Great Learning

AIML Online Capstone - Pneumonia Detection Challenge – Group 7

Contributors: Sanket Kadam, Gaurav Shankar, Paras Ashra, Karthik S, Wasim Hassan

Mentor: Vibha

Table of Contents

Problem statement

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. Pneumonia usually manifests as an area or areas of increased opacity on CXR. However, the diagnosis ofpneumonia 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.

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.

Pneumonia Detection

Now to detection Pneumonia we need to detect **Inflammation** of the lungs. In this project, you're challenged to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically,your algorithm needs to automatically locate lung opacities on chest radiographs.

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).

Guided by relevant clinical questions, powerful AI techniques can unlock clinically relevant information hidden in the massive amount of data, which in turn can assist clinical decision making.

Recent Pandemic – Coronavirus

We have seen the recent outbreak of COVID-19 a.k.a the coronavirus, similar to the spread of spanish influenza in 1900s. Pneumonia is an infection of the lungs. Viruses, bacteria, and fungi can cause it. Pneumonia can cause the small air sacs in your lungs, known as alveoli, to fill with fluid. Pneumonia can be a complication of COVID-19, the illness caused by the new coronavirus known as SARS-CoV-2.

It was desirable to develop an automatic and accurate detection of COVID-19 using chest CT. Thus, purpose was to develop a fully automatic framework to detect COVID-19 using chest CT and evaluate its performance. The challenge here would be to aid the diagnosis process which allows for expedited treatment and better clinical outcomes

Figure 1.0: Pneumonia diagram

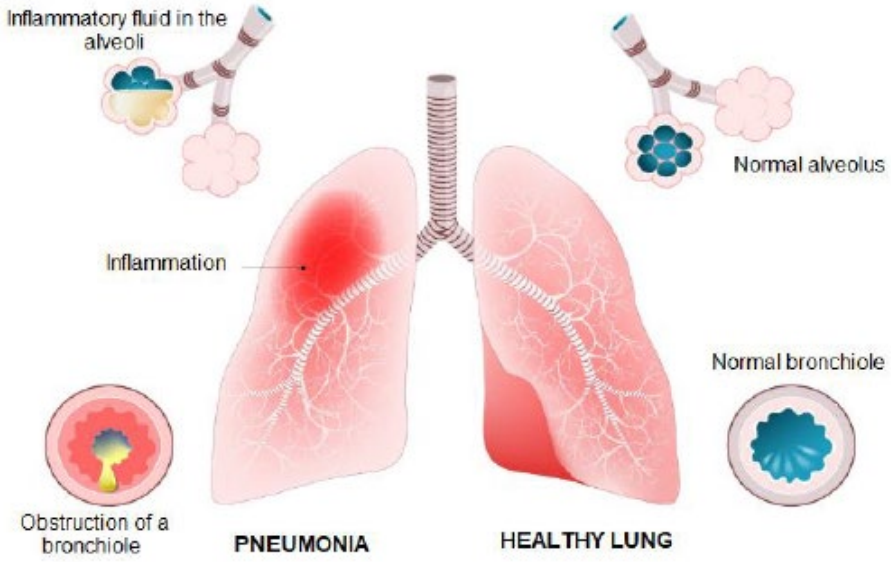
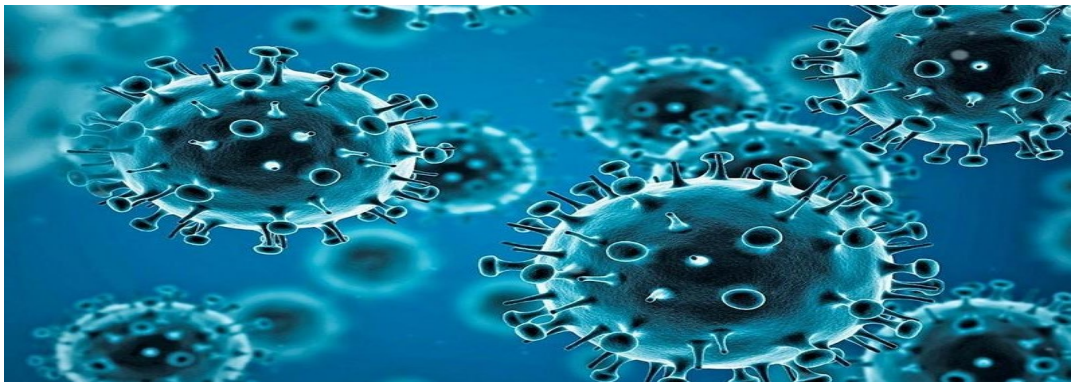


Figure 1.1 : Coronavirus



Source : https://www.who.int/health-topics/coronavirus#tab=tab_1

Data

The dataset that will be used for this project will be the Chest X-Ray Images (Pneumonia) from Kaggle. composed of a subset of 30,000 exams from the original 112,000 dataset (train + test) from the NIH CXR14 dataset using their original labels which were derived from radiology reports and, therefore with the understanding that they were not always accurate. The 30,000 selected exams were comprised of 15,000 exams with pneumonia-like labels ('Pneumonia', 'Infiltration', and 'Consolidation'), a random selection of 7,500 exams with a 'No Findings' label, and another random selection of 7,500 exams without the pneumonia-like labels and without the 'No Findings' label. Random unique identifiers were generated for each of the 30,000 exams.

