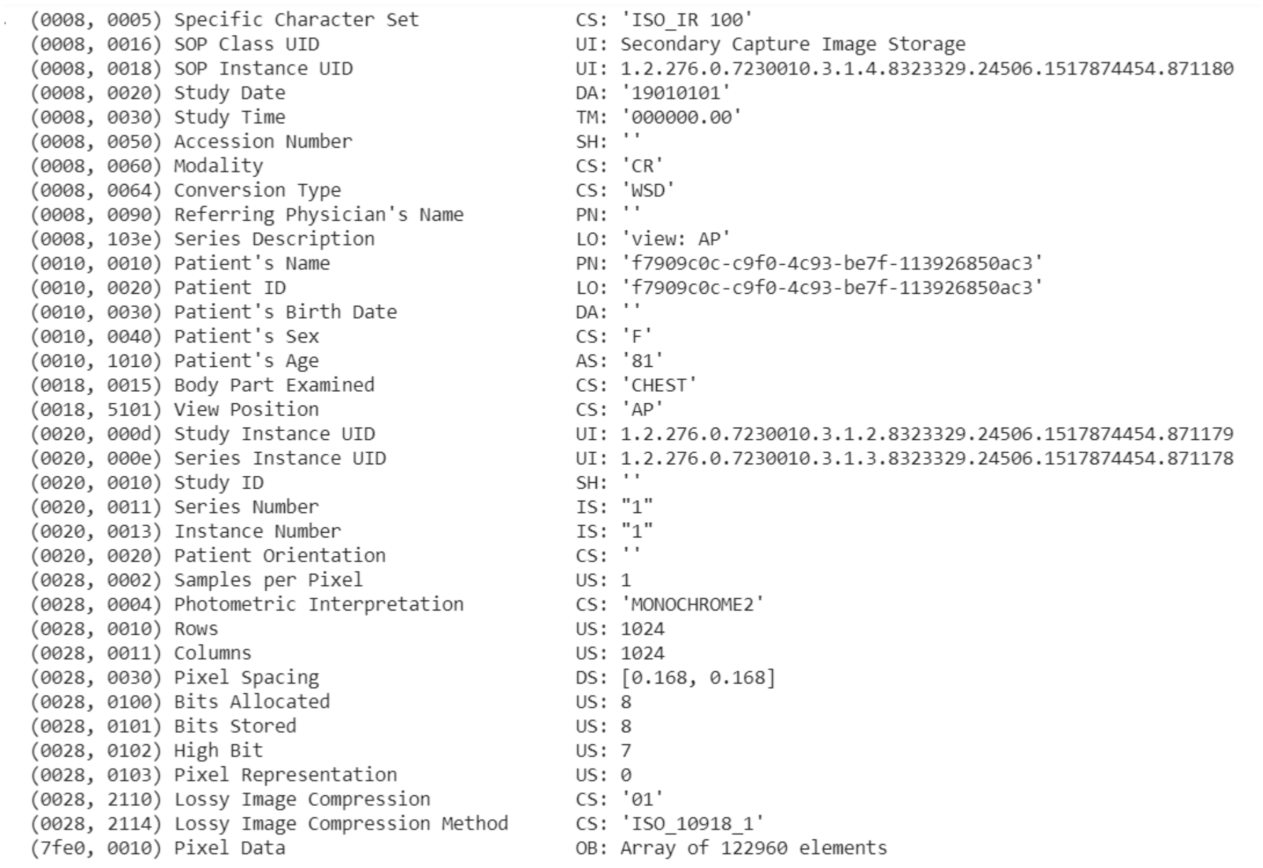


- PatientId with only one sets of bounding box information – 2614
- PatientId with two sets of bounding box information – 3266
- PatientId with three sets of bounding box information – 119
- PatientId with four sets of bounding box information – 13

- Total no unique patient Id - 6012

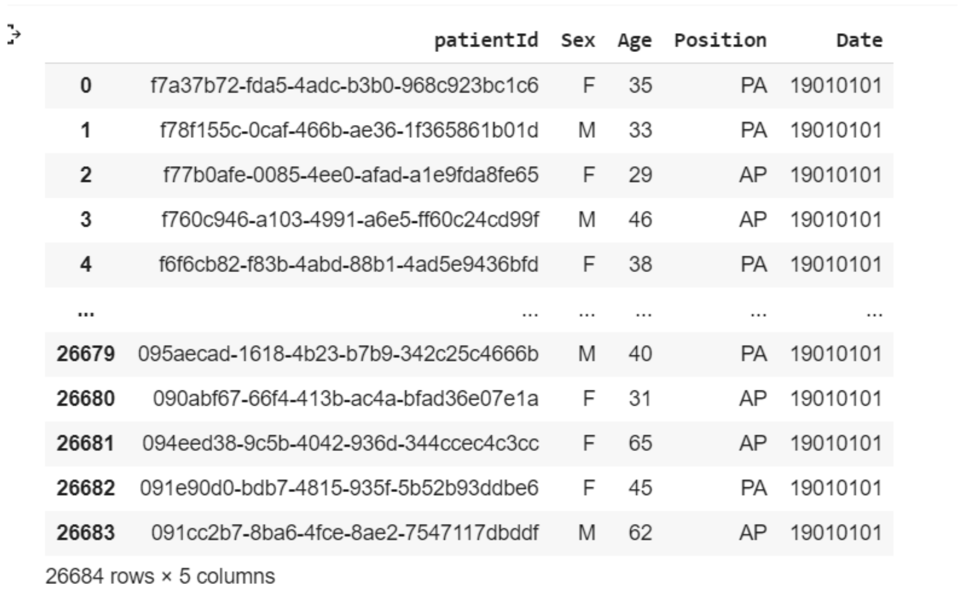
4.2 Exploring the metadata information

The given dicom files contains the following metadata information



```
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.24506.1517874454.871180
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: AP'
(0010, 0010) Patient's Name                      PN: 'f7909c0c-c9f0-4c93-be7f-113926850ac3'
(0010, 0020) Patient ID                          LO: 'f7909c0c-c9f0-4c93-be7f-113926850ac3'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '81'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'AP'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.24506.1517874454.871179
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.24506.1517874454.871178
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.168, 0.168]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 122960 elements
```

- All the given images have been of same size (1024, 1024)
- The Patient's Age, Patient's Sex, View Position, Study Date is extracted from the dicom file for analysis



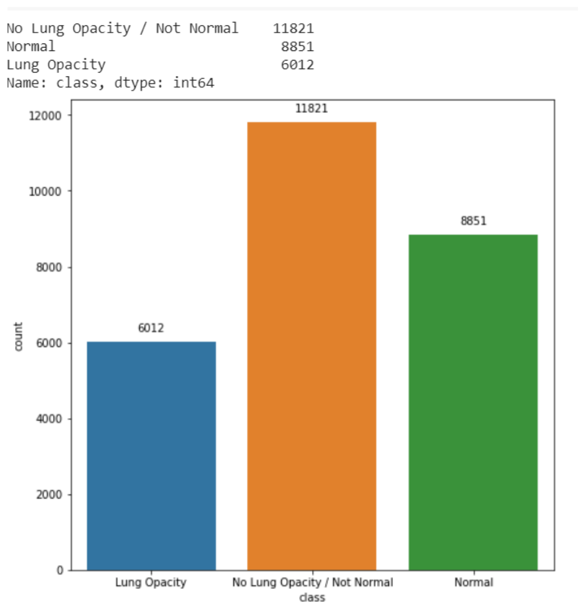
	patientId	Sex	Age	Position	Date
0	f7a37b72-fda5-4adc-b3b0-968c923bc1c6	F	35	PA	19010101
1	f78f155c-0caf-466b-ae36-1f365861b01d	M	33	PA	19010101
2	f77b0afe-0085-4ee0-afad-a1e9fda8fe65	F	29	AP	19010101
3	f760c946-a103-4991-a6e5-ff60c24cd99f	M	46	AP	19010101
4	f6f6cb82-f83b-4abd-88b1-4ad5e9436bfd	F	38	PA	19010101
...	...	...	...	...	...
26679	095aecad-1618-4b23-b7b9-342c25c4666b	M	40	PA	19010101
26680	090abf67-66f4-413b-ac4a-bfad36e07e1a	F	31	AP	19010101
26681	094eed38-9c5b-4042-936d-344ccec4c3cc	F	65	AP	19010101
26682	091e90d0-bdb7-4815-935f-5b52b93ddbe6	F	45	PA	19010101
26683	091cc2b7-8ba6-4fce-8ae2-7547117dbddf	M	62	AP	19010101

26684 rows × 5 columns

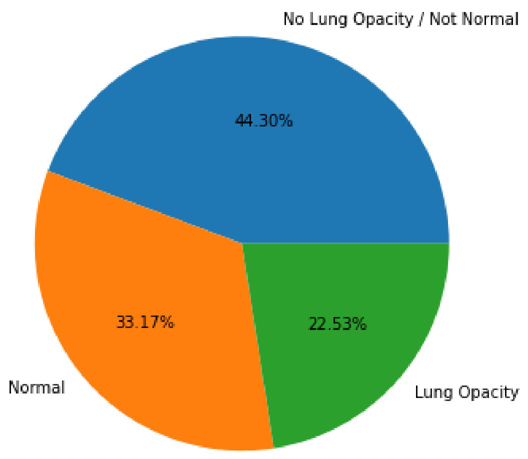
- It is observed that no.of images provided is 26684 and no.of unique patient id is 26684.
- So each patient have one cxr image. Mutiple chest X-ray for a single patient is not available. So ignoring the Study Date from the analysis.

4.2.1 Exploring the attribute “class” from the metadata

- ➢ Three different classes of images were observed
    - o No Lung Opacity/ Not Normal – 11821 information
    - o Lung Opacity – 6012 information
    - o Normal – 8851 information
- ➢ All the three classes have almost equal distributions



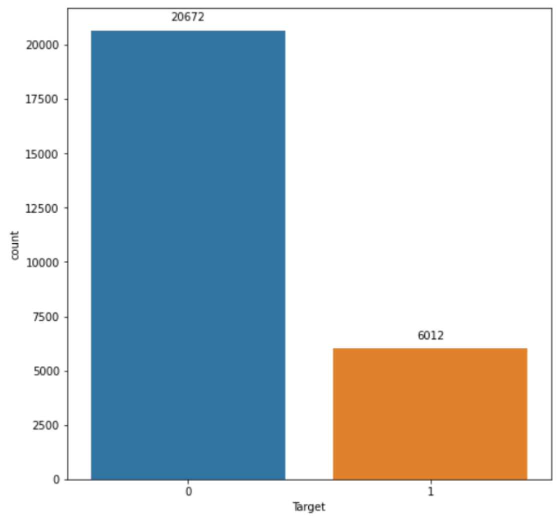
```
No Lung Opacity / Not Normal    11821
Normal                           8851
Lung Opacity                     6012
Name: class, dtype: int64
```

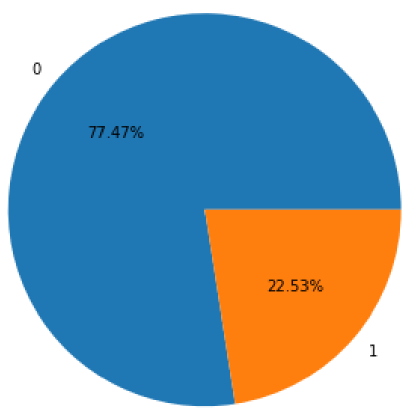


4.2.2 Exploring the attribute “Target” from the metadata

- ➢ Two different targets of images were observed
    - o Target= 0 –20672 information
    - o Target=1 – 6012 information
- ➢ The given information is more biased towards Target=0

```
0    20672
1     6012
Name: Target, dtype: int64
```

