FINAL PROJECT REPORT – Pneumonia detection challenge

Batch name- AIML online November 19-B

GROUP-4 AIML CAPSTONE -CV PROJECT

Done by

Gautam Ravi kumar

Shruthi V

Aishvarya Shri

Piyush Panchariya

Akshay Badhani

RSNA PNEUMONIA DETECTION CHALLENGE

Abstract

Pneumonia is a condition caused due to the inflammation of the lungs that primarily affects the small type of air sacs called as alveoli. This disease can be caused by bacteria, fungi or virus. It is often diagnosed by symptoms, physical type of examination and by employing the method of chest radiograph (CXR). Factors such as positioning of the patient and the depth of inspiration result in changing the appearance of the CXR which leads to complicated interpretations further. The objective of this study is to build an algorithm to detect a visual signal for pneumonia in the medical images that specifically locate the lung opacities and the position of inflammation in an image on the chest radiographs. The data set consists of the medical images which contain a combination of the header meta data as well as raw image arrays for the pixel data. The progress of the study includes pre-processing, data visualization and exploratory data analysis of the given data set. Then an object detection model is built and employment of transfer learning process to fine tune the model. Thus, based on the analysis made, certain automation systems can be made possible in detection of pneumonia which will help doctors to make better clinical decisions using the powerful AI techniques.

Pneumonia Detection

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. 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. 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.

Problem statement

Pneumonia accounts for the increase in the mortality rate of the human beings which demands for this AI challenge towards detection and localising the pneumonia. Factors pertaining to the complications in the interpretations made by the doctors by looking at the images of the CXR are change in the positioning of the patient and the depth of the inspiration can alter the appearance of the CXR images. Because of this, doctors would face difficulties in treating the patients since they will be faced with high volumes of images coming day by day during every shifts. The exact position of the inflammation of the lungs has to be shown in the image by detecting the presence of lung opacities. It is also given that all the lung opacities will not be a sign of pneumonia as given in the data set to be “not normal no lung opacity”. This denotes some kind of other abnormalities in the lungs apart from pneumonia. Hence, a robust AI technology for the detection of inflammation in the lungs is necessary.

Objectives of the study

1. To perform Data pre-processing, EDA and Visualisation of the data
2. To construct a pneumonia detection model (Object detection system)
3. To Test the model and to fine tune the processes to improve accuracy using transfer learning.

Data Collection

Data sets are downloaded from the Kaggle website in a competition organised by the Radiological Society of North America (RSNA) pneumonia detection challenge which consists of the train and test data of CRX images. The data set consists of the Patient id, lung x ray images and class of the degree of presence of lung opacities.

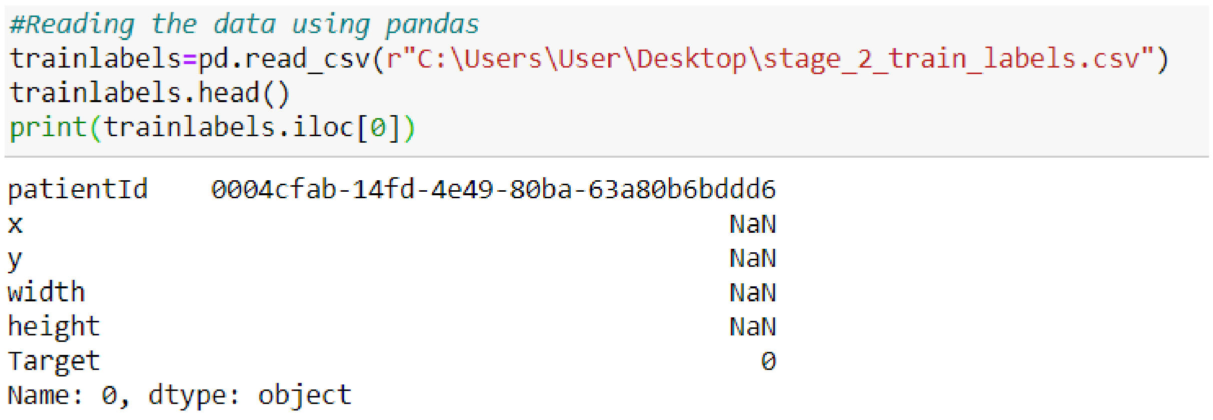
Exploratory Data Analysis (EDA)

1. Read the data into a notebook
2. Import of the necessary libraries
3. To find the shape of the data and type of individual columns
4. Descriptive statistics of the attributes in the data
5. Check the presence of missing values and unique values
6. Data Visualization and distribution of the categorical columns

Several key items in the data set given:

1. Stage_2_detailed_class_info.csv: CSV file containing the training set patient ids and labels of the class.
2. stage_2_sample_submission.csv: CSV file containing the patient ids and the prediction strings.
3. stage_2_train_labels.csv: CSV file containing the patient ids with x.y, width, height and targets involved.
4. Stage_2_test images and Stage_2_train images: Contains Medical X ray images of the lungs which is a DCM file type

Data Description

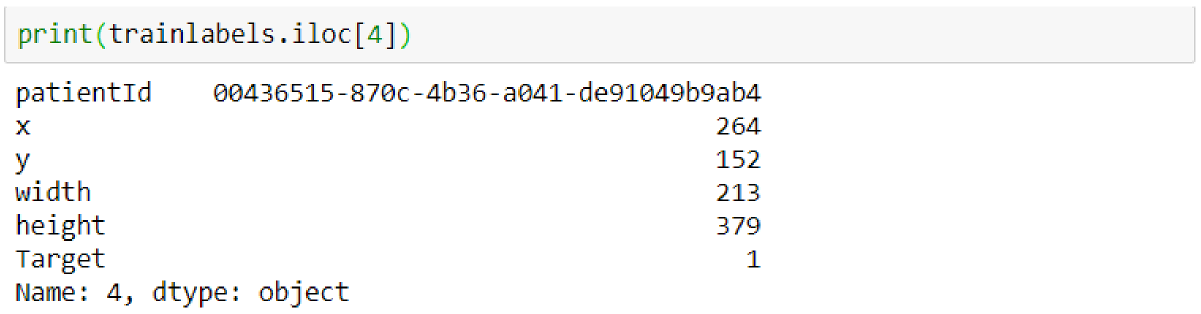


```
#Reading the data using pandas
trainlabels=pd.read_csv(r"C:\Users\User\Desktop\stage_2_train_labels.csv")
trainlabels.head()
print(trainlabels.iloc[0])
```

```
patientId    0004cfab-14fd-4e49-80ba-63a80b6bddd6
x                                             NaN
y                                             NaN
width                                         NaN
height                                        NaN
Target                                          0
Name: 0, dtype: object
```

From the above, we can see that each row in the CSV file contains a patient Id (one unique value per patient), a target (either 0 or 1 for absence or presence of pneumonia, respectively) and the corresponding abnormality bounding box defined by the upper-left hand corner (x, y) coordinate and its corresponding width and height. In this particular case, the patient does *not* have pneumonia and so the corresponding bounding box information is set to NaN.

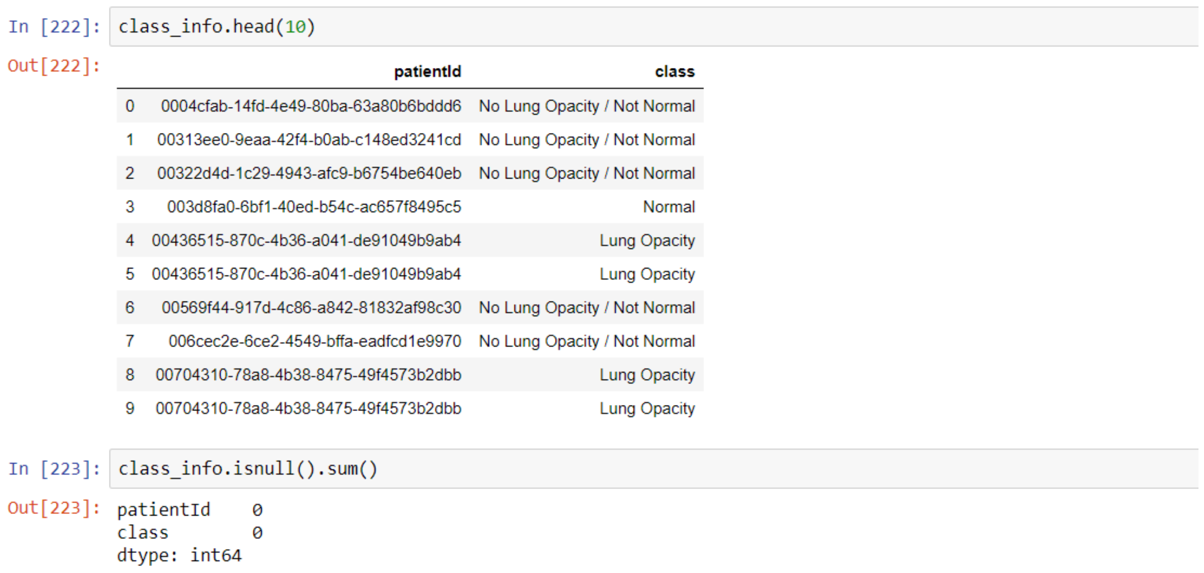
The below code represents a patient with pneumonia as the target of 1 implies



```
print(trainlabels.iloc[4])
```

```
patientId    00436515-870c-4b36-a041-de91049b9ab4
x                                             264
y                                             152
width                                         213
height                                        379
Target                                          1
Name: 4, dtype: object
```

Null values and information about the data



```
In [222]: class_info.head(10)
```

Out[222]:

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
5	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity
6	00569f44-917d-4c86-a842-81832af98c30	No Lung Opacity / Not Normal
7	006cec2e-6ce2-4549-bffa-eadfcd1e9970	No Lung Opacity / Not Normal
8	00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity
9	00704310-78a8-4b38-8475-49f4573b2dbb	Lung Opacity

```
In [223]: class_info.isnull().sum()

Out[223]: patientId    0
          class        0
          dtype: int64
```

There are no null values present in the data and the class corresponding to the patient id shows whether they have pneumonia or not.