We have total 26684 unique patient IDs from the two merged files, we used the below code to merge the csv files.

```
def merge_data_frames(left_df,right_df,merge_on):
  df = pd.merge(left = left_df, right = right_df, how = 'left', on = merge_on)
  df = df.drop_duplicates()
  df.info()

merge_data_frames(class_info,label_data,'patientId')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30227 entries, 0 to 37626
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   patientId  30227 non-null  object
 1   class      30227 non-null  object
 2   x          9555 non-null   float64
 3   y          9555 non-null   float64
 4   width      9555 non-null   float64
 5   height     9555 non-null   float64
 6   Target     30227 non-null  int64
dtypes: float64(4), int64(1), object(2)
memory usage: 1.8+ MB
```

Approach to EDA & Data Pre-processing:

Our first approach was to identify the various types of patient labels from the dataset, as that was the key factor to determine the details of our problem which is Pneumonia detection.

Here the labels are classified into;

(i) No Lung Opacity (with target as 0)
(ii) Normal (with target as 0)
(iii) Lung Opacity (with target as 1)

We have also performed the data pre-processing to ensure that there are no missing values in the class info and any cells containing NaN are replaced with 0. Our observation is that such label data are indicator of that the patient did not have pneumonia

Database table Preprocessing

```
def check_for_missing_data(df):
    total = df.isnull().sum().sort_values(ascending=False)   # finding total number of null values
    percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False) #percentage of values that are null
    missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent']) # putting the above two together
    return missing_data
```

Following are the checks performed for identifying the missing values (if any)

```
check_for_missing_data(label_data)
```

	Total	Percent
Target	0	0.0
height	0	0.0
width	0	0.0
y	0	0.0
x	0	0.0
patientId	0	0.0

```
check_for_missing_data(class_info)
```

	Total	Percent
class	0	0.0
patientId	0	0.0

We have also observed that the Classes are well distributed (refer figure 2 below)