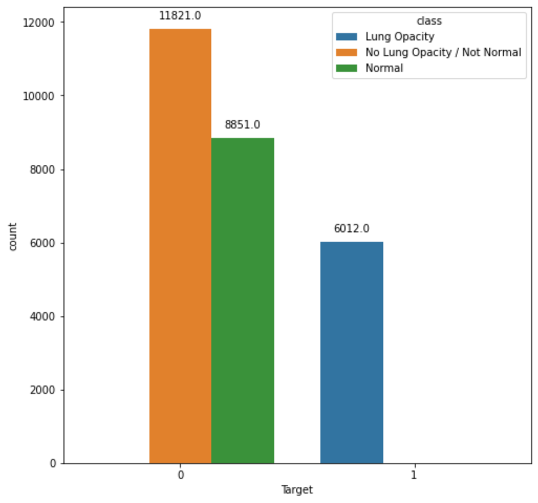




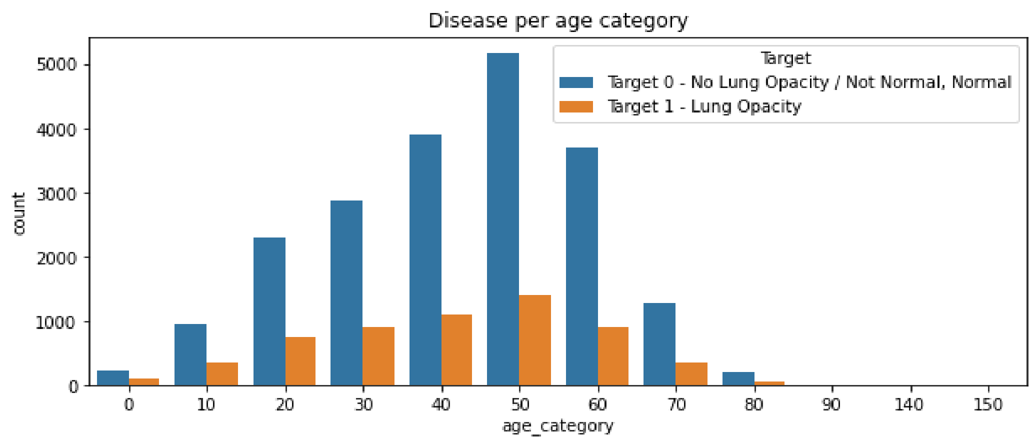
4.2.3 Relationship between “Target” and “class”

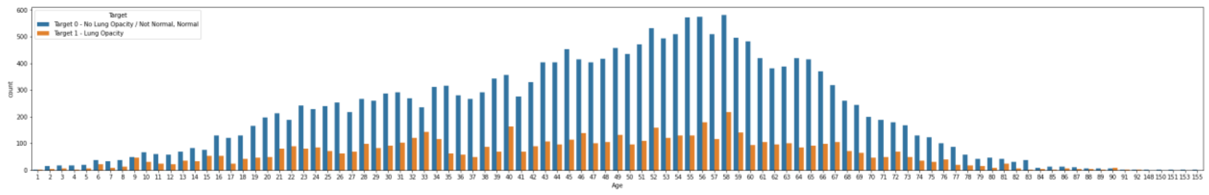
- Target = 0 includes two classes “ No Lung Opacity/ Not Normal” and “Normal”
- Class “Lung Opacity” is categorized as Target=1

Target : 0 **20672 counts**		**Target: 1** **9555 counts**
Class: No Lung Opacity/ Not Normal 11821 counts	**Class: Normal 8851 counts**	**Class: Lung Opacity 6012 counts**



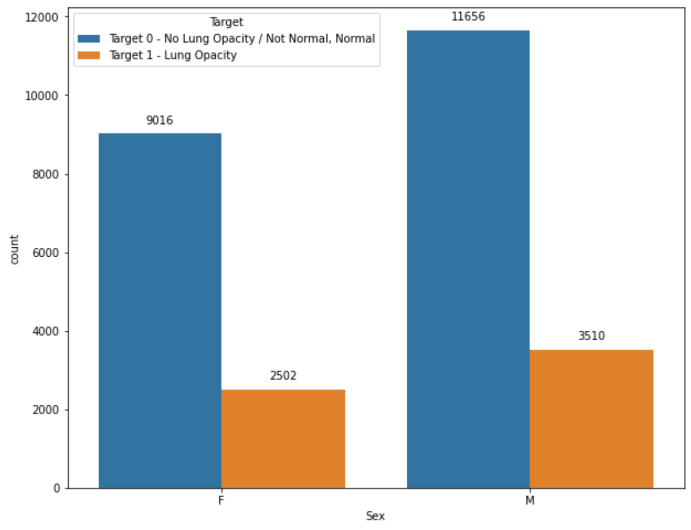
4.2.4 “Age distribution” form the given dataset





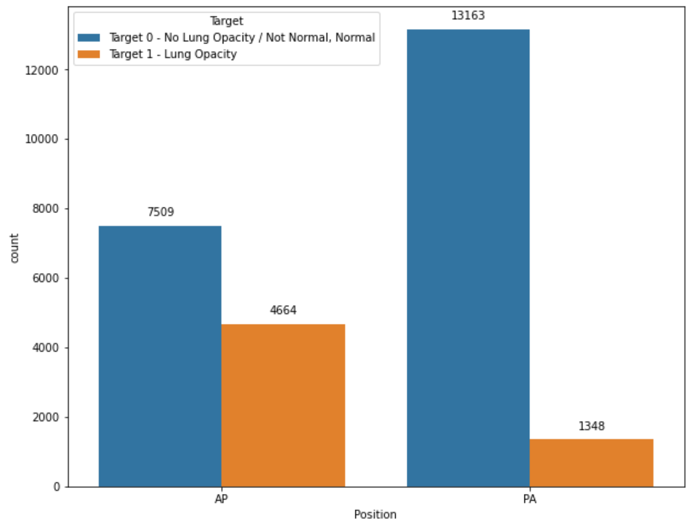
➢ More no. of age distribution lies within the range of 44 to 55.

4.2.5 Relationship between “Sex” and “Target”



- Almost equal number of images are provides from Male and Female

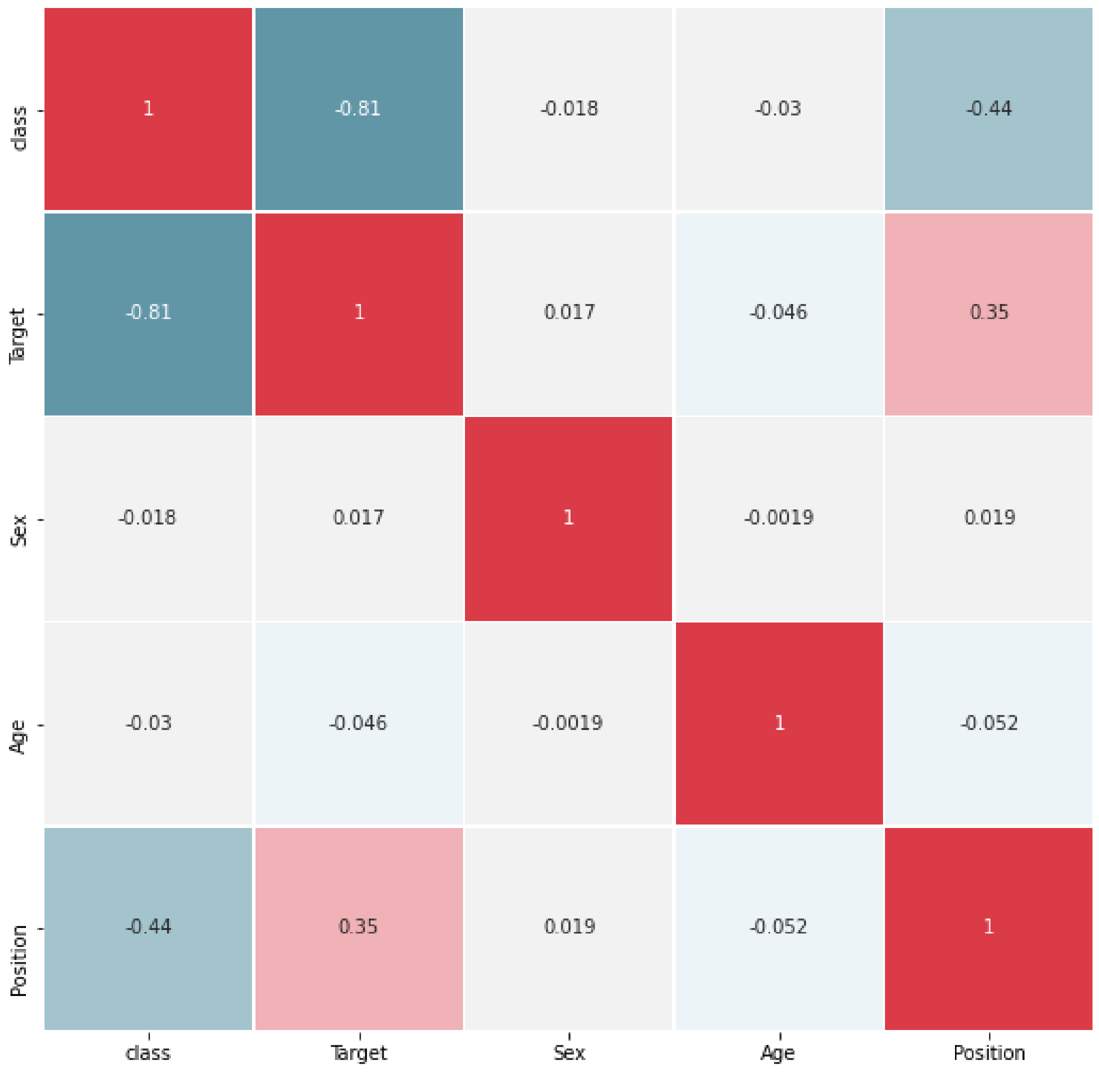
4.2.6 Relationship between “View Position” and “Target”



- View postion – AP (Anterior Posterior) having more possible chance of occurring Lung Opacity

4.2.7 Correlation map

- Correlation map between the attributes “Target”, “Class”, “Age”, “Sex” and “View Position” is plotted below

