Final Report Group - 6 B

Pneumonia Detection

Table of Contents

1. Summary of problem statement, data and findings. Every good abstract describes briefly what was intended at the outset, and summarizes findings and implications.

Objective:

This Problem objective is to explore the dataset for RSNA Pneumonia Detection Challenge and to detect Inflammation of the lungs highlighted with Bounding Boxed. We are building an algorithm to detect a visual signal for pneumonia in medical images and developing a solution to automatically locate lung opacities on chest radiographs. We are trying to achieve the F1 score of our model to be greater than 0.387 which is the Radiologist's Avg F1 score.

Prerequisites:

This is a two-stage challenge. You will need the images for the current stage - provided as **stage_2_train_images.zip** and **stage_2_test_images.zip**. You will also need the training data - **stage_2_train_labels.csv** - and the sample submission **stage_2_sample_submission.csv**, which provides the IDs for the test set, as well as a sample of what your submission should look like. The file **stage_2_detailed_class_info.csv** contains detailed information about the positive and negative classes in the training set and may be used to build more nuanced models.

Dicom Images:

We have used **Pydicom** package for working with DICOM files. Medical images are stored in a special format called DICOM files (*.dcm). They contain a combination of header metadata as well as underlying raw image arrays for pixel data.

Figure: Screenshot of Lung X-Ray Dicom Image

Summary:

<u>Aim:</u> F1 Score to be higher than the existing models

We have analysed the data and performed the Image Classification through Transfer Learning by using (VGG16, VGG19, ChexNet, and MobileNet & ResNet).

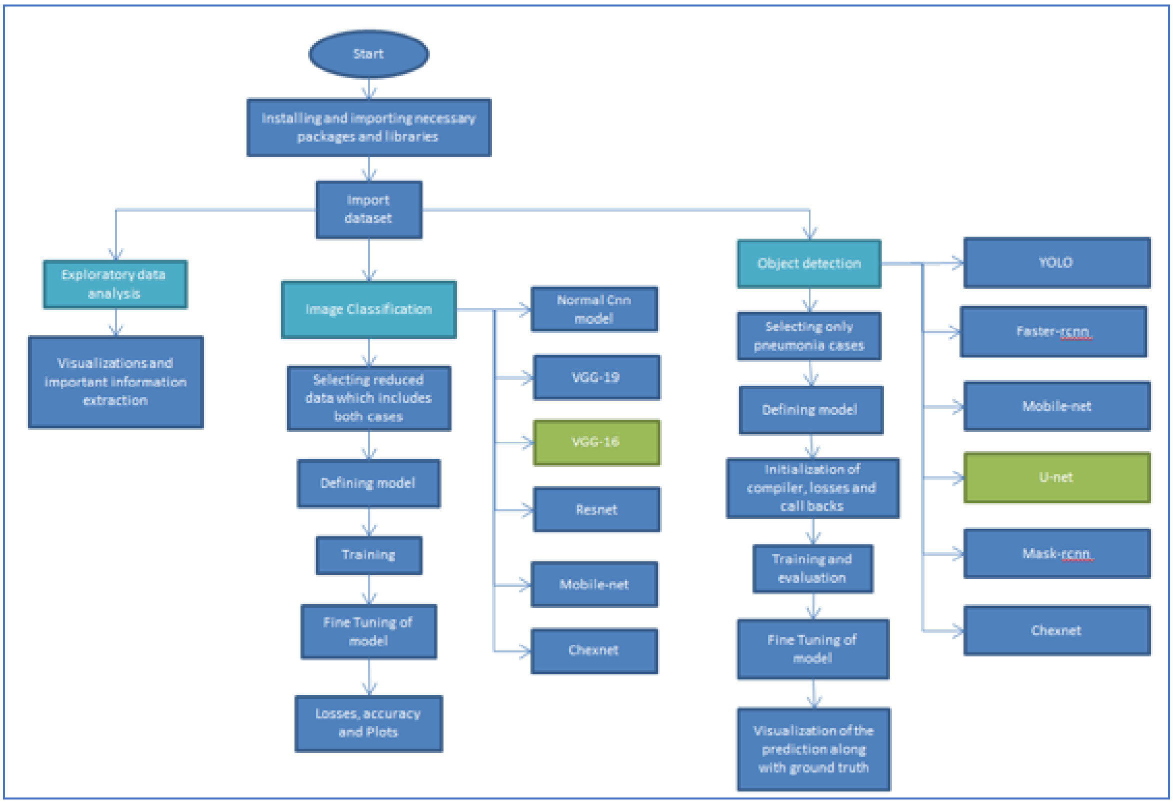


Fig: Process Flow

2. Summary of the Approach to EDA and Pre-processing

Importing the Class Info and Train Label excel sheets

The Kaggle dataset contains two excel sheets namely 'stage_2_detailed_class_info' which contains patient ids and their class definitions and 'stage_2_train_labels' which contains patient ids and their target values and bounding box dimension and coordinates. Both these files were

Understanding the files in detail

Printing head and shape of the two files and checking null values in the target variable helped us understand the files in further detail.

```
Shape of Class Info : (30227, 2)
                              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
List of null values in class labels :
Empty DataFrame
Columns: [patientId, class]
Index: []


Shape of Train Labels : (30227, 6)
                              patientId      x      y  width  height  Target
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0
1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0
2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1
List of null values in target labels :
Empty DataFrame
Columns: [patientId, x, y, width, height, Target]
Index: []
```

From the shape we could understand that there are a total of 30277 rows in both the files, meaning there was no missing datapoint.

"Class info" has 2 attributes one for patient id other for class which contained 3 values – Normal (no pneumonia), Lung Opacity (pneumonia) and No Lung Opacity/Not Normal (lung image does not have any opacity but still it does not look normal, it might have some other issues).