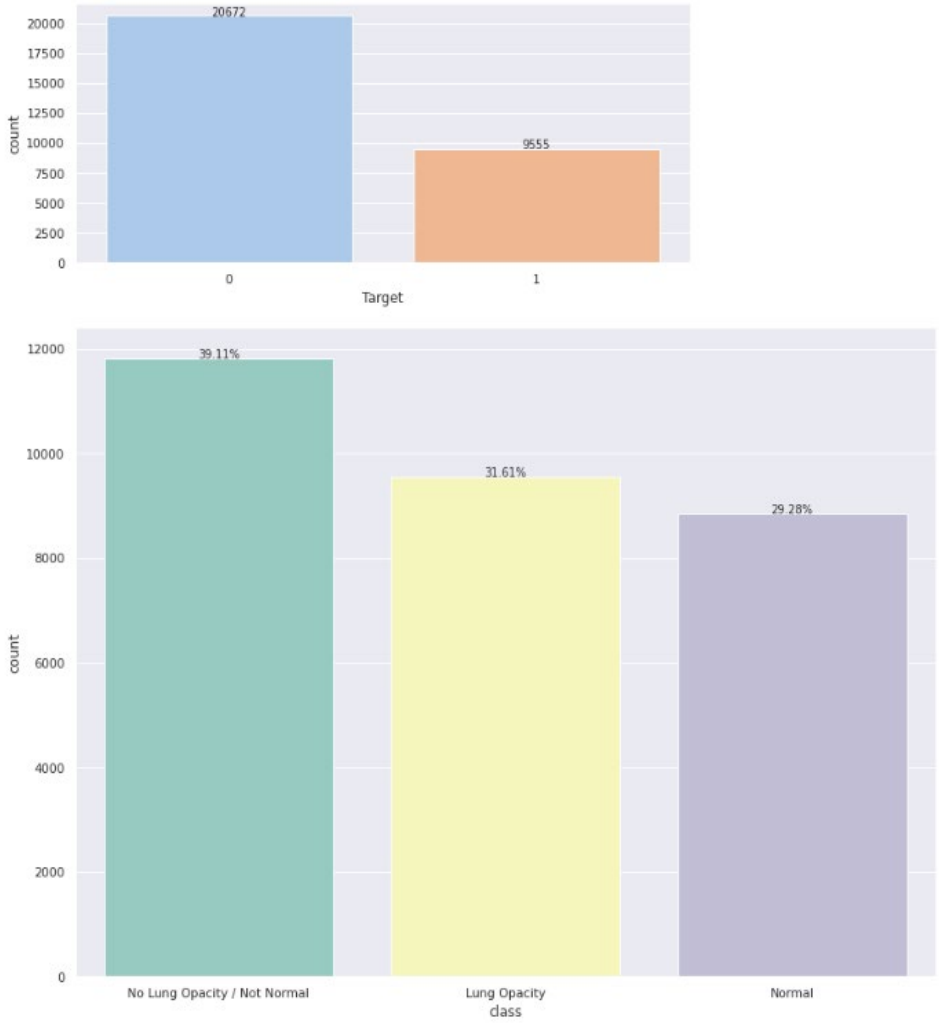
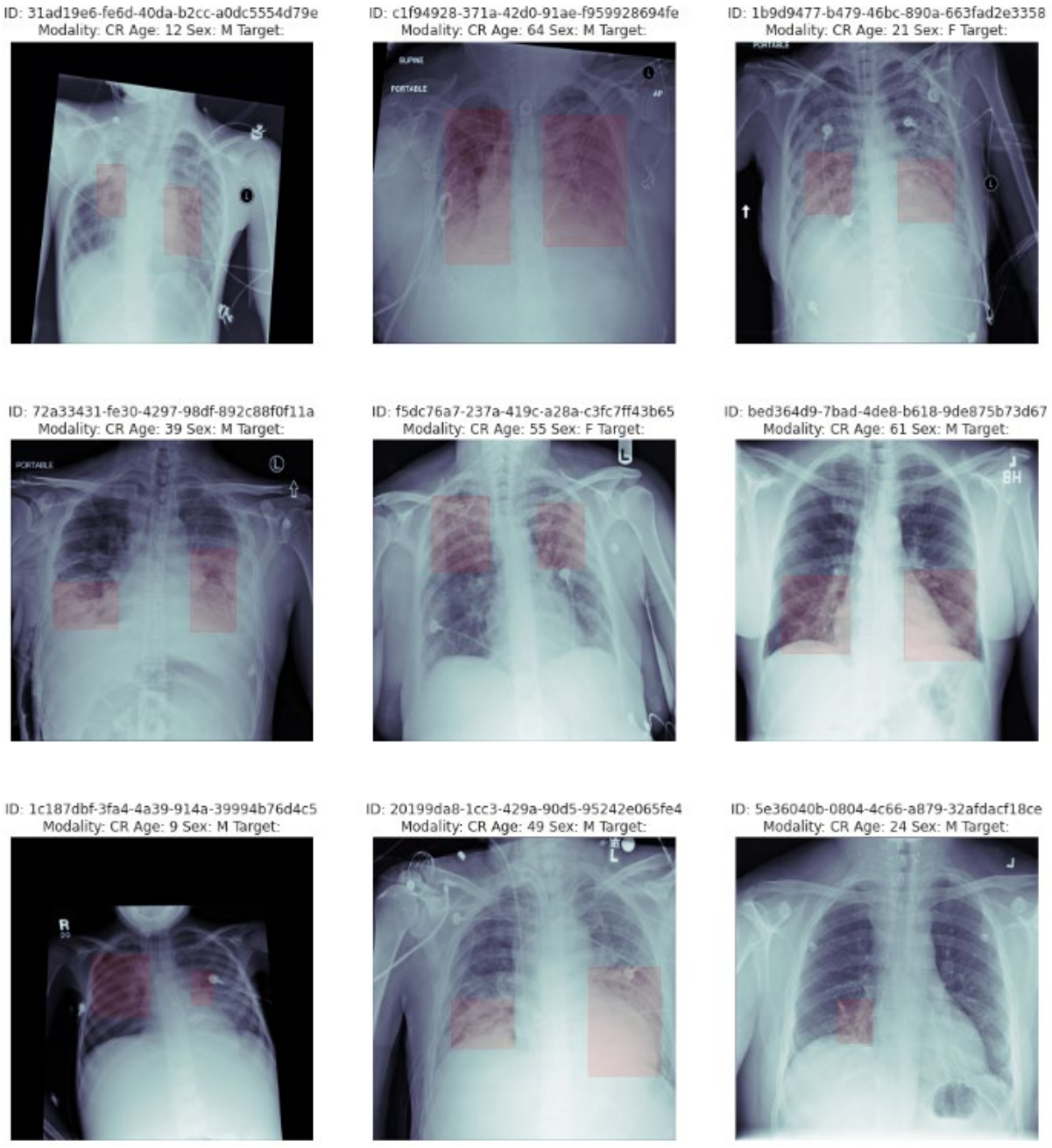


Figure 2: Classes distribution of the training dataset

EDA: Dicom Image visualisations

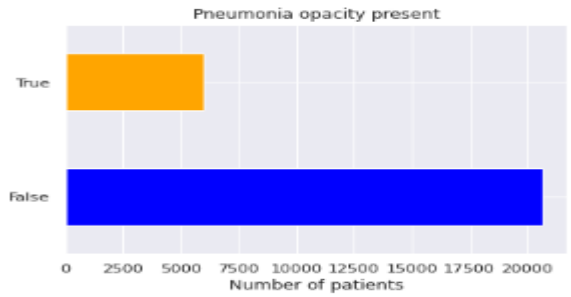
Below Dicom images are of Target =1 with bounding boxes for unhealthy patients showing the infection