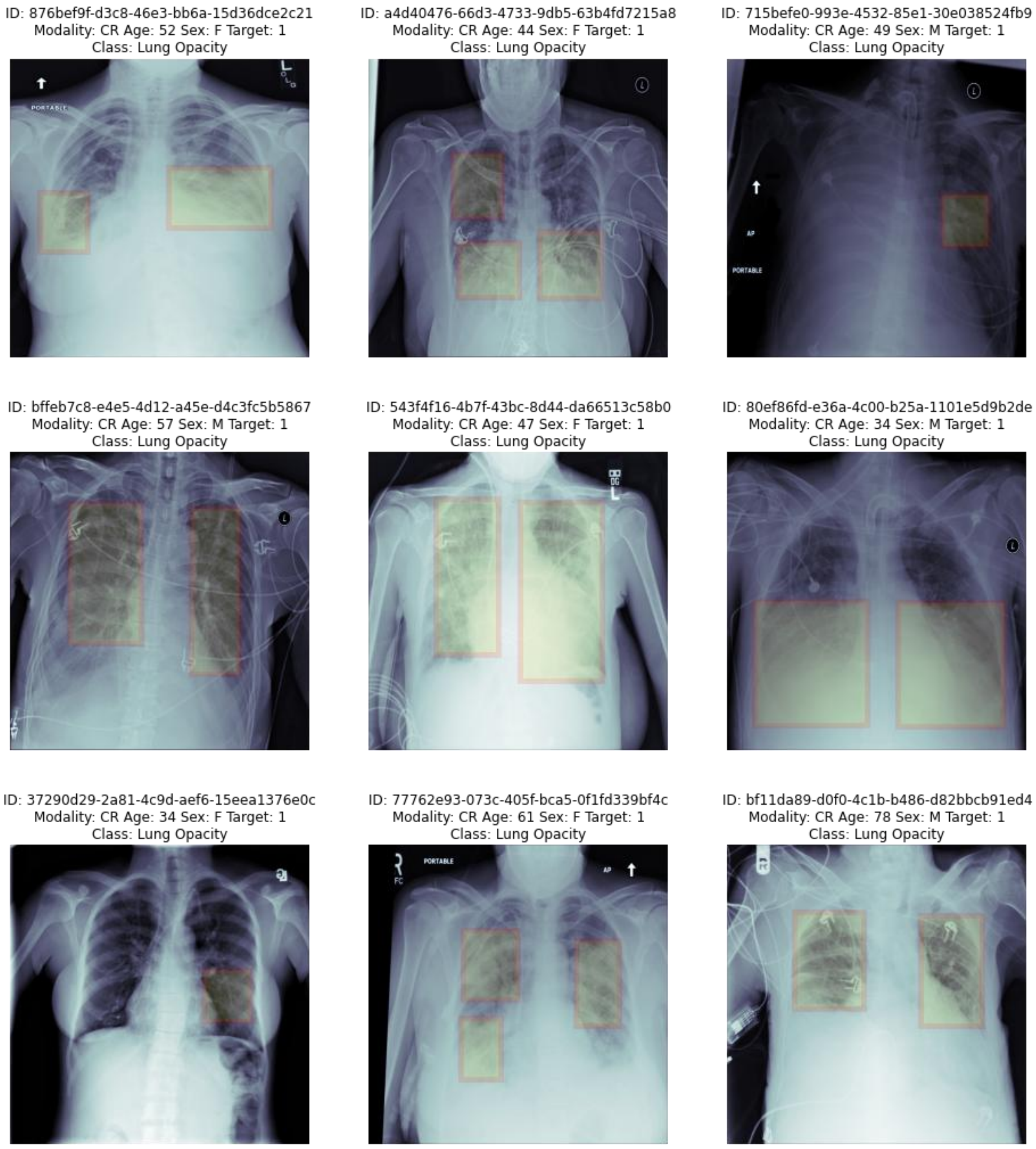


Summary after EDA

- **3543 difference observed between the information provided in the csv file and the available images.**
- **It is due to patient with lung opacity have multiple bounding box values. (i.e some patients have lung opacity at more than one places in the lungs)**
- **No missing values were observed and the given data can be used as it is for further modelling.**

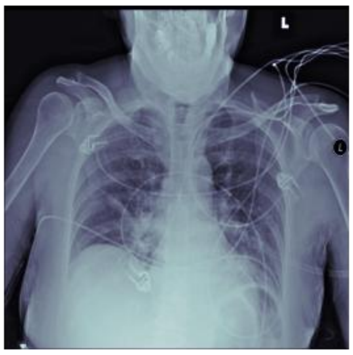
4.3 Visualizing the given images

- The images with Target =1 and class = Lung Opacity
- Form the given bounding box information, the lung opacity is highlighted with a bounding box
- A patient may have lung opacity in more than one places.

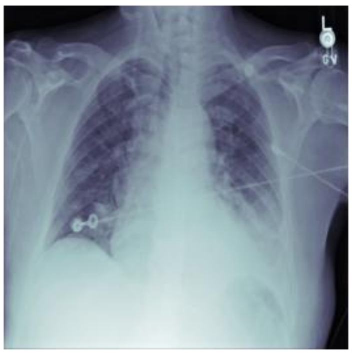

ID: 876bef9f-d3c8-46e3-bb6a-15d36dce2c21
Modality: CR Age: 52 Sex: F Target: 1
Class: Lung Opacity
ID: a4d40476-66d3-4733-9db5-63b4fd7215a8
Modality: CR Age: 44 Sex: F Target: 1
Class: Lung Opacity
ID: 715befe0-993e-4532-85e1-30e038524fb9
Modality: CR Age: 49 Sex: M Target: 1
Class: Lung Opacity
ID: bffeb7c8-e4e5-4d12-a45e-d4c3fc5b5867
Modality: CR Age: 57 Sex: M Target: 1
Class: Lung Opacity
ID: 543f4f16-4b7f-43bc-8d44-da66513c58b0
Modality: CR Age: 47 Sex: F Target: 1
Class: Lung Opacity
ID: 80ef86fd-e36a-4c00-b25a-1101e5d9b2de
Modality: CR Age: 34 Sex: M Target: 1
Class: Lung Opacity
ID: 37290d29-2a81-4c9d-aef6-15eea1376e0c
Modality: CR Age: 34 Sex: F Target: 1
Class: Lung Opacity
ID: 77762e93-073c-405f-bca5-0f1fd339bf4c
Modality: CR Age: 61 Sex: F Target: 1
Class: Lung Opacity
ID: bf11da89-d0f0-4c1b-b486-d82bbcb91ed4
Modality: CR Age: 78 Sex: M Target: 1
Class: Lung Opacity

- The images with Target =0 and class = No Lung Opacity/ Not Normal & Normal
- Bounding box information is not provided in this case

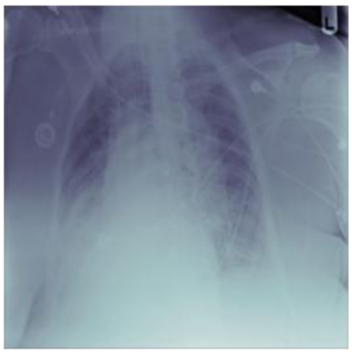
ID: 5a4042ff-dbda-4278-a556-206e7d723a51
Modality: CR Age: 66 Sex: F Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



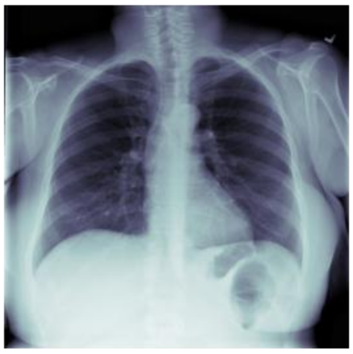
ID: cc9cea85-7cdd-4157-9c3f-d7c3edfaf4fa
Modality: CR Age: 67 Sex: M Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



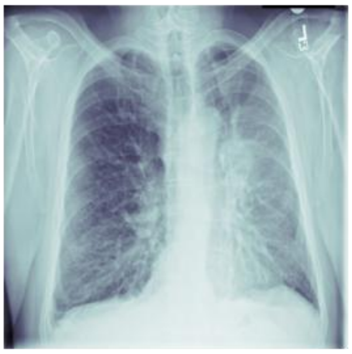
ID: 4e23ef60-7b48-489f-83fb-c06281b6e06e
Modality: CR Age: 68 Sex: F Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



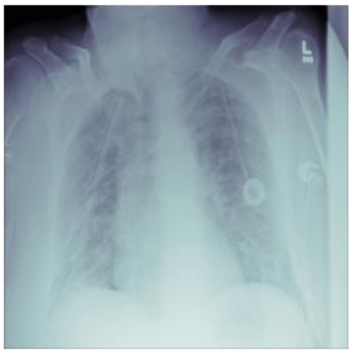
ID: 7e78d490-10fe-431e-a713-f60ae22105b3
Modality: CR Age: 37 Sex: F Target: 0
Class: Normal
Window: nan:nan:nan:nan



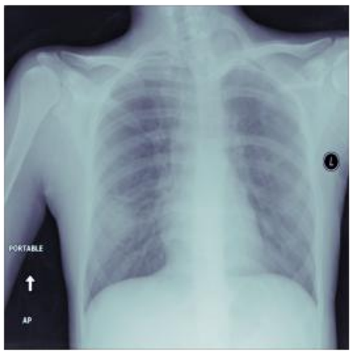
ID: 64a18f17-de76-44d1-9d6e-16267870298b
Modality: CR Age: 60 Sex: M Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



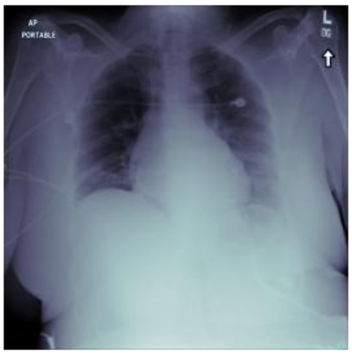
ID: ff1c8291-32bd-4d9a-8c4e-570dca044bcc
Modality: CR Age: 70 Sex: M Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



ID: 86f2e6c0-5775-45fb-91db-67dfddcddb4b
Modality: CR Age: 25 Sex: M Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



ID: 2f172025-0ba3-41d7-ada4-dcac1b651b97
Modality: CR Age: 68 Sex: F Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan



ID: ea266abc-6c5b-4921-a37a-44de924c203d
Modality: CR Age: 16 Sex: M Target: 0
Class: No Lung Opacity / Not Normal
Window: nan:nan:nan:nan

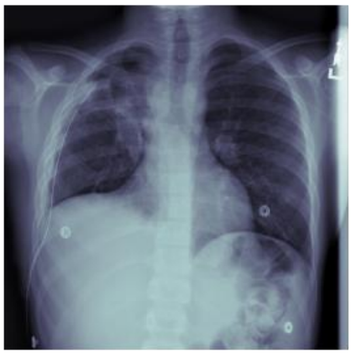


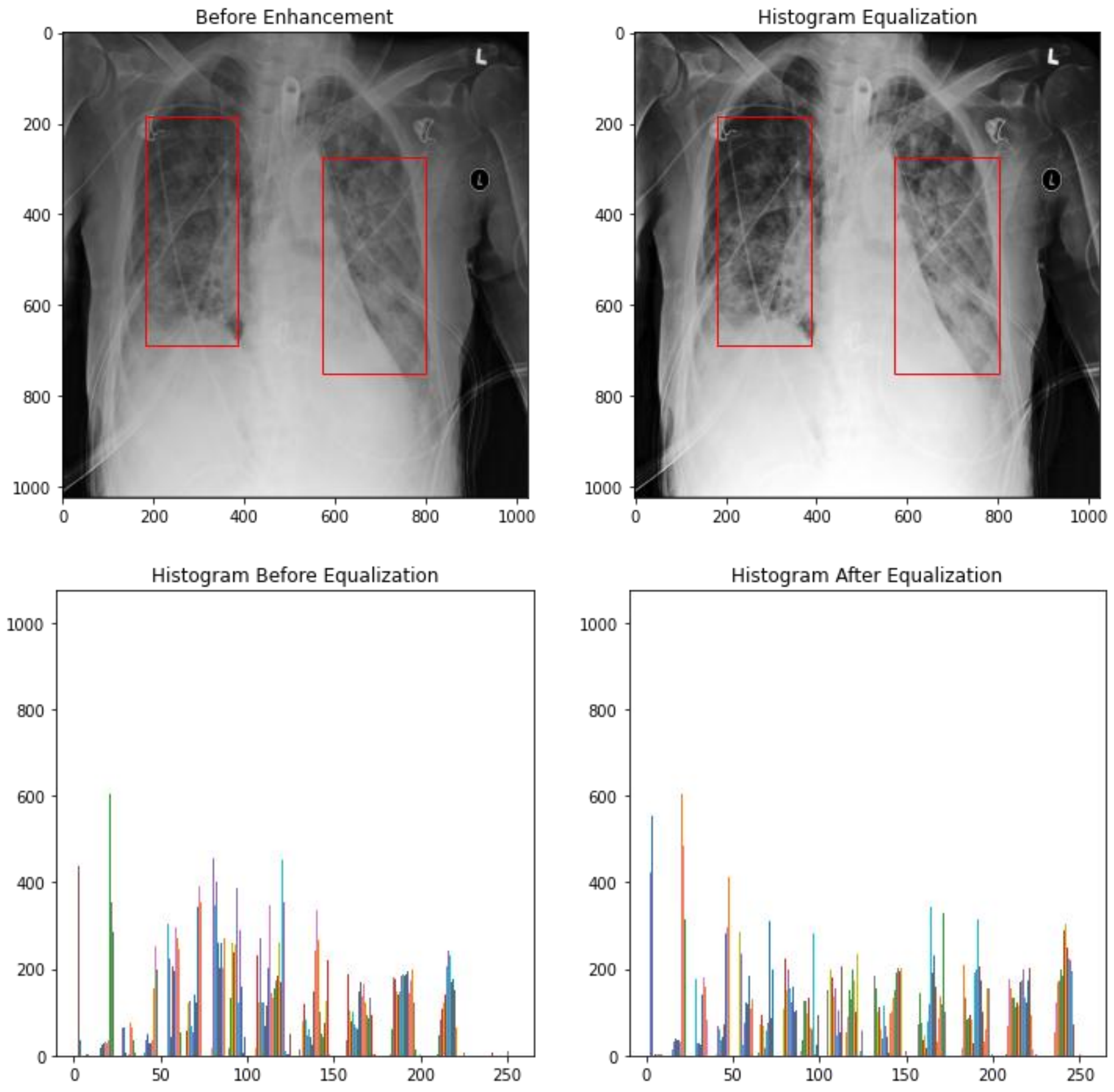
IMAGE PREPROCESSING

The given CXR images are having low contrast. So the contrast of the image is enhanced by using histogram equalization.