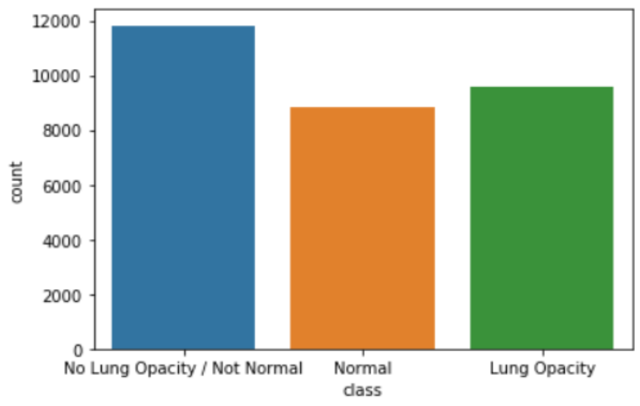
Normal- No defect in the lungs (Absence of pneumonia)

Lung opacity – Lungs affected by pneumonia

No lung opacity/Not Normal – Defective lung but not by pneumonia (may be due to some other abnormalities in the lungs)

Count plot for the number of cases:

```
In [224]: sns.countplot(x='class', data=class_info)
Out[224]: <matplotlib.axes._subplots.AxesSubplot at 0x13b99bd5f48>
```

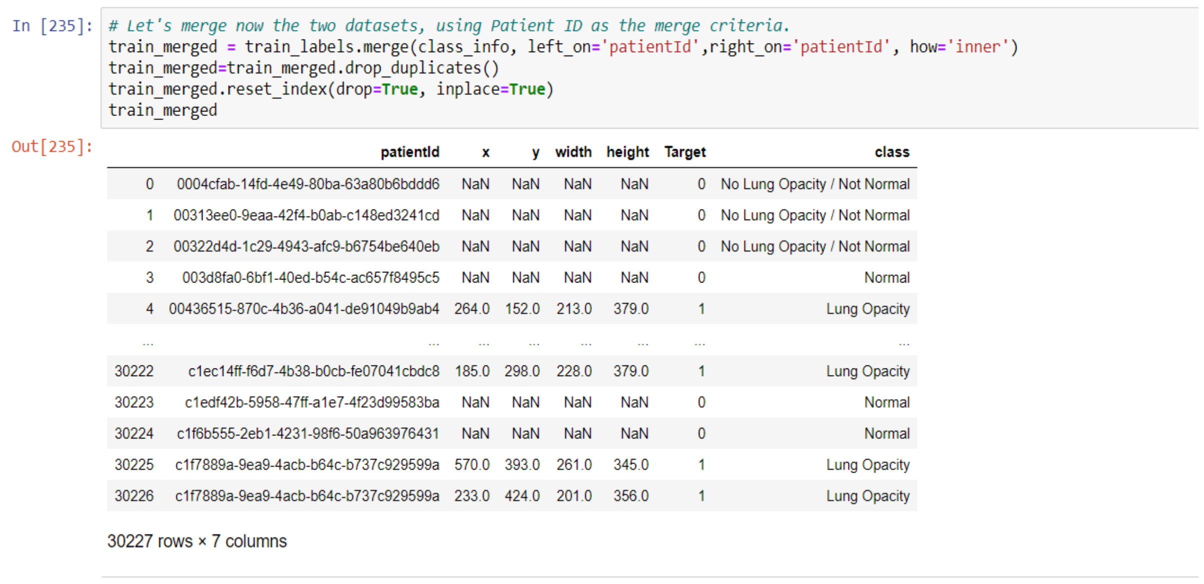


```
In [225]: class_info['class'].value_counts(dropna=False)
Out[225]: No Lung Opacity / Not Normal    11821
          Lung Opacity                     9555
          Normal                           8851
          Name: class, dtype: int64
```

From the plot, it is evident that there are 9555 people affected by pneumonia and 11821 people affected by some other abnormalities in the lungs.

Merge of the data sets

The two csv files stage_2_trainlabels and stage_2_detailedclass_info are merged together based on the patient id as a criteria as given below. In detailed class info dataset are given the detailed information about the type of positive or negative class associated with a certain patient. In Train labels dataset are given the patient ID and the window (x min, y min, width and height of the) containing evidence of pneumonia.



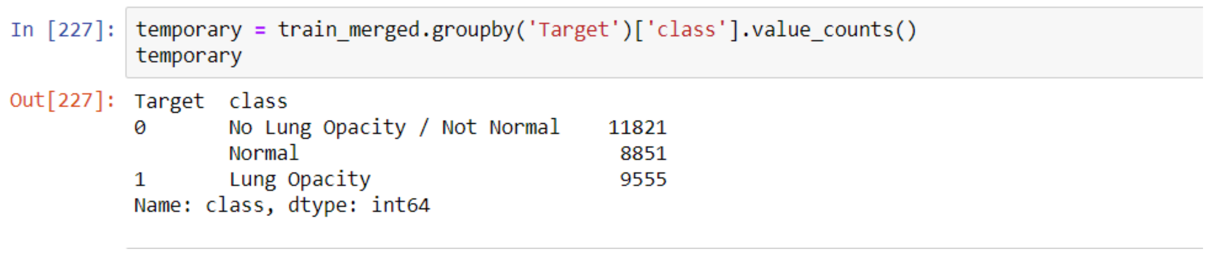
```
In [235]: # Let's merge now the two datasets, using Patient ID as the merge criteria.
          train_merged = train_labels.merge(class_info, left_on='patientId',right_on='patientId', how='inner')
          train_merged=train_merged.drop_duplicates()
          train_merged.reset_index(drop=True, inplace=True)
          train_merged
```

Out[235]:

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

30227 rows × 7 columns

The merged data set is grouped together based on the targets given to the patients as 0 and 1 where 0 is for No lung opacity/ Not normal (Other abnormalities) and 1 is for lung opacity which shows the sign for existence of pneumonia. The columns are grouped as follows.



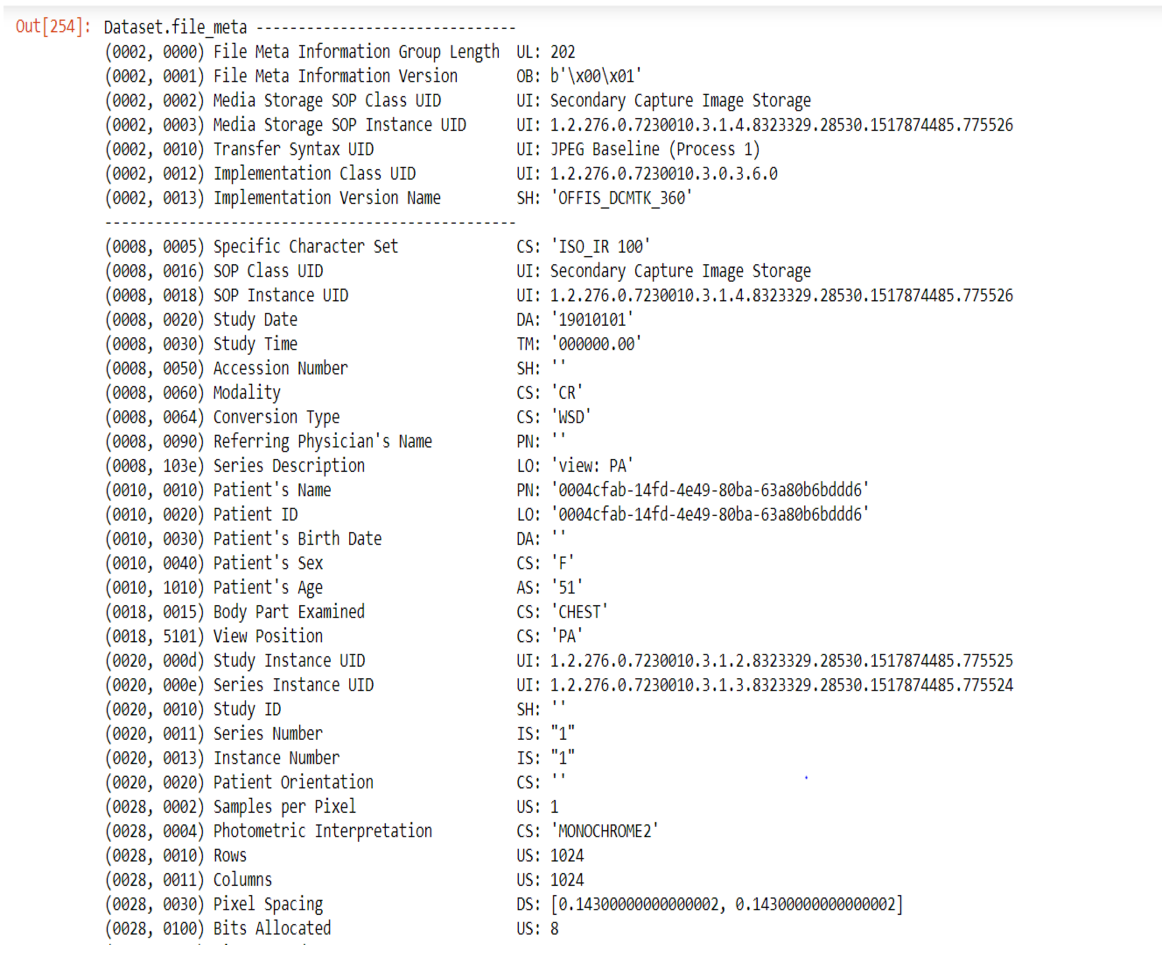
```
In [227]: temporary = train_merged.groupby('Target')['class'].value_counts()
          temporary

Out[227]: Target  class
          0       No Lung Opacity / Not Normal    11821
                  Normal                           8851
          1       Lung Opacity                     9555
          Name: class, dtype: int64
```