Figure 3: X-ray images infected patients where opacity is found



Number of Patients where Opacity is found is between 5000 to 7500 Patients,

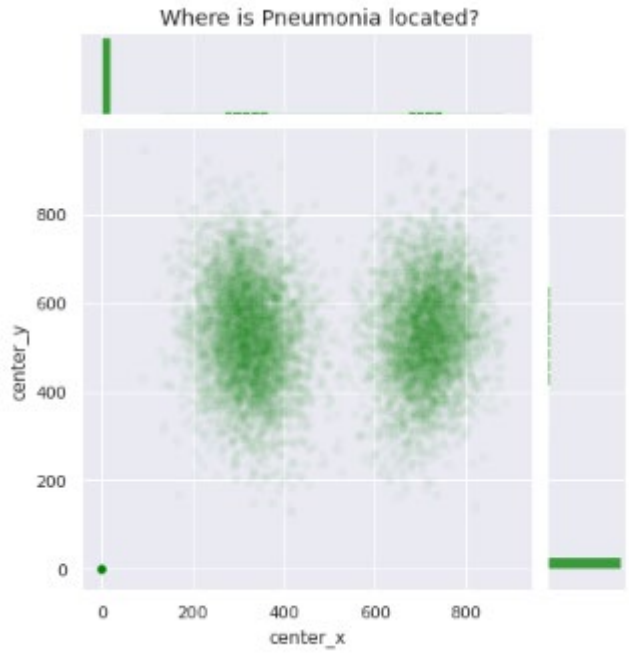
Figure 4: Opacity graph



In order to identify the location of Pneumonia, we have used a jointplot which plots the label data using x (width) and y (height).

```
centers = (label_data.dropna(subset=['x'])
           .assign(center_x=label_data.x + label_data.width / 2,
                   center_y=label_data.y + label_data.height / 2))
ax = sns.jointplot("center_x", "center_y", data=centers, height=6, alpha=0.03, color="green")
_ = ax.fig.suptitle("Where is Pneumonia located?", y=1.01)
```

Figure 5: Identify the exact location of Pneumonia



To focus on Target 0 and Target 1 we have plotted the Age Distribution by gender and target in the figure below

```
import seaborn as sns
sns.set(color_codes=True)

g = sns.FacetGrid(col='Target', hue='gender',
                  data=label_data.drop_duplicates(subset=['patientId']),
                  height=9, palette=dict(F="red", M="blue"))
_ = g.map(sns.distplot, 'age', hist_kws={'alpha': 0.5}).add_legend()
_ = g.fig.suptitle("Age distribution by gender and target?", y=1.02, fontsize=20)
```