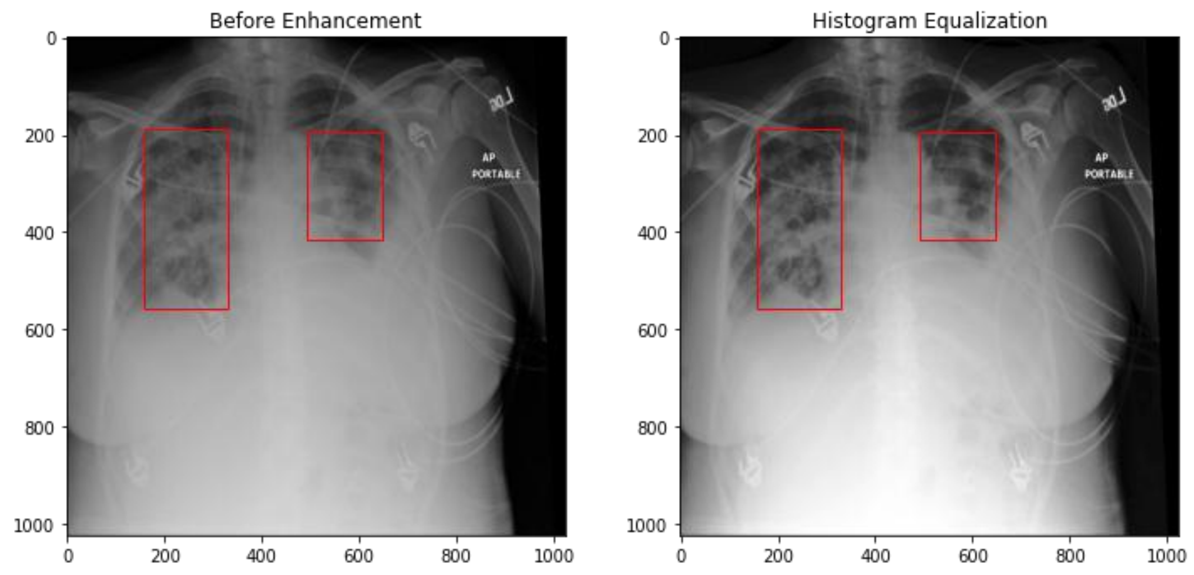
5.1 Histogram Equalization

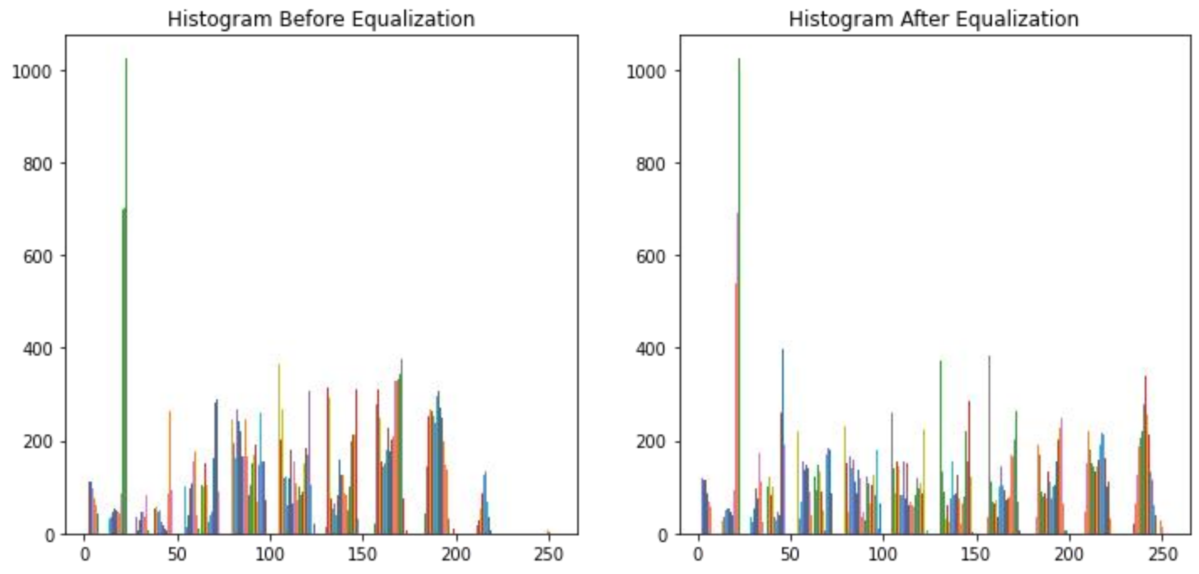
Histogram Equalization is a computer image processing technique used to improve contrast in images. It is a method in image processing for contrast adjustment using the image's histogram. Histogram equalization is simple & best method for image enhancement. It provides better quality of images without loss of any information.

5.2 Results Observed



File Name: 00a05408-8291-4231-886e-13763e103161.png





File Name: 00f08de1-517e-4652-a04f-d1dc9ee48593.png

The Result shows that the histogram equalized images provide better clarity than the original image.

DECIDING MODEL & MODEL BUILDING

6.1 Object Detection

The Object detectors in deep learning can be classified into one-stage detectors and two stage-detectors. Two-stage detectors are usually slower than one-stage detectors because of the generation of the candidate object location using an external module and because of the higher detection accuracy due to the consideration of the hard examples. Here, the hard examples are the examples poorly predicted by the model.

A faster RCNN is a two-stage object detector. It uses a Region Proposal Network (RPN) to generate candidate bounding boxes in first stage. In the second stage, the regions proposed by the RPN are used as an input for the Fast R-CNN model, which will provide the final object detection results.

Unlike Faster RCNN, RetinaNet is a one-stage object detector. It exploits FPN (Feature Pyramid Networks). The focal loss acts as a loss function to address the class imbalance and unequal contribution of hard and easy examples to the loss thus improving the detection accuracy. This will discussed in detail in later section.

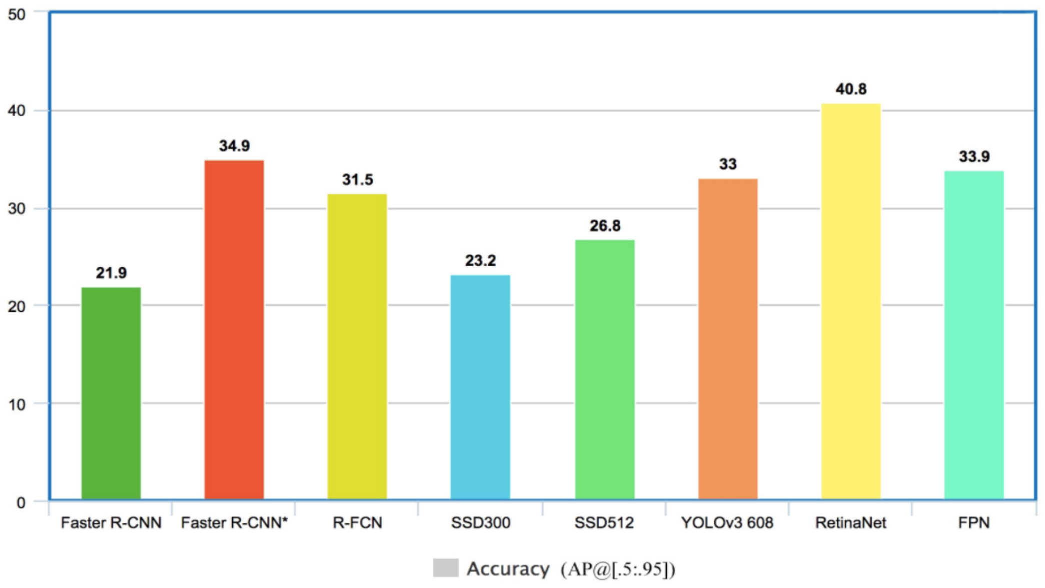
SSD is a single-stage object detector and has two parts. The first part, known as a base network, extracts multi-scale features, and the second detection part employs multi-scale features to classify objects and to obtain the bounding box containing the objects. Even though both SSD and RetinaNet extract multi-scale features, RetinaNet fuses the subsampling layer like SSD and the reconstructed semantic layer to improve the detection accuracy.

6.2 Selection of Dataset for Model Building

Initially to decide on model, after researching a lot we found out that various models were tested on COCO Object detection dataset. We decided to consider this as a baseline performance.

COCO dataset is harder for object detection and usually detectors achieve much lower mAP. Here are the comparison for some key detectors.

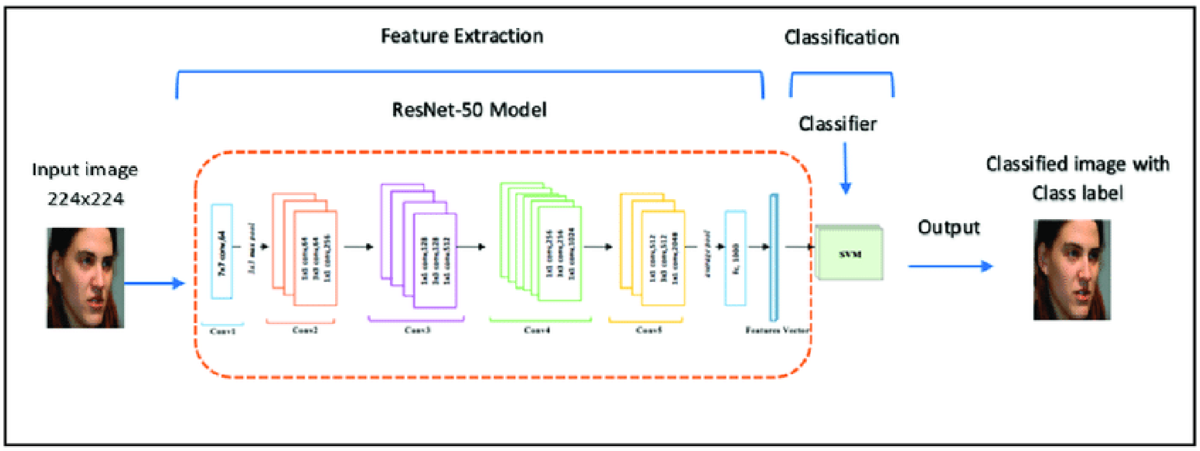
- FPN and Faster R-CNN*(using ResNet as the feature extractor) have the highest accuracy (mAP@[.5:.95]). RetinaNet builds on top of the FPN using ResNet. So the high mAP achieved by RetinaNet is the combined effect of pyramid features, the feature extractor's complexity and the focal loss.
- Yolo gives a decent mAP as well.
- Single shot detectors have a pretty impressive frame per seconds (**FPS**) using lower resolution images at the cost of accuracy. We left out SSD because it is more useful in case of real time processing but in our case the data is not real time so we have ignored it.



So to verify these results initially we used a simple resnet50 model to check for a base accuracy to start with.