All chest examinations with Target = 1 (pathology detected) associated with class: Lung Opacity. The chest examinations with Target = 0 (no pathology detected) are either of class: Normal or class: No Lung Opacity / Not Normal.

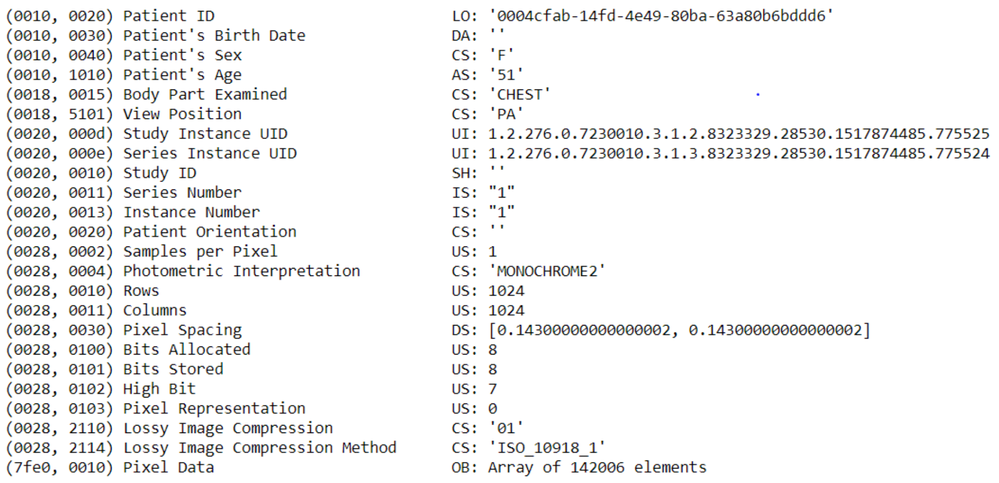
Overview of the DICOM files and Medical Images

Medical images are stored in a special format known as DICOM files (*.dcm). These type of files contain a combination of header meta data along with the underlying raw image arrays for the pixel data available. They contain a combination of header metadata as well as underlying raw image arrays for pixel data. In Python, one popular library to access and manipulate DICOM files is the pydicom module. To use the pydicom library, we have to first find out the DICOM file for a given patient Id by simply looking for the matching file in the stage_2_train_images file, and the use the pydicom.read_ file() method to load the data.

```
dcm_data = pydicom.dcmread(train_img[0])
dcm_data
```



```
Out[254]: Dataset.file_meta -------------------------------
          (0002, 0000) File Meta Information Group Length  UL: 202
          (0002, 0001) File Meta Information Version       OB: b'\x00\x01'
          (0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
          (0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
          (0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
          (0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
          (0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
          -------------------------------------------------
          (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: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
          (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: PA'
          (0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
          (0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
          (0010, 0030) Patient's Birth Date                DA: ''
          (0010, 0040) Patient's Sex                       CS: 'F'
          (0010, 1010) Patient's Age                       AS: '51'
          (0018, 0015) Body Part Examined                  CS: 'CHEST'
          (0018, 5101) View Position                       CS: 'PA'
          (0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
          (0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
          (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: [0.14300000000000002, 0.14300000000000002]
          (0028, 0100) Bits Allocated                      US: 8
```



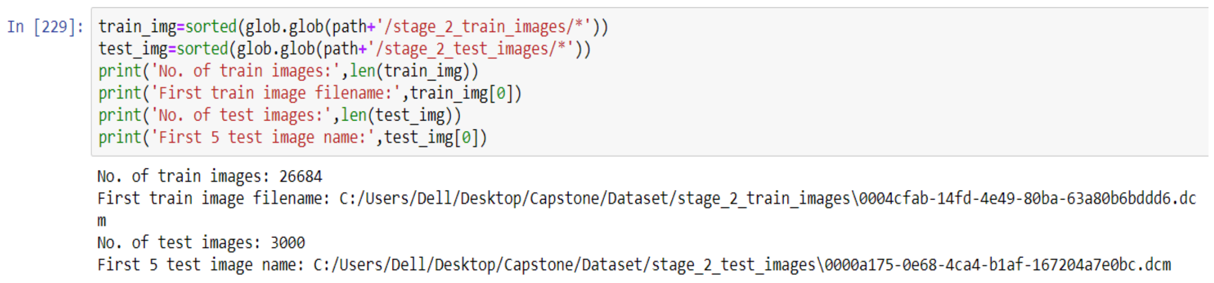
```
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(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: [0.14300000000000002, 0.14300000000000002]
(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 142006 elements
```

We can observe that we do have available some useful information in the DICOM metadata with predictive value such as Patient sex, Patient age, Modality, Body part examined, Positional view, Rows and columns and Pixel spacing.

Most of the standard headers containing patient identifiable information have been anonymized (removed) so we are left with a relatively sparse set of metadata.

Exploration of the DICOM data

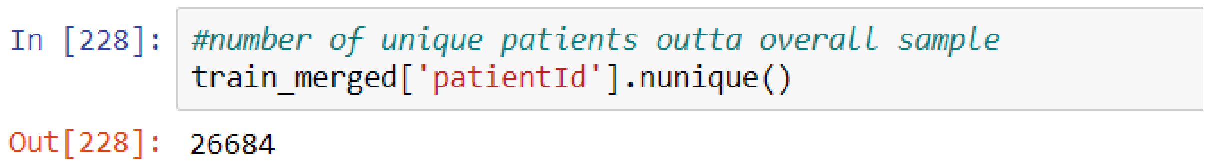
Let's check how many images are in the train and test folders containing the images of the lungs of the patients.



```
In [229]: train_img=sorted(glob.glob(path+'/stage_2_train_images/*'))
          test_img=sorted(glob.glob(path+'/stage_2_test_images/*'))
          print('No. of train images:',len(train_img))
          print('First train image filename:',train_img[0])
          print('No. of test images:',len(test_img))
          print('First 5 test image name:',test_img[0])

          No. of train images: 26684
          First train image filename: C:/Users/Dell/Desktop/Capstone/Dataset/stage_2_train_images\0004cfab-14fd-4e49-80ba-63a80b6bddd6.dc
          m
          No. of test images: 3000
          First 5 test image name: C:/Users/Dell/Desktop/Capstone/Dataset/stage_2_test_images\0000a175-0e68-4ca4-b1af-167204a7e0bc.dcm
```

There are 26684 images in the train image set and 3000 images present in the test image set.

To check the presence of duplicates in the train data set

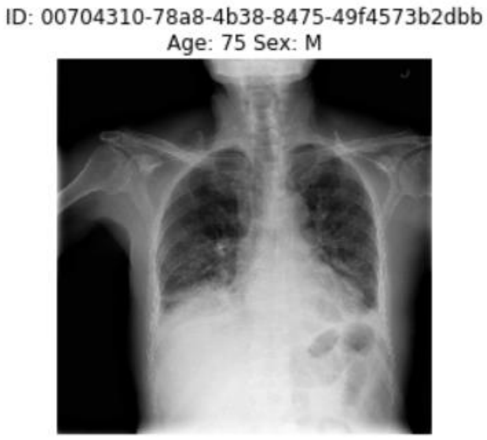


```
In [228]: #number of unique patients outta overall sample
          train_merged['patientId'].nunique()

Out[228]: 26684
```

We confirmed that the number of *unique* patients Id are equal with the number of DICOM images in the train set. It is evident that there are 26684 patient ids which are equal to the number of train images as given as explained from the code above.