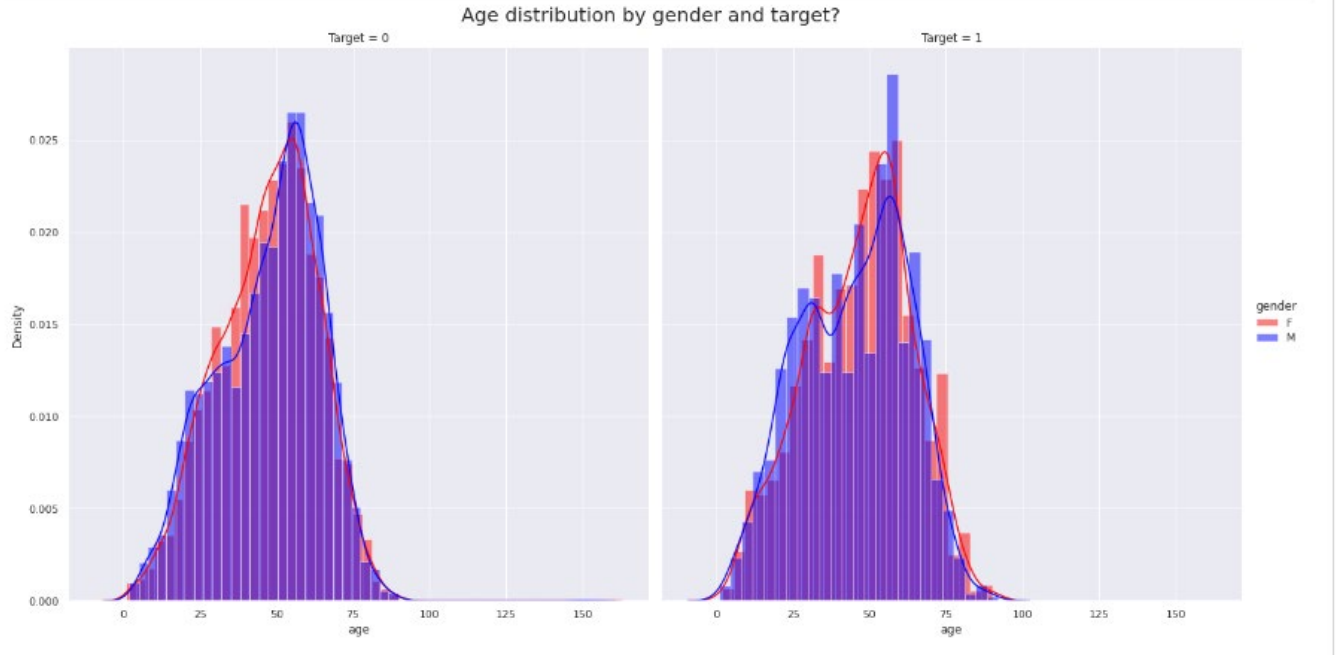


Figure 6: Patient's age distribution by gender and target

Model Building

Type of Model Used: Transfer Learning Techniques, VGG16

We have used here the architecture of the conventional pre-trained model in our model. These models consist of two parts: a convolutional base and a fully connected neural network base. The convolutional base is used to identify and extract features from our images, and then the fully connected neural network base is used to classify those features.

We can add three layers to classify this dataset.

We used the convolutional base of the VGG 16 model and then we added one fully connected hidden layer and one output layer to classify features extracted from VGG 16 convolutional base.

We used a pretrained weight of 'Imagenet'.We only want to use the convolutional part from the Imagenet model. Since convolutional bases are reusable, they are mainly used to extract features and categorize images.

We have first defined the parameters of VGG16 image transformation

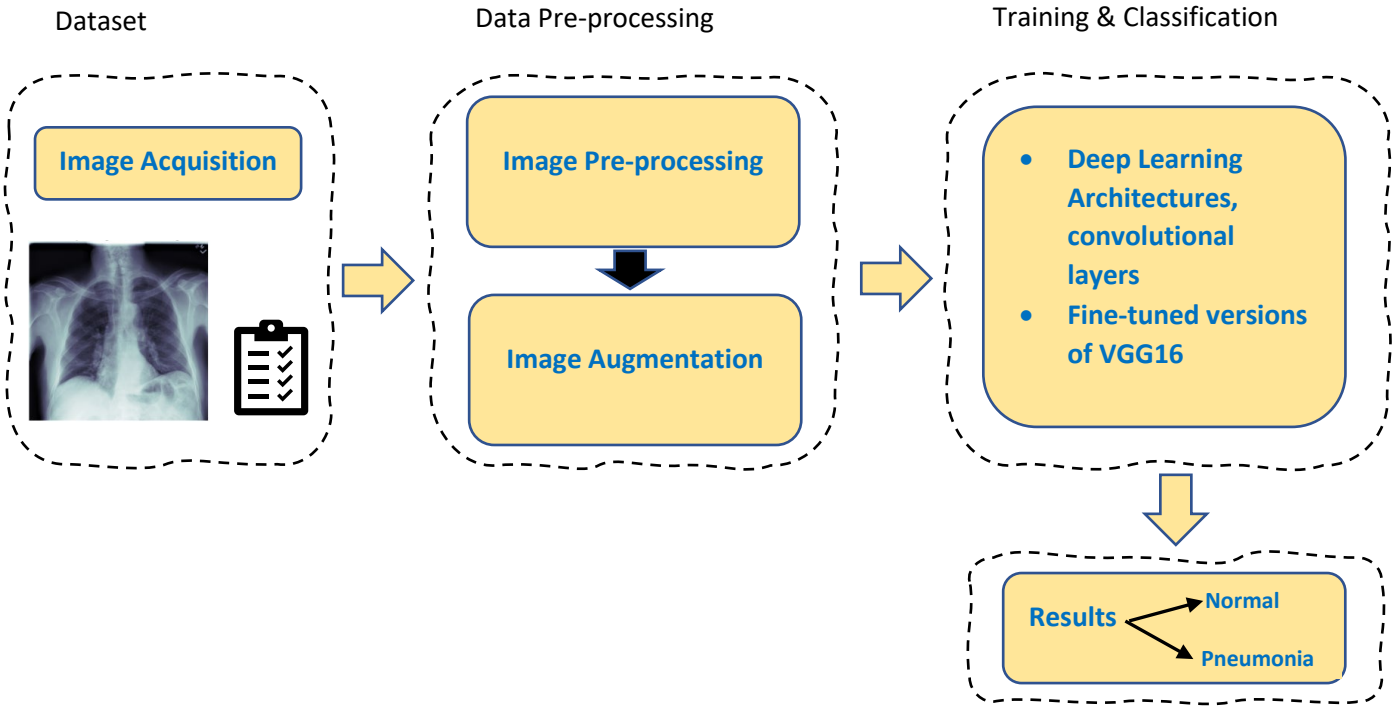
Figure 7 : VGG16 model fine tuning & using weight of the pre-trained model i.e. imagenet

```
# Fine tuning VGG16
image_input = Input(shape=(224,224,1))
image_input = Concatenate()([image_input,image_input,image_input])
vgg_model = VGG16(weights="imagenet", include_top=False,input_tensor= image_input)
vgg_model.trainable = False

basemodel_output = vgg_model.output
flatten = Flatten()(basemodel_output)

# New FC layers for Regression
bboxHead = Dense(128, activation="relu")(flatten)
bboxHead = Dense(64, activation="relu")(bboxHead)
bboxHead = Dense(32, activation="relu")(bboxHead)
bboxHead = Dense(4, activation="sigmoid", name="BBOX_Head")(bboxHead)

# New FC layers for Classification
classHead = Dense(512, activation="relu")(flatten)
classHead = Dense(512, activation="relu")(classHead)
classHead = Dense(128, activation="relu")(classHead)
classHead = Dense(1, activation="sigmoid", name="Classification_Head")(classHead)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