6.3 Resnet50

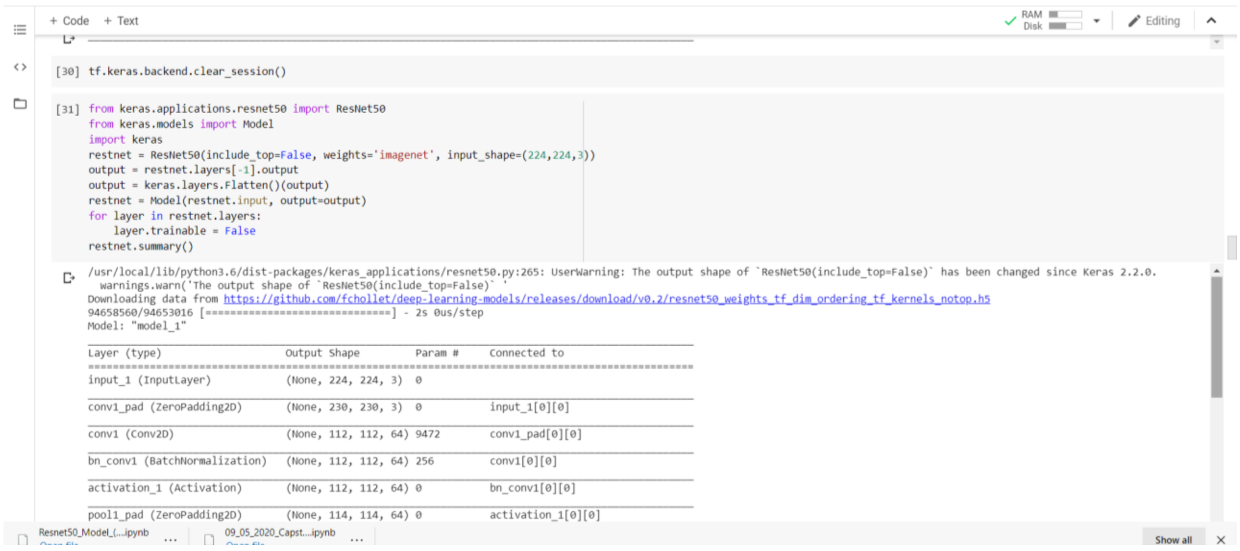
ResNet-50 is a convolutional neural network that is 50 layers deep. Pretrained version of Resnet50 was loaded from the ImageNet database.



The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

Since the input size of the network architecture is 224 *224 all the images are reduced from the size of 1024*1024 to 224*224. Additional three fully connected layers were added to the pretrained Resnet 50.

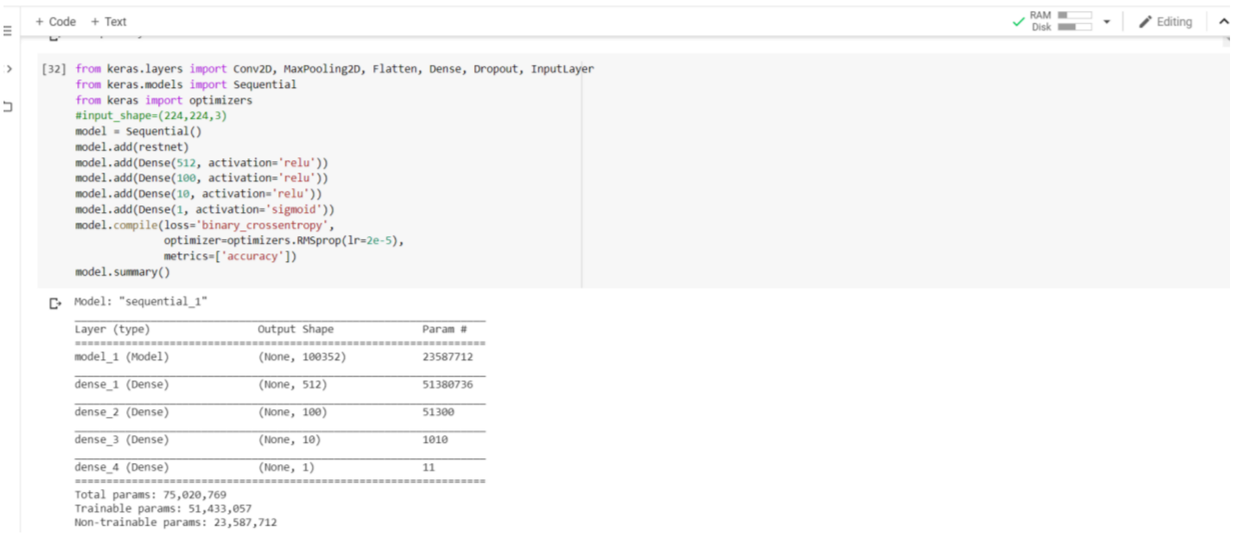
Notebook Screenshot



```
[30] tf.keras.backend.clear_session()
```

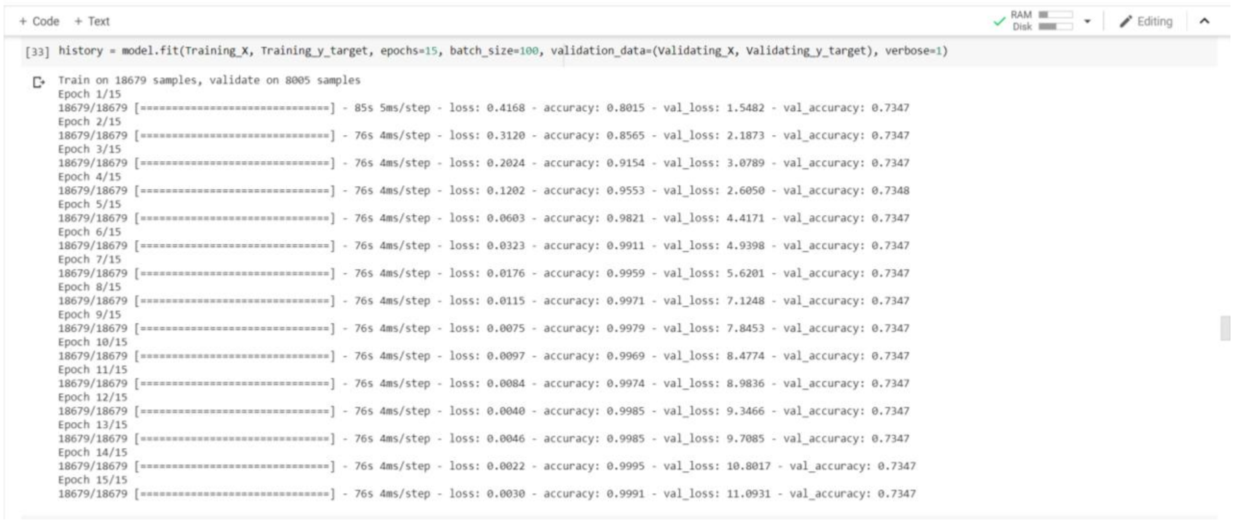
```
[31] from keras.applications.resnet50 import ResNet50
     from keras.models import Model
     import keras
     restnet = ResNet50(include_top=False, weights='imagenet', input_shape=(224,224,3))
     output = restnet.layers[-1].output
     output = keras.layers.Flatten()(output)
     restnet = Model(restnet.input, output=output)
     for layer in restnet.layers:
         layer.trainable = False
     restnet.summary()
```

```
/usr/local/lib/python3.6/dist-packages/keras_applications/resnet50.py:265: UserWarning: The output shape of `ResNet50(include_top=False)` has been changed since Keras 2.2.0.
  warnings.warn('The output shape of `ResNet50(include_top=False)` '
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94658560/94653016 [==============================] - 2s 0us/step
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 224, 224, 3)  0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 112, 112, 64) 256         conv1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 112, 112, 64) 0           bn_conv1[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           activation_1[0][0]
```



```
[32] from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
     from keras.models import Sequential
     from keras import optimizers
     #input_shape=(224,224,3)
     model = Sequential()
     model.add(restnet)
     model.add(Dense(512, activation='relu'))
     model.add(Dense(100, activation='relu'))
     model.add(Dense(10, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))
     model.compile(loss='binary_crossentropy',
                   optimizer=optimizers.RMSprop(lr=2e-5),
                   metrics=['accuracy'])
     model.summary()
```

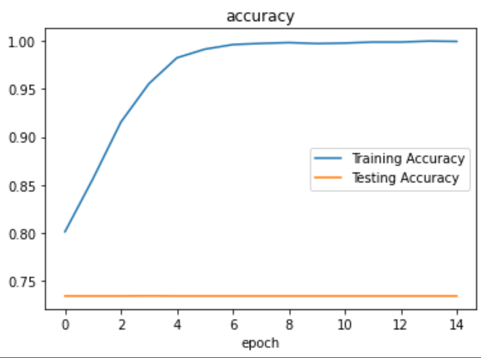
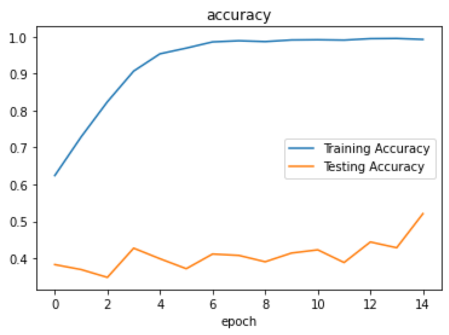
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 100352)            23587712
_________________________________________________________________
dense_1 (Dense)              (None, 512)               51380736
_________________________________________________________________
dense_2 (Dense)              (None, 100)               51300
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 75,020,769
Trainable params: 51,433,057
Non-trainable params: 23,587,712
```



```
[33] history = model.fit(Training_X, Training_y_target, epochs=15, batch_size=100, validation_data=(Validating_X, Validating_y_target), verbose=1)
```