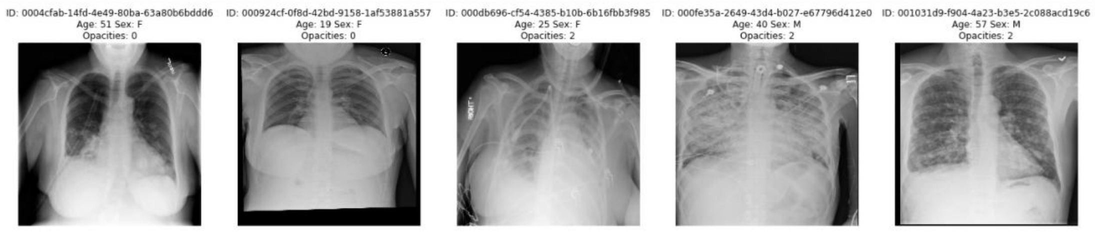
DCM image is plotted now using the age, sex and ID of the patient as given below.

```
In [193]: ax=plt.gca()
          pid=dcm_data.PatientID
          age = dcm_data.PatientAge
          sex = dcm_data.PatientSex
          plt.imshow(im)
          plt.axis("off")
          ax.set_title('ID: {}\nAge: {} Sex: {}'.format(pid,age, sex))

Out[193]: Text(0.5, 1.0, 'ID: 00704310-78a8-4b38-8475-49f4573b2dbb\nAge: 75 Sex: M')
```

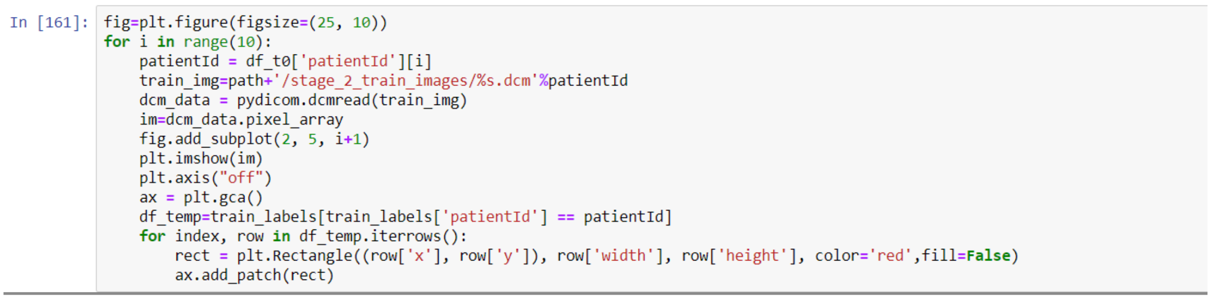


Then the merged data set consisting of the classes of abnormalities in the lungs is given as the criteria in the differentiation of the dicom images of the patients as given in the code below which denotes the number of lung opacities in the patients through the images. These are based on the Patient id, age , sex and the classes.

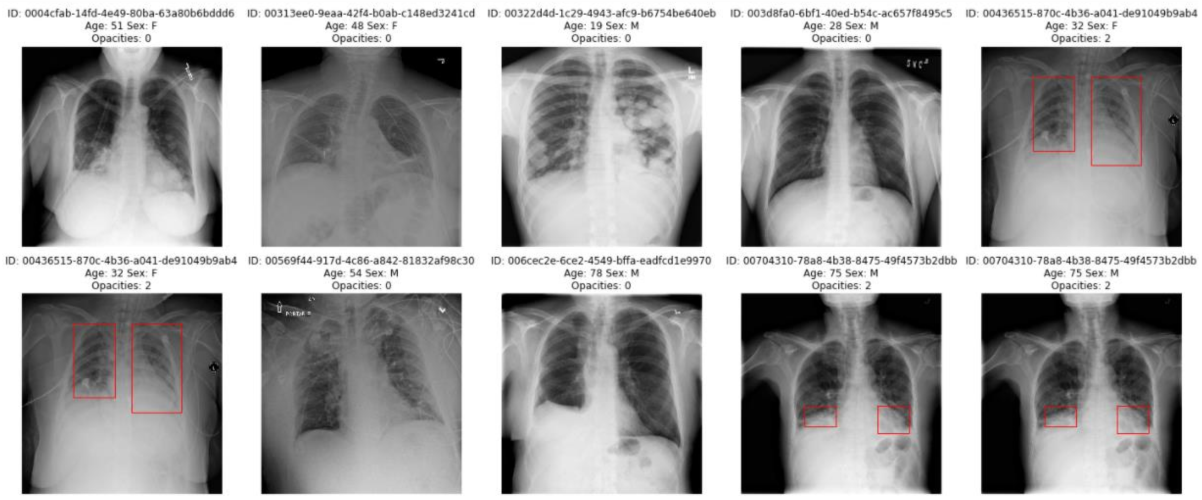
```
In [195]: fig=plt.figure(figsize=(25, 12))
          for i in range(0,10):
              dcm_data = pydicom.dcmread(train_img[i])
              im=dcm_data.pixel_array
              fig.add_subplot(2, 5, i+1)
              ax=plt.gca()
              stat=train_merged[train_merged.patientId==dcm_data.PatientID]["Target"].sum()
              pid=dcm_data.PatientID
              age = dcm_data.PatientAge
              sex = dcm_data.PatientSex
              plt.imshow(im)
              plt.axis("off")
              ax.set_title('ID: {}\nAge: {} Sex: {}\nOpacities: {}'.format(pid,age,sex,stat))
```


ID: 0004cfab-14fd-4e49-80ba-63a80b6bddd6
Age: 51 Sex: F
Opacities: 0
ID: 000924cf-0f8d-42bd-9158-1af53881a557
Age: 19 Sex: F
Opacities: 0
ID: 000db696-cf54-4385-b10b-6b16fbb3f985
Age: 25 Sex: F
Opacities: 2
ID: 000fe35a-2649-43d4-b027-e67796d412e0
Age: 40 Sex: M
Opacities: 2
ID: 001031d9-f904-4a23-b3e5-2c088acd19c6
Age: 57 Sex: M
Opacities: 2

Plot of the rectangular bounding boxes for the existence of the lung opacities which shows a target of 1.



```
In [161]: fig=plt.figure(figsize=(25, 10))
          for i in range(10):
              patientId = df_t0['patientId'][i]
              train_img=path+'/stage_2_train_images/%s.dcm'%patientId
              dcm_data = pydicom.dcmread(train_img)
              im=dcm_data.pixel_array
              fig.add_subplot(2, 5, i+1)
              plt.imshow(im)
              plt.axis("off")
              ax = plt.gca()
              df_temp=train_labels[train_labels['patientId'] == patientId]
              for index, row in df_temp.iterrows():
                  rect = plt.Rectangle((row['x'], row['y']), row['width'], row['height'], color='red',fill=False)
                  ax.add_patch(rect)
```


ID: 0004cfab-14fd-4e49-80ba-63a80b6bddd6
Age: 51 Sex: F
Opacities: 0
ID: 00313ee0-9eaa-42f4-b0ab-c148ed3241cd
Age: 48 Sex: F
Opacities: 0
ID: 00322d4d-1c29-4943-afc9-b6754be640eb
Age: 19 Sex: M
Opacities: 0
ID: 003d8fa0-6bf1-40ed-b54c-ac657f8495c5
Age: 28 Sex: M
Opacities: 0
ID: 00436515-870c-4b36-a041-de91049b9ab4
Age: 32 Sex: F
Opacities: 2
ID: 00436515-870c-4b36-a041-de91049b9ab4
Age: 32 Sex: F
Opacities: 2
ID: 00569f44-917d-4c86-a842-81832af98c30
Age: 54 Sex: M
Opacities: 0
ID: 006cec2e-6ce2-4549-bffa-eadfcd1e9970
Age: 78 Sex: M
Opacities: 0
ID: 00704310-78a8-4b38-8475-49f4573b2dbb
Age: 75 Sex: M
Opacities: 2
ID: 00704310-78a8-4b38-8475-49f4573b2dbb
Age: 75 Sex: M
Opacities: 2

Building up a model

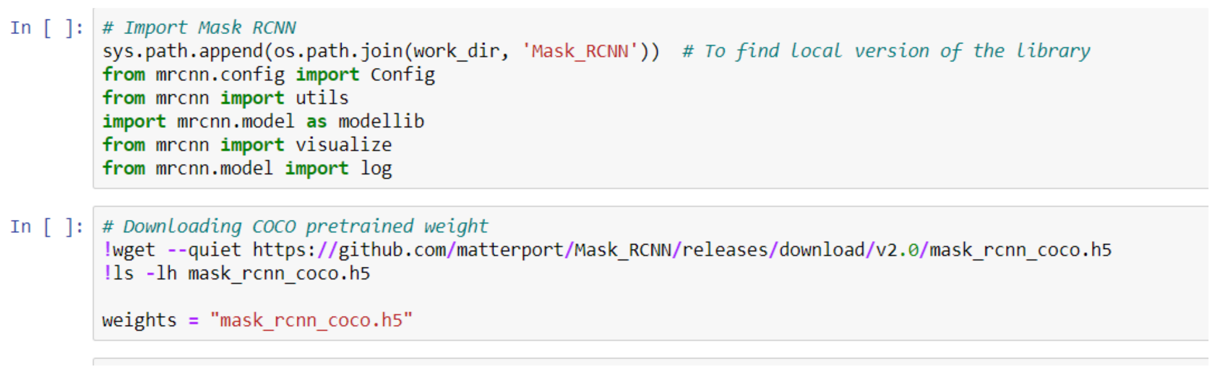
Mask-RCNN model is employed for the detection of pneumonia along with the process of transfer learning. The type of weights that are used are COCO weights which is a large scale object detection, segmentation and captioning of the data sets. The pre trained COCO weights are used in this process.