```
model = Model(inputs=vgg_model.input, outputs=(classHead, bboxHead))
```

Block Diagram to represent the steps to work on the challenge of Pneumonia detection



To get the model details, we have printed the model summary

Figure 8.1: Model summary

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 1  0           []
                                )]

concatenate (Concatenate)       (None, 224, 224, 3)  0           ['input_1[0][0]',
                                                                  'input_1[0][0]',
                                                                  'input_1[0][0]']

block1_conv1 (Conv2D)           (None, 224, 224, 64  1792        ['concatenate[0][0]']
                                )

block1_conv2 (Conv2D)           (None, 224, 224, 64  36928       ['block1_conv1[0][0]']
                                )

block1_pool (MaxPooling2D)      (None, 112, 112, 64  0           ['block1_conv2[0][0]']
                                )

block2_conv1 (Conv2D)           (None, 112, 112, 12  73856       ['block1_pool[0][0]']
                                8)

block2_conv2 (Conv2D)           (None, 112, 112, 12  147584      ['block2_conv1[0][0]']
                                8)

block2_pool (MaxPooling2D)      (None, 56, 56, 128)  0           ['block2_conv2[0][0]']

block3_conv1 (Conv2D)           (None, 56, 56, 256)  295168      ['block2_pool[0][0]']

block3_conv2 (Conv2D)           (None, 56, 56, 256)  590080      ['block3_conv1[0][0]']

block3_conv3 (Conv2D)           (None, 56, 56, 256)  590080      ['block3_conv2[0][0]']

block3_pool (MaxPooling2D)      (None, 28, 28, 256)  0           ['block3_conv3[0][0]']

block4_conv1 (Conv2D)           (None, 28, 28, 512)  1180160     ['block3_pool[0][0]']

block4_conv2 (Conv2D)           (None, 28, 28, 512)  2359808     ['block4_conv1[0][0]']

block4_conv3 (Conv2D)           (None, 28, 28, 512)  2359808     ['block4_conv2[0][0]']

block4_pool (MaxPooling2D)      (None, 14, 14, 512)  0           ['block4_conv3[0][0]']

block5_conv1 (Conv2D)           (None, 14, 14, 512)  2359808     ['block4_pool[0][0]']

block5_conv2 (Conv2D)           (None, 14, 14, 512)  2359808     ['block5_conv1[0][0]']

block5_conv3 (Conv2D)           (None, 14, 14, 512)  2359808     ['block5_conv2[0][0]']

block5_pool (MaxPooling2D)      (None, 7, 7, 512)    0           ['block5_conv3[0][0]']

flatten (Flatten)               (None, 25088)        0           ['block5_pool[0][0]']

dense_3 (Dense)                 (None, 512)          12845568    ['flatten[0][0]']

dense (Dense)                   (None, 128)          3211392     ['flatten[0][0]']

dense_4 (Dense)                 (None, 512)          262656      ['dense_3[0][0]']

dense_1 (Dense)                 (None, 64)           8256        ['dense[0][0]']

dense_5 (Dense)                 (None, 128)          65664       ['dense_4[0][0]']

dense_2 (Dense)                 (None, 32)           2080        ['dense_1[0][0]']

Classification_Head (Dense)     (None, 1)            129         ['dense_5[0][0]']

BBOX_Head (Dense)               (None, 4)            132         ['dense_2[0][0]']

==================================================================================================
Total params: 31,110,565
Trainable params: 16,395,877
Non-trainable params: 14,714,688
__________________________________________________________________________________________________
```

Architecture of our model is plotted as below,

```
from keras.utils.vis_utils import plot_model
import tensorflow as tf
tf.keras.utils.plot_model(
model, to_file='model.png', show_shapes=False, show_dtype=False,
show_layer_names=True, rankdir='TB', expand_nested=False, dpi=96
)
```