```
Train on 18679 samples, validate on 8005 samples
Epoch 1/15
18679/18679 [==============================] - 85s 5ms/step - loss: 0.4168 - accuracy: 0.8015 - val_loss: 1.5482 - val_accuracy: 0.7347
Epoch 2/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.3120 - accuracy: 0.8565 - val_loss: 2.1873 - val_accuracy: 0.7347
Epoch 3/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.2024 - accuracy: 0.9154 - val_loss: 3.0789 - val_accuracy: 0.7347
Epoch 4/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.1202 - accuracy: 0.9553 - val_loss: 2.6050 - val_accuracy: 0.7348
Epoch 5/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0603 - accuracy: 0.9821 - val_loss: 4.4171 - val_accuracy: 0.7347
Epoch 6/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0323 - accuracy: 0.9911 - val_loss: 4.9398 - val_accuracy: 0.7347
Epoch 7/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0176 - accuracy: 0.9959 - val_loss: 5.6201 - val_accuracy: 0.7347
Epoch 8/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0115 - accuracy: 0.9971 - val_loss: 7.1248 - val_accuracy: 0.7347
Epoch 9/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0075 - accuracy: 0.9979 - val_loss: 7.8453 - val_accuracy: 0.7347
Epoch 10/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0097 - accuracy: 0.9969 - val_loss: 8.4774 - val_accuracy: 0.7347
Epoch 11/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0084 - accuracy: 0.9974 - val_loss: 8.9836 - val_accuracy: 0.7347
Epoch 12/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0040 - accuracy: 0.9985 - val_loss: 9.3466 - val_accuracy: 0.7347
Epoch 13/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0046 - accuracy: 0.9985 - val_loss: 9.7085 - val_accuracy: 0.7347
Epoch 14/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0022 - accuracy: 0.9995 - val_loss: 10.8017 - val_accuracy: 0.7347
Epoch 15/15
18679/18679 [==============================] - 76s 4ms/step - loss: 0.0030 - accuracy: 0.9991 - val_loss: 11.0931 - val_accuracy: 0.7347
```

The model was trained by considering the attribute "class" and "Target"
The following results were observed after 15 epochs.

Evaluation Parameters	**Resnet50 with the attribute "Target"**	**Resnet50 with the attribute "Class"**
Training Accuracy	99.74%	99.46%
Validation Accuracy	73.47%	34.48%
Recall	50 %	35.68 %
Precision	36.73 %	47.91 %
F1 Score	42.35 %	21.11 %
Confusion Matrix	[[5881 0] [2124 0]]	[[178 3138 19] [1 2545 0] [226 1861 37]]
		

- The model seems to be overfit.
- The performance of the model can be improved further if the number of epochs increased and by considering the layers of the Resnet as trainable parameter.

Limitation of this model: This model can be used for classifying the images. The location of the lung opacity can not be predicted using this model.

6.4 Faster RCNN

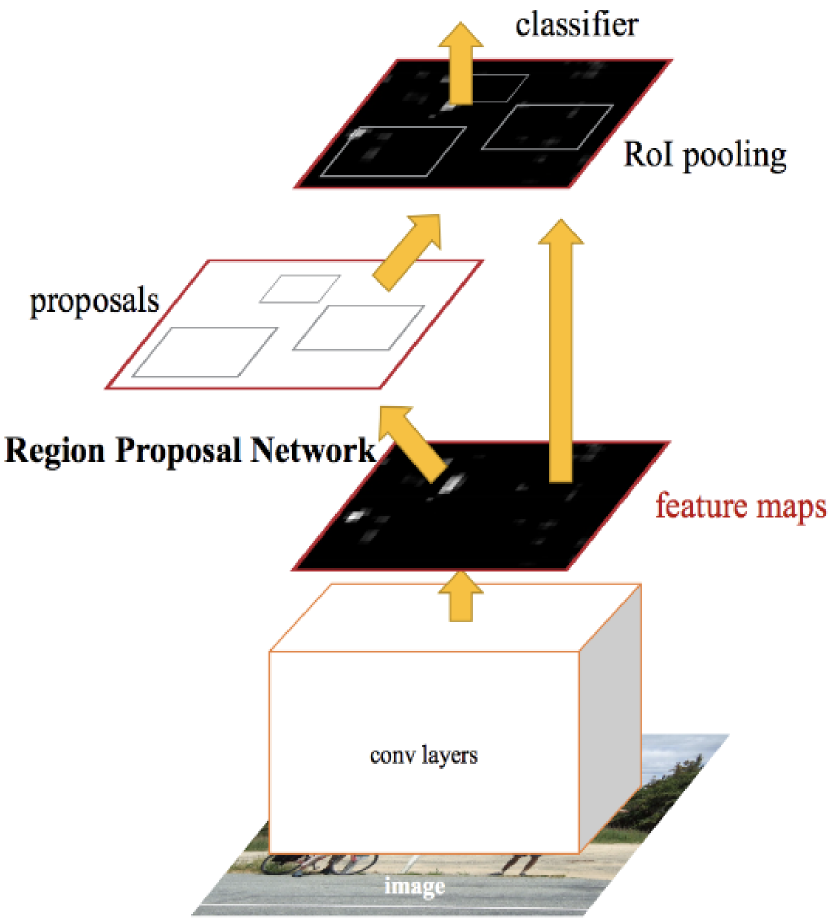
We also tried implementing Faster-RCNN using Keras with VGG-16 as backbone,owing to its high accuracy and faster results in object detection technique.

Faster RCNN is an object detection architecture that uses convolution neural networks like YOLO (You Look Only Once) and SSD (Single Shot Detector). Faster RCNN is the modified version of Fast RCNN. The major difference between them is that Fast RCNN uses selective search for generating Regions of Interest, while Faster RCNN uses "Region Proposal

Network", aka RPN. RPN takes image feature maps as an input and generates a set of object proposals, each with an objectness score as output.

The below steps are typically followed in a Faster RCNN approach:

1. We take an image as input and pass it to the ConvNet which returns the feature map for that image.
2. Region proposal network is applied on these feature maps. This returns the object proposals along with their objectness score.
3. A RoI pooling layer is applied on these proposals to bring down all the proposals to the same size.
4. Finally, the proposals are passed to a fully connected layer which has a softmax layer and a linear regression layer at its top, to classify and output the bounding boxes for objects.



Region Proposal Network (RPN)

To begin with, Faster RCNN takes the feature maps from CNN and passes them on to the Region Proposal Network. RPN uses a sliding window over these feature maps, and at each window, it generates K Anchor boxes of different shapes and sizes.

Anchor boxes are fixed sized boundary boxes that are placed throughout the image and have different shapes and sizes. For each anchor, RPN predicts two things:

- The first is the probability that an anchor is an object (it does not consider which class the object belongs to)
- Second is the bounding box regressor for adjusting the anchors to better fit the object

We now have bounding boxes of different shapes and sizes which are passed on to the RoI pooling layer. Now it might be possible that after the RPN step, there are proposals with no classes assigned to them. We can take each proposal and crop it so that each proposal contains an object. This is what the RoI pooling layer does. It extracts fixed sized feature maps for each anchor:

Then these feature maps are passed to a fully connected layer which has a softmax and a linear regression layer. It finally classifies the object and predicts the bounding boxes for the identified objects.

For every point in the output feature map, the network has to learn whether an object is present in the input image at its corresponding location and estimate its size. This is done by placing a set of “Anchors” on the input image for each location on the output feature map from the backbone network. These anchors indicate possible objects in various sizes and aspect ratios at this location. The figure below shows 9 possible anchors in 3 different aspect ratios and 3 different sizes placed on the input image for a point A on the output feature map. For the PASCAL challenge, the anchors used have 3 scales of box area 128^2, 256^2, 512^2 and 3 aspect ratios of 1:1, 1:2 and 2:1.

As the network moves through each pixel in the output feature map, it has to check whether these K corresponding anchors spanning the input image actually contain objects, and refine these anchors’ coordinates to give bounding boxes as “Object proposals” or regions of interest.

Evaluating object detection models

The most common evaluation metric that is used in object recognition tasks is ‘mAP’, which stands for **‘mean average precision’**. It is a number from 0 to 100 and higher values are typically better, but it’s value is different from the accuracy metric in classification.

Each bounding box will have a score associated (likelihood of the box containing an object). Based on the predictions a precision-recall curve (PR curve) is computed for each class by varying the score threshold. The average precision (AP) is the area under the PR curve. First the AP is computed for each class, and then averaged over the different classes. The end result is the mAP.

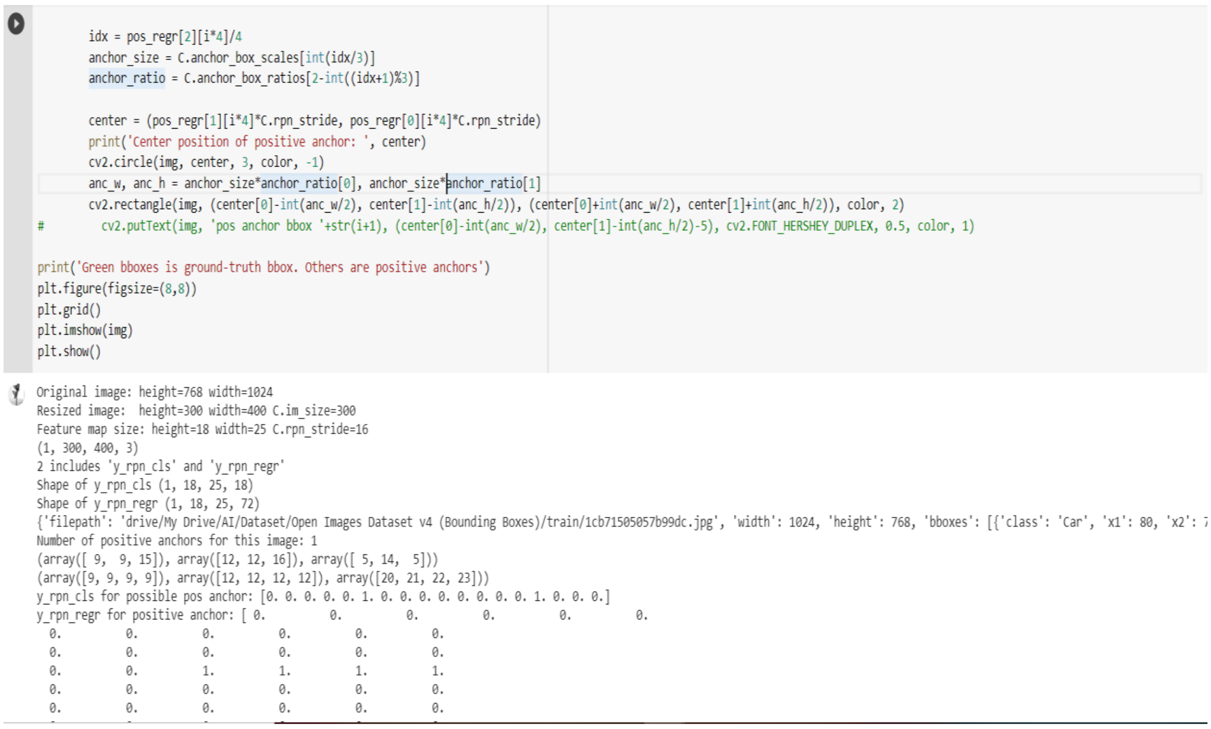
Note that a detection is a true positive if it has an **‘intersection over union’** (IoU or overlap) with the ground-truth box greater than some threshold (usually 0.5). Instead of using mAP we typically use mAP@0.5 or mAP@0.25 to refer to the IoU that was used.

Score = Area of overlap / Area of union

Model Improvement

We propose an improved algorithm based on faster region-based CNN (Faster R-CNN) for small object detection. Using the two-stage detection idea, in the positioning stage, we propose an improved loss function based on intersection over Union (IoU) for bounding box regression, and use bilinear interpolation to improve the regions of interest (RoI) pooling operation to solve the problem of positioning deviation, in the recognition stage, we use the multi-scale convolution feature fusion to make the feature map contain more information, and use the improved non-maximum suppression (NMS) algorithm to avoid loss of overlapping objects.

Notebook Screenshot



```
        idx = pos_regr[2][i*4]/4
        anchor_size = C.anchor_box_scales[int(idx/3)]
        anchor_ratio = C.anchor_box_ratios[2-int((idx+1)%3)]

        center = (pos_regr[1][i*4]*C.rpn_stride, pos_regr[0][i*4]*C.rpn_stride)
        print('Center position of positive anchor: ', center)
        cv2.circle(img, center, 3, color, -1)
        anc_w, anc_h = anchor_size*anchor_ratio[0], anchor_size*anchor_ratio[1]
        cv2.rectangle(img, (center[0]-int(anc_w/2), center[1]-int(anc_h/2)), (center[0]+int(anc_w/2), center[1]+int(anc_h/2)), color, 2)
#         cv2.putText(img, 'pos anchor bbox '+str(i+1), (center[0]-int(anc_w/2), center[1]-int(anc_h/2)-5), cv2.FONT_HERSHEY_DUPLEX, 0.5, color, 1)

print('Green bboxes is ground-truth bbox. Others are positive anchors')
plt.figure(figsize=(8,8))
plt.grid()
plt.imshow(img)
plt.show()
```