Mask- RCNN model

Mask RCNN is a type of deep neural network which is designed and aimed at solving segmentation problems in computer vision as well as machine learning. It is used for the separation of different types of objects in any given type of an image or a video. Incase of an image, this model helps in giving out the object bounding boxes, classes and the masks. In this case this model is used for the identification of the extent of inflammation in the lungs and the other sorts of abnormalities with the help of the bounding boxes as a part of identification and segmentation modelling.

There are two stages present in the Mask RCNN model. First step is that it helps in the generation of the proposals about the presence of regions where there may be an object shown based on the input image given. The next step would be, it will also predict the class of the object and refines the bounding boxes which leads to the generation of the mask in the level of pixels of the detected object based on the first step and both the steps would be interconnected to each other.

The weights for the pre trained model is downloaded in which the Mask R-CNN model is trained on the MS coco Datasets. Finally, we will use the Mask R-CNN architecture and the pretrained weights to generate predictions for our own images.



```
In [ ]: # Import Mask RCNN
        sys.path.append(os.path.join(work_dir, 'Mask_RCNN'))  # To find local version of the library
        from mrcnn.config import Config
        from mrcnn import utils
        import mrcnn.model as modellib
        from mrcnn import visualize
        from mrcnn.model import log
```

```
In [ ]: # Downloading COCO pretrained weight
        !wget --quiet https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5
        !ls -lh mask_rcnn_coco.h5

        weights = "mask_rcnn_coco.h5"
```

On experimenting with 2 epochs, we got the following results,

Rpn_class_loss : 0.028

Rpn_bbox_loss : 0.5910

Mrcnn_class_loss: 0.2778

Mrcnn_bbox_loss: 0.5423

Mrcnn_mask_loss: 0.3967

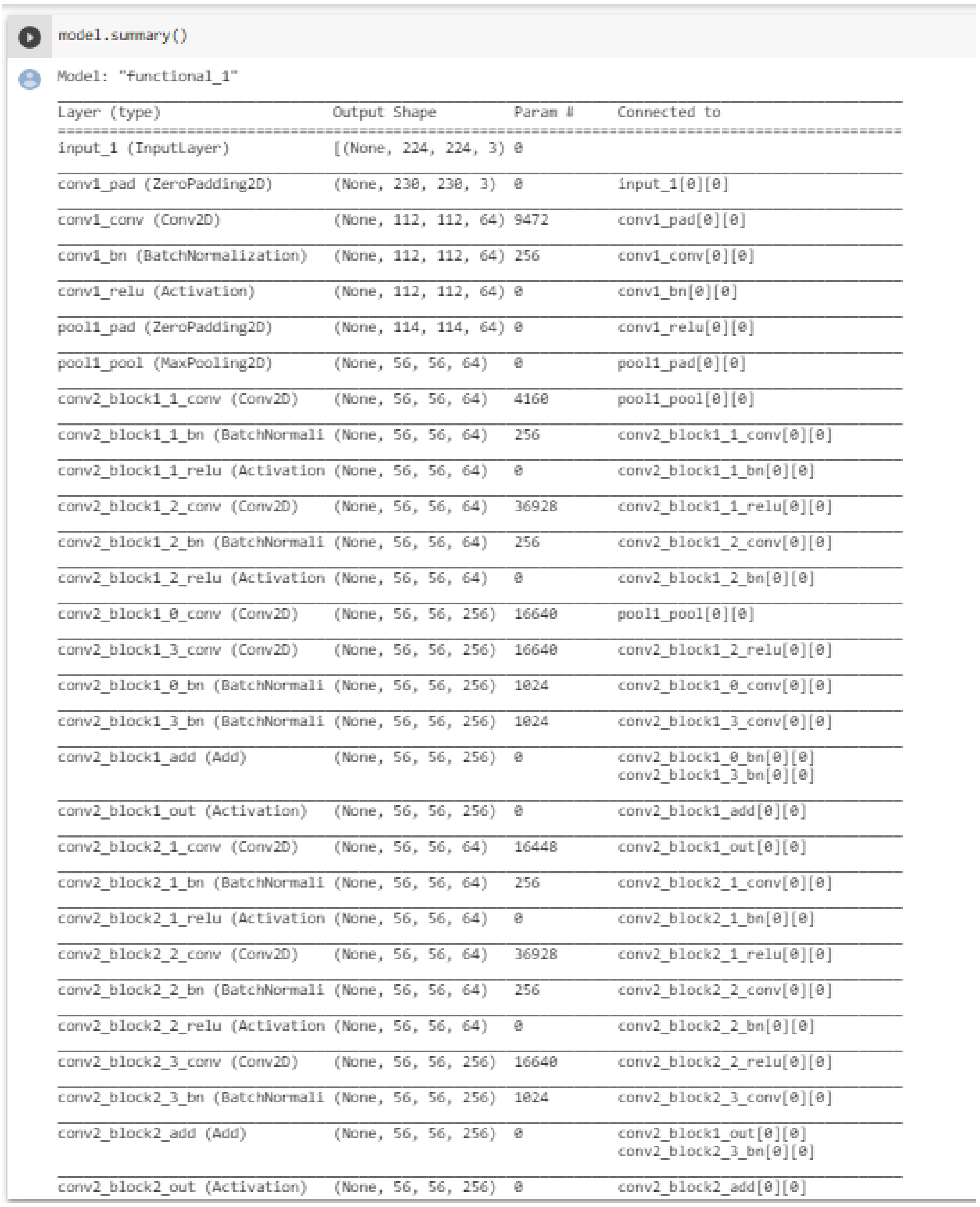
We could observe that the bounding box loss with masked RCNN is around 0.5 and has to be improved further. This loss depicts the difference in the bounding boxes that are predicted to the ground truth.

Resnet Model

Residual Network (ResNet) is a Convolutional Neural Network (CNN) architecture which was designed to enable hundreds or thousands of convolutional layers. While previous CNN architectures had a drop off in the effectiveness of additional layers, ResNet can add a large number of layers with strong performance.

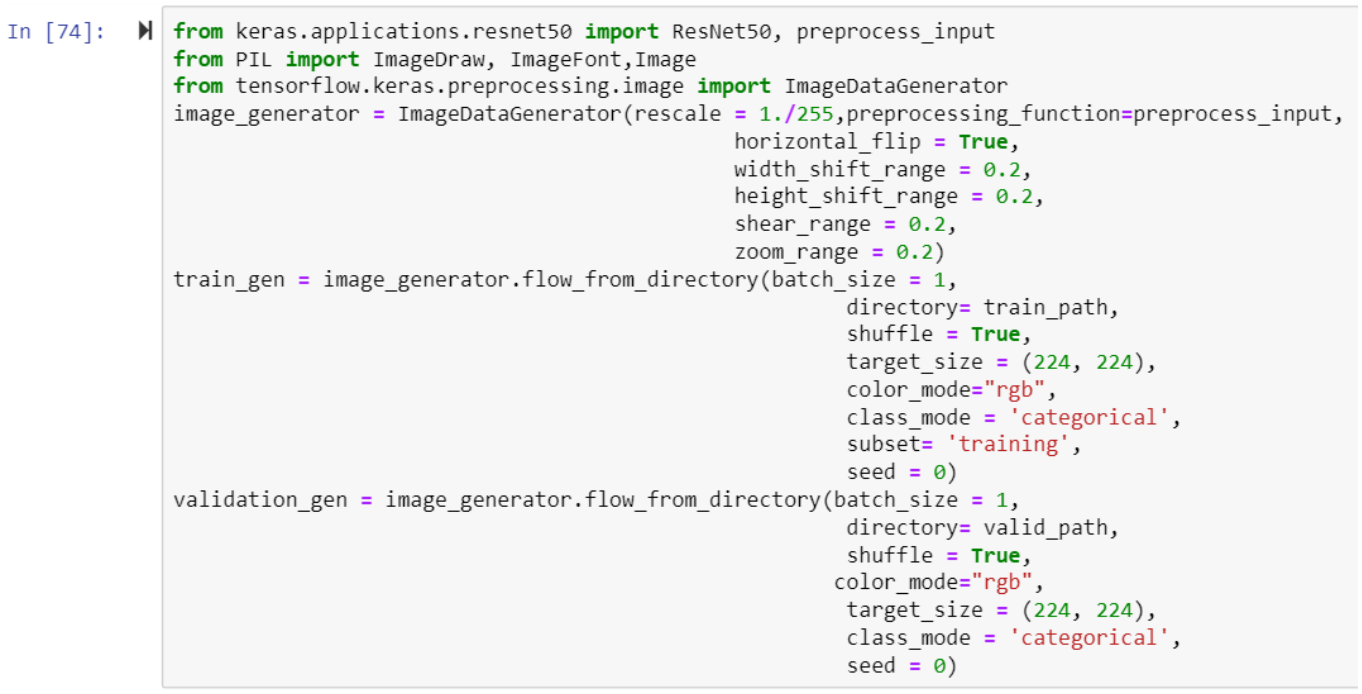
ResNet was an innovative solution to the “vanishing gradient” problem. Neural networks train via the backpropagation process (see our guide on backpropagation), which relies on gradient descent, moving down the loss function to find the weights that minimize it. If there are too many layers, repeated multiplication makes the gradient smaller and smaller, until it “disappears”, causing performance to saturate or even degrade with each additional layer.