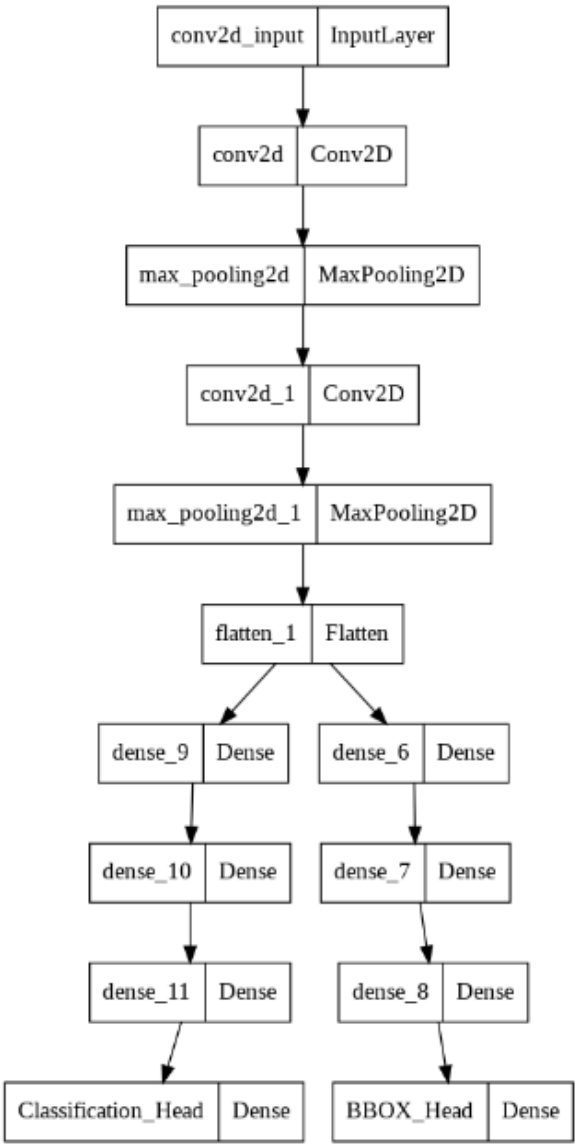


Figure 8.2: Architecture of the model

Next, we have applied the fitting of training and validation dataset

Figure 9: Training and validation of the dataset

```
print("[INFO] training model...")

History = model.fit(
    trainImages, [trainLabels, trainBBoxes],
        validation_data=(testImages, [testLabels,testBBoxes]),
        batch_size=32,
        epochs=20,
        verbose=1)
```

```
[INFO] training model...
Epoch 1/20
219/219 [==============================] - 121s 397ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0306 - Classification_Head_ac
curacy: 0.6691 - BBOX_Head_accuracy: 0.5087 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0278 - val_Classification_Head_a
ccuracy: 0.6807 - val_BBOX_Head_accuracy: 0.4027
Epoch 2/20
219/219 [==============================] - 77s 353ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0277 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.5134 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0267 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5677
Epoch 3/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0267 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.5134 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0260 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5253
Epoch 4/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0261 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.5167 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0253 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.6327
Epoch 5/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0255 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.5089 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0249 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.6067
Epoch 6/20
219/219 [==============================] - 78s 355ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0251 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4873 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0248 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5473
Epoch 7/20
219/219 [==============================] - 78s 355ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0247 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4766 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0242 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.3287
Epoch 8/20
219/219 [==============================] - 78s 355ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0245 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4644 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0239 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5767
Epoch 9/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0242 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4491 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0238 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.6297
Epoch 10/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0240 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4537 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0235 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5037
Epoch 11/20
219/219 [==============================] - 77s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0237 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4430 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0234 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.4913
Epoch 12/20
219/219 [==============================] - 78s 355ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0237 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4417 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0235 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.5757
Epoch 13/20
219/219 [==============================] - 96s 438ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0234 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4316 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0237 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.3360
Epoch 14/20
219/219 [==============================] - 78s 354ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0233 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4300 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0230 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.4930
Epoch 15/20
219/219 [==============================] - 77s 353ms/step - loss: nan - Classification_Head_loss: nan - BBOX_Head_loss: 0.0231 - Classification_Head_acc
uracy: 0.6706 - BBOX_Head_accuracy: 0.4201 - val_loss: nan - val_Classification_Head_loss: nan - val_BBOX_Head_loss: 0.0229 - val_Classification_Head_ac
curacy: 0.6807 - val_BBOX_Head_accuracy: 0.4110
Epoch 16/20
```

We have saved our model at this point.