```
Original image: height=768 width=1024
Resized image:  height=300 width=400 C.im_size=300
Feature map size: height=18 width=25 C.rpn_stride=16
(1, 300, 400, 3)
2 includes 'y_rpn_cls' and 'y_rpn_regr'
Shape of y_rpn_cls (1, 18, 25, 18)
Shape of y_rpn_regr (1, 18, 25, 72)
{'filepath': 'drive/My Drive/AI/Dataset/Open Images Dataset v4 (Bounding Boxes)/train/1cb71505057b99dc.jpg', 'width': 1024, 'height': 768, 'bboxes': [{'class': 'Car', 'x1': 80, 'x2': 
Number of positive anchors for this image: 1
(array([ 9,  9, 15]), array([12, 12, 16]), array([ 5, 14,  5]))
(array([9, 9, 9, 9]), array([12, 12, 12, 12]), array([20, 21, 22, 23]))
y_rpn_cls for possible pos anchor: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
y_rpn_regr for positive anchor: [ 0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         1.         1.         1.         1.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
```

6.5 Mask-RCNN

Concurrently we found a model Mask RCNN which gives a better accuracy and can be used for segmentation as well as predicting bounding boxes.

Mask R-CNN is basically an extension of Faster R-CNN. Faster R-CNN is widely used for object detection tasks. For a given image, it returns the class label and bounding box coordinates for each object in the image. **The Mask R-CNN framework is built on top of Faster R-CNN**. So, for a given image, Mask R-CNN, in addition to the class label and bounding box coordinates for each object, will also return the object mask.

Mask RCNN model generates bounding boxes and segmentation masks for each instance of an object in the image. It is based on **Feature Pyramid Network (FPN) and a ResNet101 backbone.**

We need to understand the working of Faster RCNN to grasp the intuition behind the Mask RCNN as well:

- Faster R-CNN first uses a ConvNet to extract feature maps from the images
- These feature maps are then passed through a Region Proposal Network (RPN) which returns the candidate bounding boxes
- We then apply an RoI pooling layer on these candidate bounding boxes to bring all the candidates to the same size
- And finally, the proposals are passed to a fully connected layer to classify and output the bounding boxes for objects

Backbone Model

Similar to the ConvNet that we are using in Faster R-CNN to extract feature maps from the image, we use the ResNet 101 architecture to extract features from the images in Mask R-CNN. So, the first step is to take an image and extract features using the ResNet 101 architecture. These features act as an input for the next layer.

Region Proposal Network (RPN)

Now, after obtaining the feature maps from the previous step and we have to apply a region proposal network (RPM). Which basically predicts if an object is present in that region (or not). In this step, we get those regions or feature maps which the model predicts contain some object.

Region of Interest (RoI)

The regions obtained from the RPN might be of different shapes. Hence, we apply a pooling layer and convert all the regions to the same shape. Next, these regions are passed through a fully connected network so that the class label and bounding boxes are predicted.

Till this point, the steps are almost like Faster R-CNN. Now comes the difference between the two frameworks. In addition to this, Mask R-CNN also generates the segmentation mask.

For that, we first compute the region of interest so that the computation time can be reduced. For all the predicted regions, we compute the Intersection over Union (IoU) with the ground truth boxes. We can computer IoU like this:

IoU = Area of the intersection / Area of the union

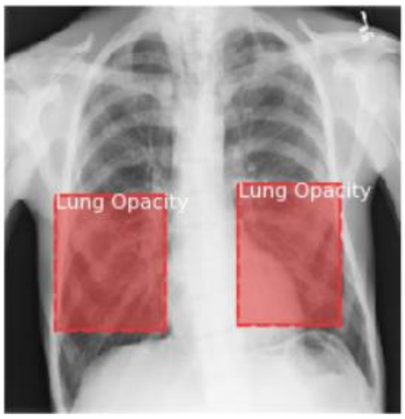
Now, only if the IoU is greater than or equal to 0.5, we consider that as a region of interest. Otherwise, we neglect that region. We do this for all the regions and then select only a set of regions for which the IoU is greater than 0.5.

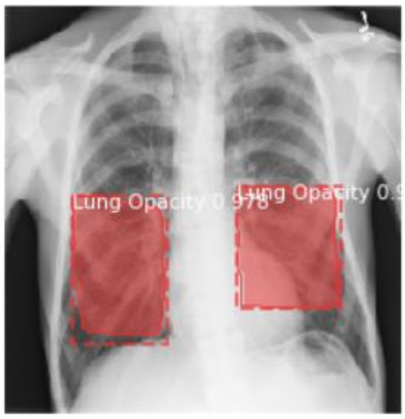
Final step of Mask RCNN: Segmentation Mask

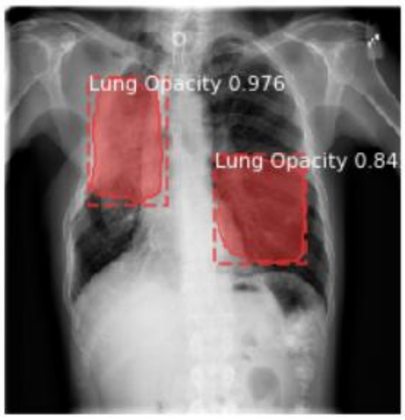
Once we have the RoIs based on the IoU values, we have added a mask branch to the existing architecture. This returns the segmentation mask for each region that contains an object. It returns a mask of size 28 X 28 for each region which is then scaled up for inference.

This model aims to segment all the objects in the image. This is the final step in Mask R-CNN where we predict the masks for all the objects in the image.

Note: Training time for Mask R-CNN is quite high. It can take around 1 to 2 days to train the Mask R-CNN on the famous COCO dataset. So, for this project we used the pretrained weights of the Mask R-CNN model trained on the COCO dataset.



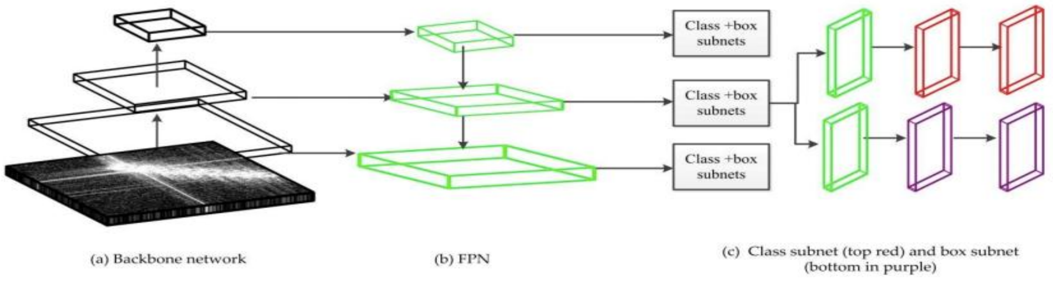




6.6 RetinaNet

We also tried implementing RetinaNet using Keras with Resnet50 as backbone, owing to its high accuracy results and faster speed as it is a one-stage object detection technique.

For Introduction, the architecture of RetinaNet has three components, i.e., a backbone network for feature extraction and two subnetworks, i.e., one for classification and the other for box



regression.

Findings from RetinaNet architecture -

a) RetinaNet uses backbone network, such as residual networks (ResNet), convolutional neural networks named by Visual Geometry Group (VGG), or densely connected convolutional networks (DenseNet) to extract higher semantic feature maps.

b) FPN is applied to extract the same dimension features with various scales. For example multi object detection. **This is a very important observation because by using RetinaNet we can not only limit our model to detect pneumonia but many other anomalies in future with very less effort.**

c) There are two subnets, i.e., the top red one for classification and the purple bottom for bounding box regression.

Brief Intro of FPN

Feature pyramid networks act as a feature extractor with the consideration of the low-level high-resolution and high-level low-resolution semantic meaning .

FPN has two pathways, including a bottom-up pathway which employs convolutional networks to extract features from a high-resolution input and a top-down pathway which constructs the high-resolution multi-scale layers from the top layer in the bottom-up pathway. The bottom-up pathway usually consists of parts of convolutional neural networks, such as VGG16 , ResNet , and DenseNet.

As for the top-down pathway, it first adapts a 1 × 1 convolution to reduce the number of feature map channels to 256 and then uses lateral connections to combine the corresponding feature map and the reconstructed layers to help the detector better predict the location.

Focal Loss

The very essence of RetinaNet implementing focal loss differentiates it and makes it superior to other models.

RetinaNet belongs to the **one-stage object** detectors in deep learning. It uses FPN to obtain multi-scale features, which are used for object classification and bounding box regression. As for the **bounding box regression**, FPN first compares the candidate bounding boxes with the ground truth to acquire the positive and negative examples. During the training, the class imbalance and unequal contribution of hard and easy examples to the loss have an **impact** on the detection accuracy.

To counter this, the focal loss is used. It puts more emphasis on hard examples and focuses on the fact that the loss of hard examples is higher during training. It is expressed as Equation (1) and has been used to improve the detection accuracy.

$$FL(pt) = -\alpha t(1 - pt)\ \gamma\ \log(pt)$$

where αt and γ are two hyper-parameters and they function as the role of moderating the weights between easy and hard examples.

Implementing the model

As we have seen in the before sections, on COCO Dataset RetinaNet gives a high AP of 40.8. Here they chose $\alpha = 0.25$, $\gamma = 2$, Adam optimiser and lr = 0.00001 and backbone is chosen as resnet50.

We are thinking to use ResNet50, ResNet151 and VGG as backbones for our implementation. We have used ResNet50 with the same hyper-parameters as above already and observed the predictions to be good.
Since αt and γ influence the experimental results, we will be setting them different, i.e., $\{(\alpha = 0.25, \gamma = 2),(\alpha = 0.5, \gamma = 2), \alpha = 0.25, \gamma = 1\}$, to better select the model.

It was also found that resolution of training images may affect the accuracy as well.

6.7 YOLOv3

YOLO v3 uses a variant of Darknet, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a **106 layer fully convolutional underlying architecture for YOLO v3**.

In YOLO v3, **the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network.**

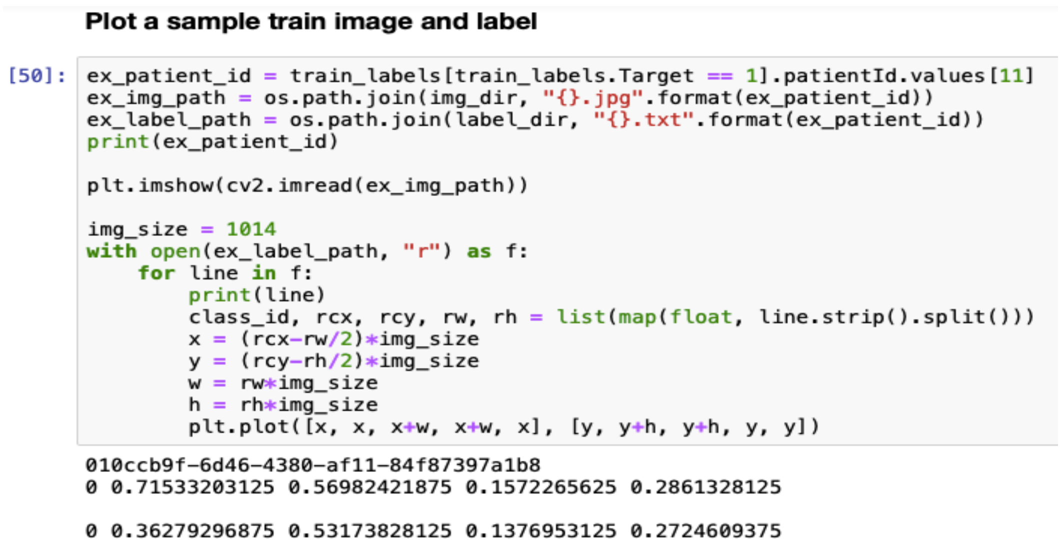
Detections at different layers helps address the issue of detecting small objects, a frequent complaint with YOLO v2. This was the reason we tried to check out this model as a lung opacity maybe a smaller section in our image to be detected.

Object confidence and class predictions in YOLO v3 are now predicted through logistic regression.

We have implemented this model using darkness framework and observed a mAP of 0.14 on RSNA dataset. We find it is too early to reach to a conclusion before we test out other models. But mAP of 0.14 on medical data is considered to be a decent score.

We have used weights from pertained model darknet53.conv.74.

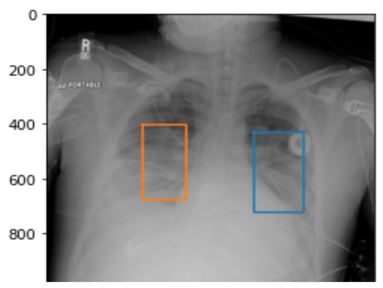
Added some screenshots of the notebook.

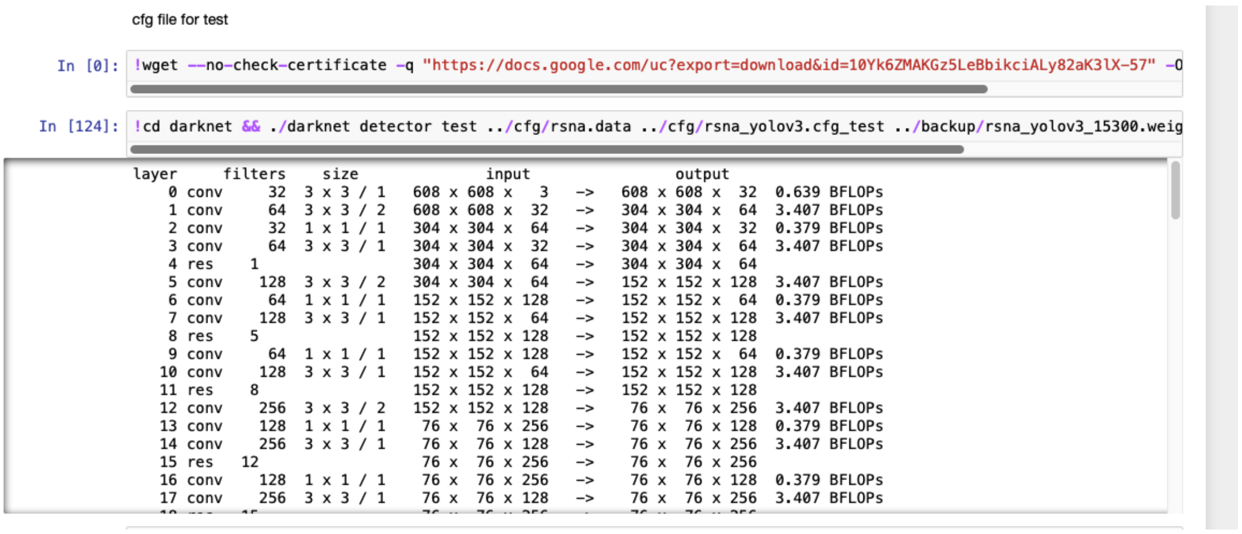


Plot a sample train image and label