The ResNet solution is “identity shortcut connections”. ResNet stacks up identity mappings, layers that initially don’t do anything, and skips over them, reusing the activations from previous layers. Skipping initially compresses the network into only a few layers, which enables faster learning. Then, when the network trains again, all layers are expanded and the “residual” parts of the network explore more and more of the feature space of the source image.

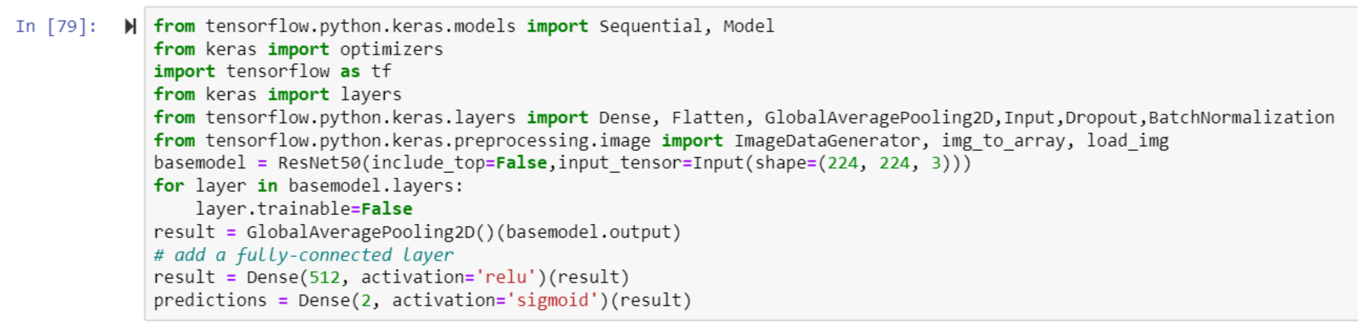


```
model.summary()
```

```
Model: "functional_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_conv (Conv2D)            (None, 112, 112, 64) 9472        conv1_pad[0][0]
___________________________________________________________________________________________________
conv1_bn (BatchNormalization)  (None, 112, 112, 64) 256         conv1_conv[0][0]
___________________________________________________________________________________________________
conv1_relu (Activation)        (None, 112, 112, 64) 0           conv1_bn[0][0]
___________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)      (None, 114, 114, 64) 0           conv1_relu[0][0]
___________________________________________________________________________________________________
pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           pool1_pad[0][0]
___________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160        pool1_pool[0][0]
___________________________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)  256         conv2_block1_1_conv[0][0]
___________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)  0           conv2_block1_1_bn[0][0]
___________________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36928       conv2_block1_1_relu[0][0]
___________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)  256         conv2_block1_2_conv[0][0]
___________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)  0           conv2_block1_2_bn[0][0]
___________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640       pool1_pool[0][0]
___________________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640       conv2_block1_2_relu[0][0]
___________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256) 1024        conv2_block1_0_conv[0][0]
___________________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256) 1024        conv2_block1_3_conv[0][0]
___________________________________________________________________________________________________
conv2_block1_add (Add)         (None, 56, 56, 256)  0           conv2_block1_0_bn[0][0]
                                                                conv2_block1_3_bn[0][0]
___________________________________________________________________________________________________
conv2_block1_out (Activation)  (None, 56, 56, 256)  0           conv2_block1_add[0][0]
___________________________________________________________________________________________________
conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16448       conv2_block1_out[0][0]
___________________________________________________________________________________________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)  256         conv2_block2_1_conv[0][0]
___________________________________________________________________________________________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)  0           conv2_block2_1_bn[0][0]
___________________________________________________________________________________________________
conv2_block2_2_conv (Conv2D)   (None, 56, 56, 64)   36928       conv2_block2_1_relu[0][0]
___________________________________________________________________________________________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)  256         conv2_block2_2_conv[0][0]
___________________________________________________________________________________________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)  0           conv2_block2_2_bn[0][0]
___________________________________________________________________________________________________
conv2_block2_3_conv (Conv2D)   (None, 56, 56, 256)  16640       conv2_block2_2_relu[0][0]
___________________________________________________________________________________________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256) 1024        conv2_block2_3_conv[0][0]
___________________________________________________________________________________________________
conv2_block2_add (Add)         (None, 56, 56, 256)  0           conv2_block1_out[0][0]
                                                                conv2_block2_3_bn[0][0]
___________________________________________________________________________________________________
conv2_block2_out (Activation)  (None, 56, 56, 256)  0           conv2_block2_add[0][0]
```



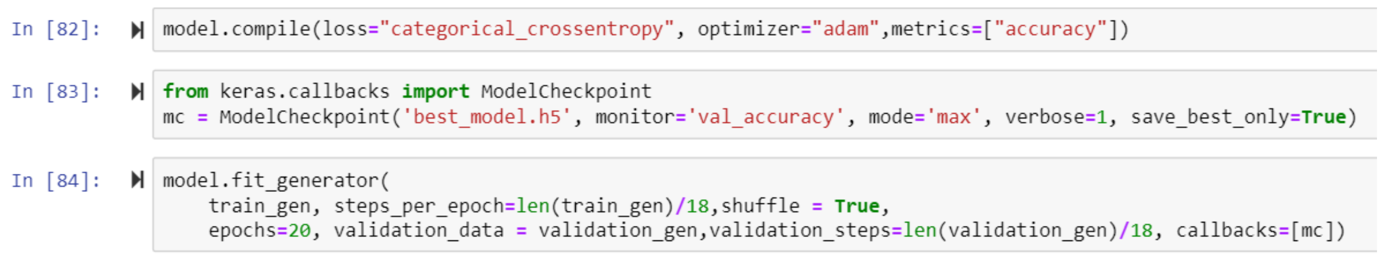
```
In [74]:  from keras.applications.resnet50 import ResNet50, preprocess_input
          from PIL import ImageDraw, ImageFont,Image
          from tensorflow.keras.preprocessing.image import ImageDataGenerator
          image_generator = ImageDataGenerator(rescale = 1./255,preprocessing_function=preprocess_input,
                                               horizontal_flip = True,
                                               width_shift_range = 0.2,
                                               height_shift_range = 0.2,
                                               shear_range = 0.2,
                                               zoom_range = 0.2)
          train_gen = image_generator.flow_from_directory(batch_size = 1,
                                                        directory= train_path,
                                                        shuffle = True,
                                                        target_size = (224, 224),
                                                        color_mode="rgb",
                                                        class_mode = 'categorical',
                                                        subset= 'training',
                                                        seed = 0)
          validation_gen = image_generator.flow_from_directory(batch_size = 1,
                                                        directory= valid_path,
                                                        shuffle = True,
                                                       color_mode="rgb",
                                                        target_size = (224, 224),
                                                        class_mode = 'categorical',
                                                        seed = 0)
```



```
In [79]:  from tensorflow.python.keras.models import Sequential, Model
          from keras import optimizers
          import tensorflow as tf
          from keras import layers
          from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D,Input,Dropout,BatchNormalization
          from tensorflow.python.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
          basemodel = ResNet50(include_top=False,input_tensor=Input(shape=(224, 224, 3)))
          for layer in basemodel.layers:
              layer.trainable=False
          result = GlobalAveragePooling2D()(basemodel.output)
          # add a fully-connected layer
          result = Dense(512, activation='relu')(result)
          predictions = Dense(2, activation='sigmoid')(result)
```