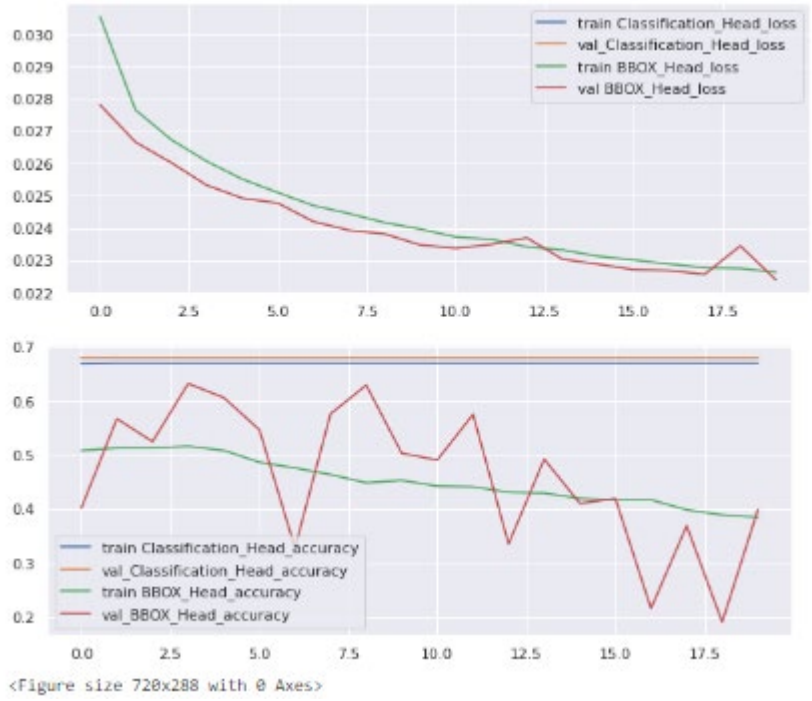
```
# Save model
print("[INFO] saving object detector model...")
model.save("/content/drive/MyDrive/VGGMOD", save_format="h5")#Use your drive path
```

```
[INFO] saving object detector model...
```

Now, inorder to run visualisation of the accuracy and the loss of training & validation of our dataset

Pre-Trained Model

Figure 10.1: Accuracy and loss of training graph

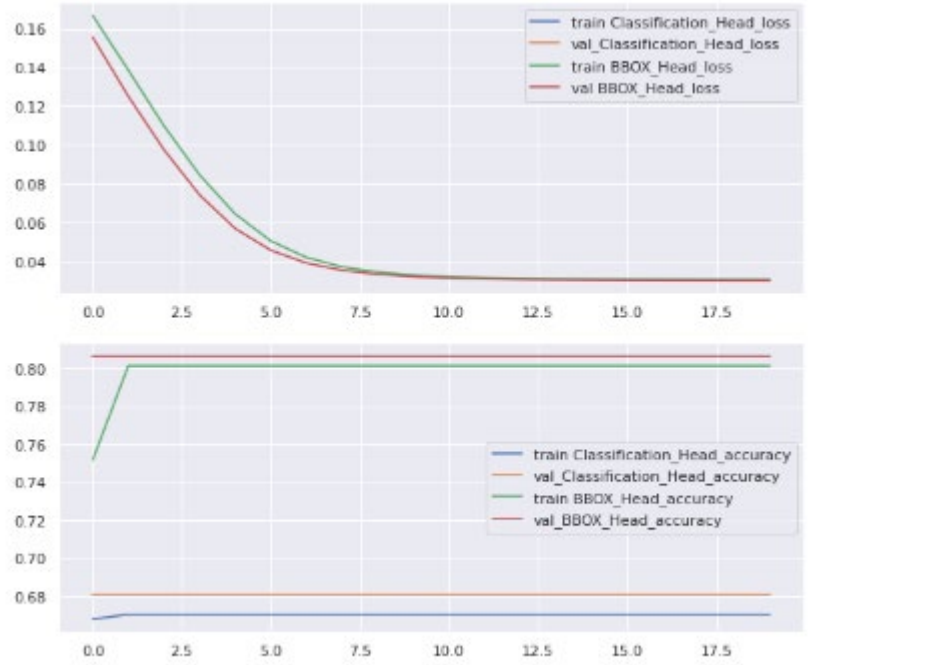


Without a Pre-Trained Model

We have used the same steps to build and compile a model which is not pre-trained. Hence, we are showing the final graph which depicts that the BBOX accuracy has improved.

Following graph has been taken from the output of our 2nd approach i.e. Without a Pre-Trained Model

Figure 10.2: Accuracy and loss of training graph



<Figure size 720x288 with 0 Axes>

Finally, we have predicted the test image as below,

```
y_pred= model.predict(testImages[testLabels])
```

```
y_pred[1]
```

```
array([[0.1285418 , 0.12050199, 0.08590765, 0.10769655],
       [0.1285418 , 0.12050199, 0.08590765, 0.10769655],
       [0.1285418 , 0.12050199, 0.08590765, 0.10769655],
       ...,
       [0.1285418 , 0.12050199, 0.08590765, 0.10769655],
       [0.1285418 , 0.12050199, 0.08590765, 0.10769655],
       [0.1285418 , 0.12050199, 0.08590765, 0.10769655]], dtype=float32)
```

Conclusion

It is observed from the above steps, that both training and validation reach the point of the Classification accuracy of approximate 67% & 68%, and with BBOX accuracy as 38.54% & 39.90% respectively at epoch=20. Whereas, when we tried the other method of without a Pre-trained model, we get the results with same Classification accuracy but the BBOX accuracy improving to 80.14% and 80.67%.

We would thus recommend the without Pre-trained model for Pneumonia detection.

References

1) https://depositphotos.com/255538672/stock-illustration-pneumonia-difference-and-comparison-of.html
2) https://www.who.int/health-topics/coronavirus#tab=tab_1