```
[50]: ex_patient_id = train_labels[train_labels.Target == 1].patientId.values[11]
      ex_img_path = os.path.join(img_dir, "{}.jpg".format(ex_patient_id))
      ex_label_path = os.path.join(label_dir, "{}.txt".format(ex_patient_id))
      print(ex_patient_id)

      plt.imshow(cv2.imread(ex_img_path))

      img_size = 1014
      with open(ex_label_path, "r") as f:
          for line in f:
              print(line)
              class_id, rcx, rcy, rw, rh = list(map(float, line.strip().split()))
              x = (rcx-rw/2)*img_size
              y = (rcy-rh/2)*img_size
              w = rw*img_size
              h = rh*img_size
              plt.plot([x, x, x+w, x+w, x], [y, y+h, y+h, y, y])
```

```
010ccb9f-6d46-4380-af11-84f87397a1b8
0 0.71533203125 0.56982421875 0.1572265625 0.2861328125

0 0.36279296875 0.53173828125 0.1376953125 0.2724609375
```



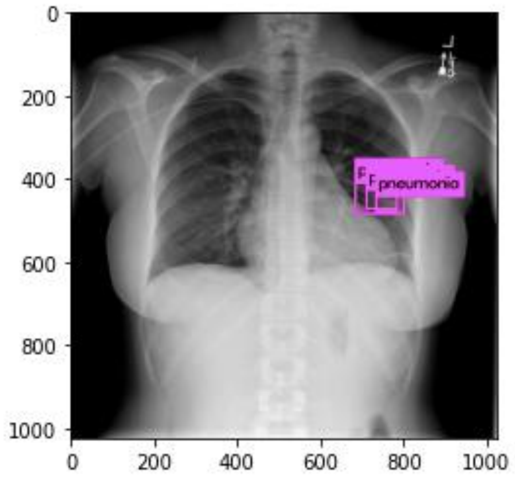


cfg file for test

```
In [0]: !wget --no-check-certificate -q "https://docs.google.com/uc?export=download&id=10Yk6ZMAKGz5LeBbikciALy82aK3lX-57" -O

In [124]: !cd darknet && ./darknet detector test ../cfg/rsna.data ../cfg/rsna_yolov3.cfg_test ../backup/rsna_yolov3_15300.weig
```

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32  0.639 BFLOPs
    1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    2 conv     32  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  32  0.379 BFLOPs
    3 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64  3.407 BFLOPs
    4 res    1                 304 x 304 x  64   ->   304 x 304 x  64
    5 conv    128  3 x 3 / 2   304 x 304 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    6 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
    7 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
    8 res    5                 152 x 152 x 128   ->   152 x 152 x 128
    9 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
   10 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
   11 res    8                 152 x 152 x 128   ->   152 x 152 x 128
   12 conv    256  3 x 3 / 2   152 x 152 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   13 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   14 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
   15 res   12                  76 x  76 x 256   ->    76 x  76 x 256
   16 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
   17 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
```

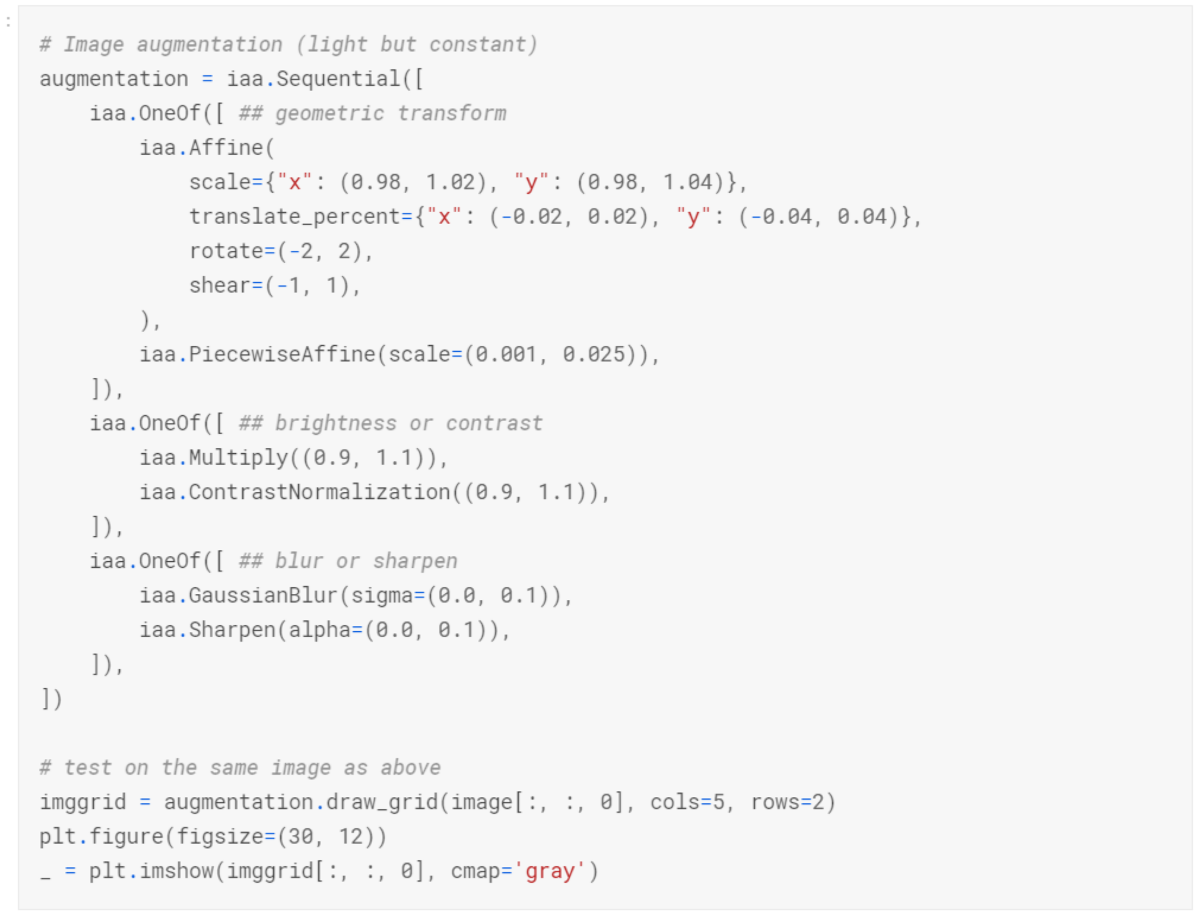
```
In [125]: plt.imshow(cv2.imread("./darknet/predictions.jpg"))

Out[125]: <matplotlib.image.AxesImage at 0x7f0cc729c2e8>
```



WAYS TO IMPROVE THE MODEL PERFORMANCE

The model performance can be further improved by

- Data Augumenation
- By applying different hyperparameter values { α , γ, lr }
- By loading different pretrained weights



```
# Image augmentation (light but constant)
augmentation = iaa.Sequential([
    iaa.OneOf([ ## geometric transform
        iaa.Affine(
            scale={"x": (0.98, 1.02), "y": (0.98, 1.04)},
            translate_percent={"x": (-0.02, 0.02), "y": (-0.04, 0.04)},
            rotate=(-2, 2),
            shear=(-1, 1),
        ),
        iaa.PiecewiseAffine(scale=(0.001, 0.025)),
    ]),
    iaa.OneOf([ ## brightness or contrast
        iaa.Multiply((0.9, 1.1)),
        iaa.ContrastNormalization((0.9, 1.1)),
    ]),
    iaa.OneOf([ ## blur or sharpen
        iaa.GaussianBlur(sigma=(0.0, 0.1)),
        iaa.Sharpen(alpha=(0.0, 0.1)),
    ]),
])

# test on the same image as above
imggrid = augmentation.draw_grid(image[:, :, 0], cols=5, rows=2)
plt.figure(figsize=(30, 12))
_ = plt.imshow(imggrid[:, :, 0], cmap='gray')
```

PROGRESS REVIEW

Tasks		Status
Understanding the data		Completed
Exploratory Data Analysis		Completed
Image Pre-processing		Completed
Model Building & Evaluation	Resnet50	Model Building completed with layers of Resnet as non-trainable. Classification accuracy may improve if Resnet layers are considered as trainable paramaters.
	RetinaNet	In-Progress
	FasterRCNN	In-Progress
	YOLO	Completed
	MaskRCNN	In-Progress
Comparison to benchmark		Will be done after Completing all the Model Building
Visualizations		Can be done after Model Building
Implications		Can be decided after selecting the best model
Limitations of the proposed method		Can be observed after deciding the best model
Closing Reflections		Can be decided after evaluating all the models.

REFERENCES

[1] https://www.kaggle.com/zahaviguy/what-are-lung-opacities

[2] https://www.healthline.com/health/pneumonia#in-kids

[3] https://en.wikipedia.org/wiki/Histogram_equalization

[4] https://towardsdatascience.com/histogram-equalization-5d1013626e64

[5] https://arxiv.org/pdf/1905.08545.pdf

[6] Redmon, J.; Farhadi, A. Yolo9000: Better, faster, stronger. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6517–6525.

[7] LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. Nature 2015, 521, 436–444. [CrossRef] [PubMed]

[8] Lin, T.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 936–944.

[9] https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[10] https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[11] https://pjreddie.com/darknet/yolo/

[12] https://arxiv.org/abs/1708.02002

[13] https://www.researchgate.net/publication/330330962_Deep_RetinaNet-Based_ Detection_ and_Classification_of_Road_Markings_by_Visible_Light_Camera_Sensors

[14] https://blog.zenggyu.com/en/post/2018-12-05/retinanet-explained-and-demystified/