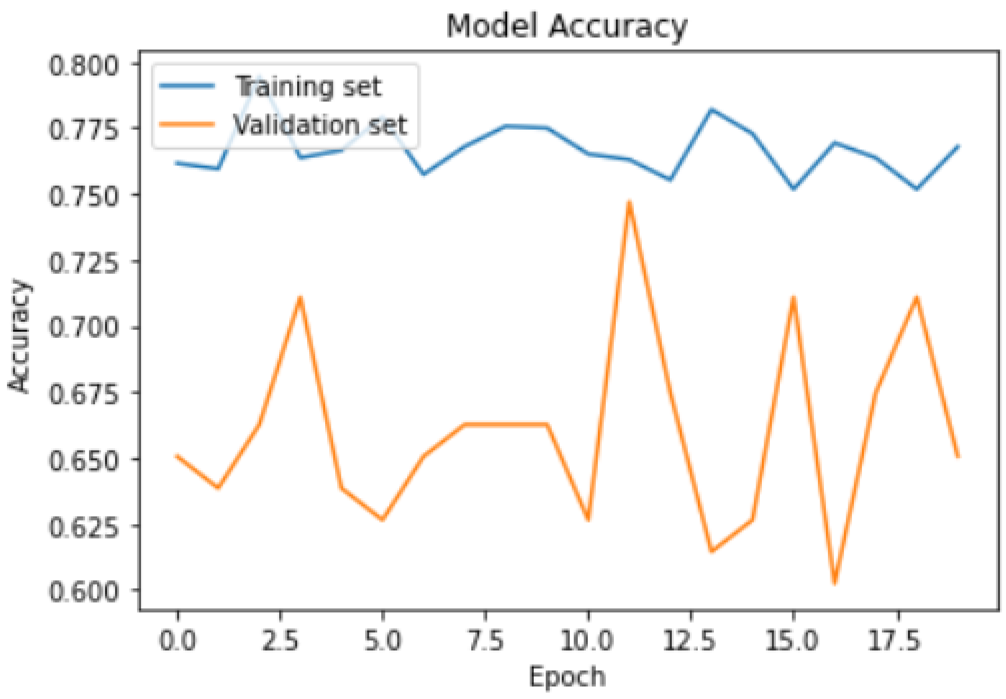
The above code builds the Resnet model and creaters training and validation data sets.Sequencial class are imported which allows one to easily contruct a neural network by just stacking layer one after another.GAP(Global Average Pooling) is used to minimize the overfitting by reducing the total number of parameters in the model.

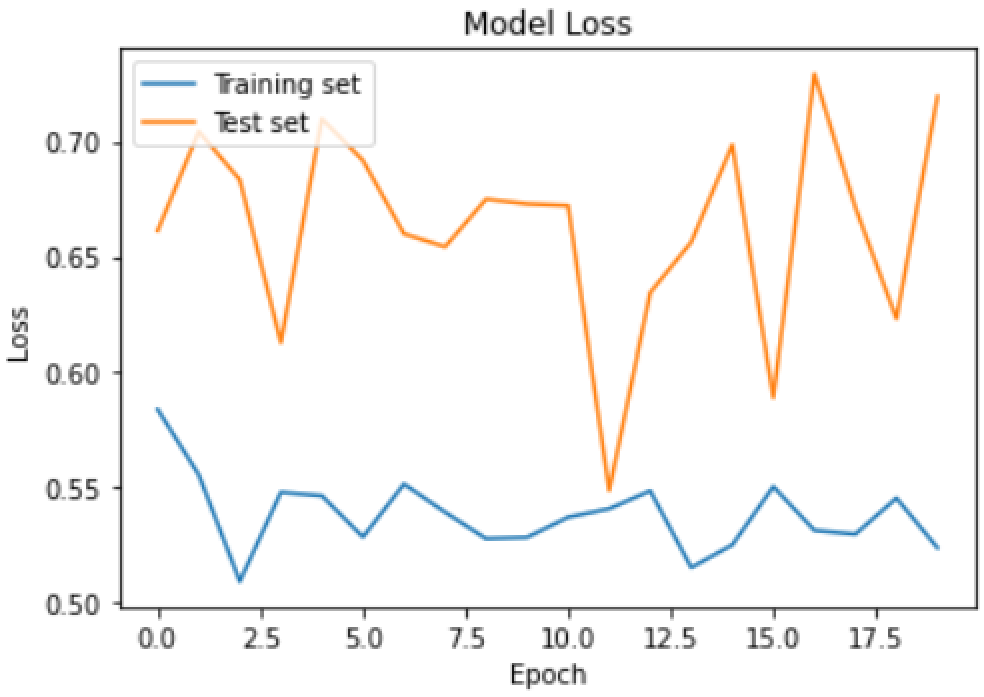
Now model is compiled and fitted using epoch-20 and callbacks-checkpoint.The model is optimized using adam alogorithm which combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.



```
In [82]:  model.compile(loss="categorical_crossentropy", optimizer="adam",metrics=["accuracy"])

In [83]:  from keras.callbacks import ModelCheckpoint
          mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)

In [84]:  model.fit_generator(
              train_gen, steps_per_epoch=len(train_gen)/18,shuffle = True,
              epochs=20, validation_data = validation_gen,validation_steps=len(validation_gen)/18, callbacks=[mc])
```

Plot of Model Accuracy and Model loss:

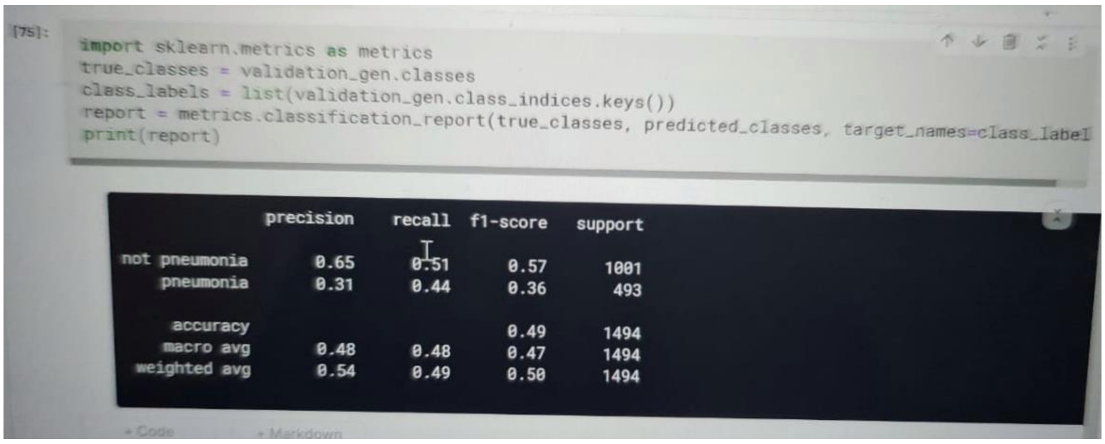




As the epoch is increased, the fluctuation/noise in the validation set gradually reduces and thereafter it is able to predict with moreover consistent accuracy of around 67%. Whereas the maximum possible accuracy observed was 71%.

The training loss is moreover constant which implies model is fully learnt and no further improvement is expected.

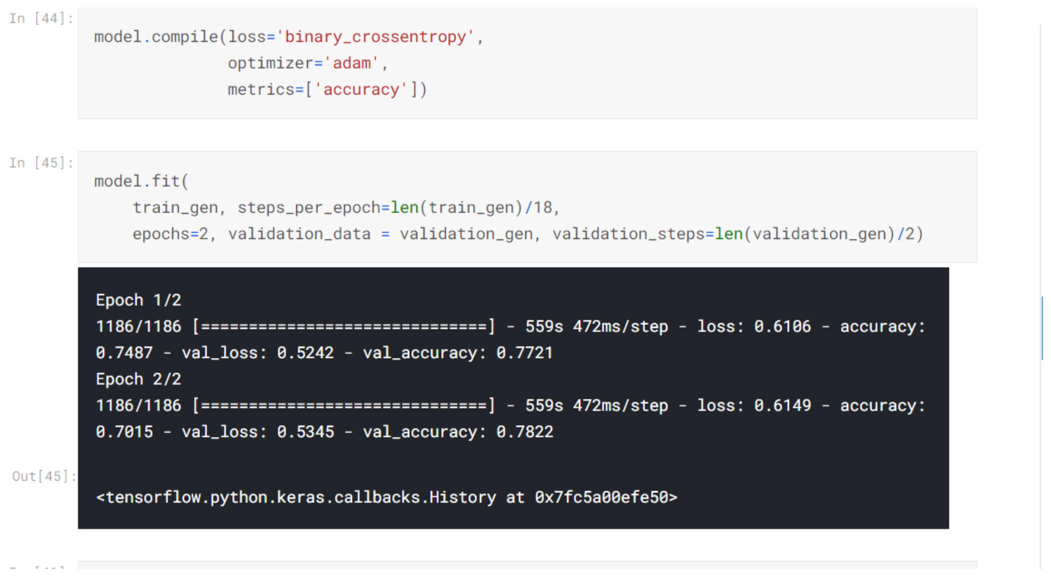
The confusion matrix was plotted which revealed a precision 0.65, 0.31 for classes 0 and 1, recall 0.51 and 0.44 for classes 0 and 1 respectively.



```
[75]:
import sklearn.metrics as metrics
true_classes = validation_gen.classes
class_labels = list(validation_gen.class_indices.keys())
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_label
print(report)
```

```
               precision    recall  f1-score   support

not pneumonia       0.65      0.51      0.57      1001
    pneumonia       0.31      0.44      0.36       493

     accuracy                           0.49      1494
    macro avg       0.48      0.48      0.47      1494
 weighted avg       0.54      0.49      0.50      1494
```

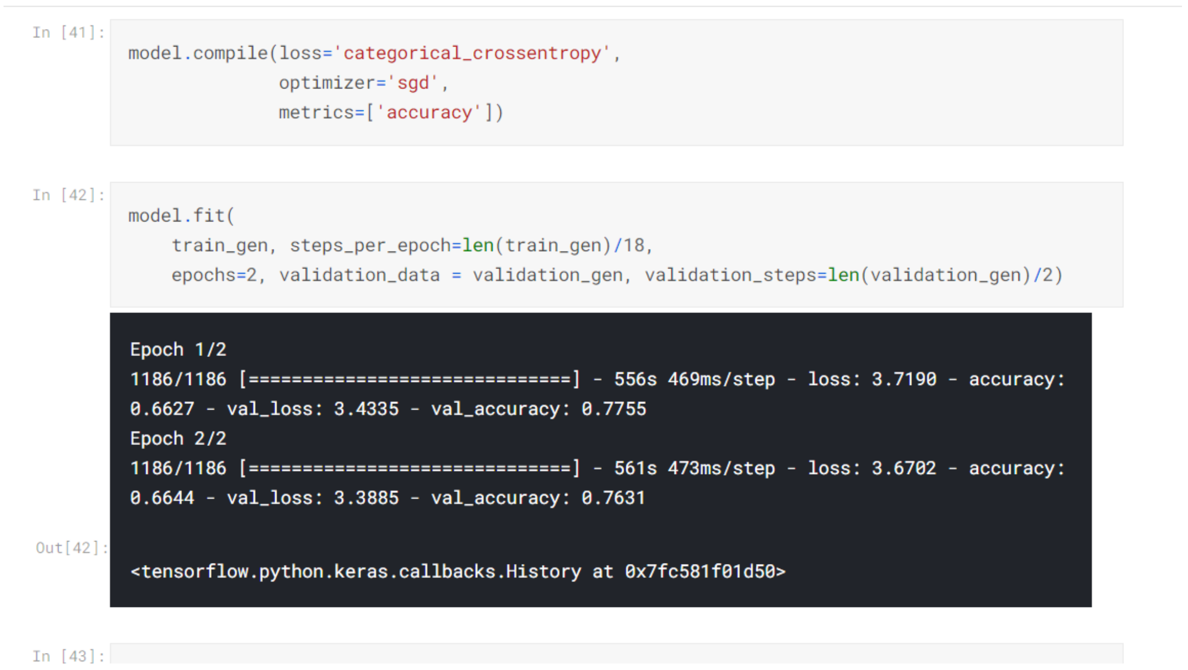
We also trained the model with combinations of optimizers and loss (among ‘adam’, ‘sgd’; loss being ‘categorical_cross entropy’ and binary_cross entropy) for few epochs. The accuracy for validation set was compared and the best combination of ‘adam’ optimizer with loss being ‘categorical_cross entropy’ was employed.



```
In [44]:
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
In [45]:
model.fit(
    train_gen, steps_per_epoch=len(train_gen)/18,
    epochs=2, validation_data = validation_gen, validation_steps=len(validation_gen)/2)
```

```
Epoch 1/2
1186/1186 [==============================] - 559s 472ms/step - loss: 0.6106 - accuracy:
0.7487 - val_loss: 0.5242 - val_accuracy: 0.7721
Epoch 2/2
1186/1186 [==============================] - 559s 472ms/step - loss: 0.6149 - accuracy:
0.7015 - val_loss: 0.5345 - val_accuracy: 0.7822
Out[45]:
<tensorflow.python.keras.callbacks.History at 0x7fc5a00efe50>
```

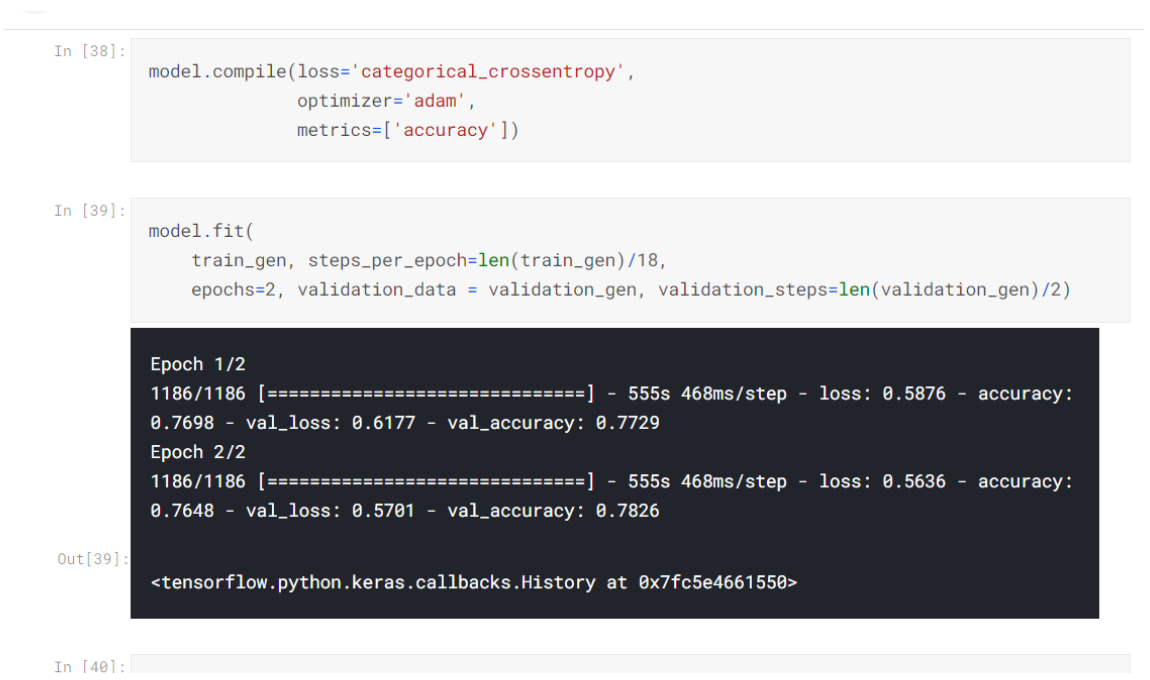


```
In [41]:
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

```
In [42]:
model.fit(
    train_gen, steps_per_epoch=len(train_gen)/18,
    epochs=2, validation_data = validation_gen, validation_steps=len(validation_gen)/2)
```

```
Epoch 1/2
1186/1186 [==============================] - 556s 469ms/step - loss: 3.7190 - accuracy:
0.6627 - val_loss: 3.4335 - val_accuracy: 0.7755
Epoch 2/2
1186/1186 [==============================] - 561s 473ms/step - loss: 3.6702 - accuracy:
0.6644 - val_loss: 3.3885 - val_accuracy: 0.7631
Out[42]:
<tensorflow.python.keras.callbacks.History at 0x7fc581f01d50>
```

```
In [43]:
```



```
In [38]:
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
In [39]:
model.fit(
    train_gen, steps_per_epoch=len(train_gen)/18,
    epochs=2, validation_data = validation_gen, validation_steps=len(validation_gen)/2)
```

```
Epoch 1/2
1186/1186 [==============================] - 555s 468ms/step - loss: 0.5876 - accuracy:
0.7698 - val_loss: 0.6177 - val_accuracy: 0.7729
Epoch 2/2
1186/1186 [==============================] - 555s 468ms/step - loss: 0.5636 - accuracy:
0.7648 - val_loss: 0.5701 - val_accuracy: 0.7826

Out[39]:
<tensorflow.python.keras.callbacks.History at 0x7fc5e4661550>

In [40]:
```

Summary:

We approached the problem starting with an exploratory data analysis of the CSV files and DICOM images. The training set 9555 belonging to class pneumonia and remaining belonging to normal and non-pneumonic abnormalities.

We tried two models: A mask RCNN model and ResNet based classification model

Mask RCNN model was built with pretrained weights from coco-model weights. On evaluation the bounding box error was around 0.5. A classification model was built with ResNet layers and pretrained weights from 'ImageNet'. The later model was decided to be taken further for finetuning and evaluation.

The model was trained with combinations of optimizers and loss (among 'adam', 'sgd'; loss being 'categorical_crossentropy' and binary_crossentropy)

The parameters were chosen as with 'adam' as optimizer and loss being 'categorical_crossentropy', since it gave the best validation accuracy among the combinations.

The prediction value for validation set was as well calculated and saved to a CSV file.