"Train labels" has 6 attributes namely patient id, x, y, width, height and Target. X and y are the coordinates of a corner of the bounding box and width and height are the dimensions of the same. Target says whether infected with pneumonia or not. Patients which show No Lung Opacity/Not Normal in "class info" will have target as 0. We can also see that data points with Target as 0 has no values for x, y, height and width as bounding boxes cannot be drawn where pneumonia is not present.

There were no null values in the class labels and target values.

For ease of understanding we have also added the Target variable onto the 'class info' file and check for the unique values in class and Target attributes.

```
['No Lung Opacity / Not Normal' 'Normal' 'Lung Opacity']
[0 1]
```

	patientId	class	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal	0
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity	1

Check properties of individual attributes

We checked the data types and properties of individual attributes in the files.

Class Info

```
Data types for Class Info :
patientId     object
class         object
Target        object
```

	count	unique
patientId	30227	26684
class	30227	3
Target	30227	2

Observation :

- Out of 30227 patient ids, 26684 are unique. That means some patient have multiple images of their lungs.

Train Labels

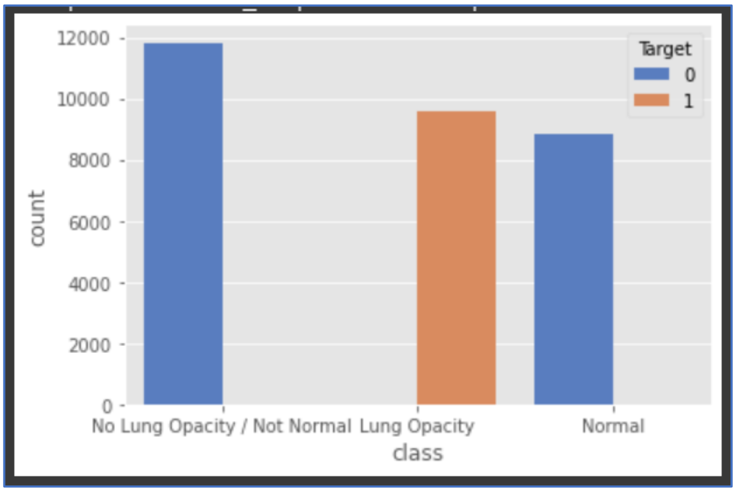
```
Data types for Train Labels :
patientId     object
x            float64
y            float64
width        float64
height       float64
Target         int64
```

	count	mean	std	min	25%	50%	75%	max
x	9555.0	394.047724	204.574172	2.0	207.0	324.0	594.0	835.0
y	9555.0	366.839560	148.940488	2.0	249.0	365.0	478.5	881.0
width	9555.0	218.471376	59.289475	40.0	177.0	217.0	259.0	528.0
height	9555.0	329.269702	157.750755	45.0	203.0	298.0	438.0	942.0
Target	30227.0	0.316108	0.464963	0.0	0.0	0.0	1.0	1.0

Observation:

- X, y, width and height only have 9555 values as these values are only populated when pneumonia is present.

Count of Class and Target attributes



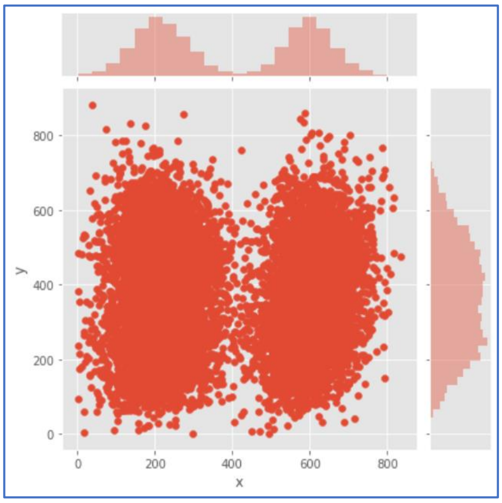
Explanation:

This plot shows the count of patients in each of the classes which are again categorized into individual targets.

Observations:

- The dataset is quite unbalanced where total number of patients without pneumonia is nearly double of those with pneumonia.
- The number of patients with No lung opacity/Not normal is more than the number of normal patients which shows that some abnormality in the lungs is a highly prevalent case in the population.

Plotting bounding box attributes

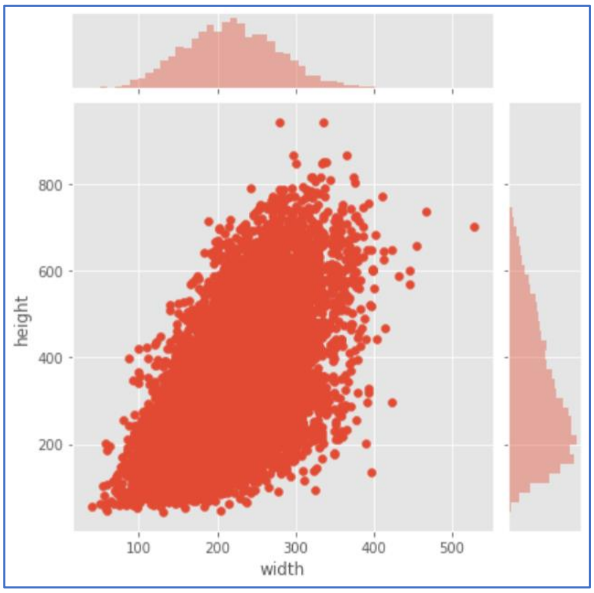


Explanation:

Plotting the x and y coordinates of the bounding box.

Observations:

- As expected the coordinates form two separate clusters denoting the left and right lungs.
- The coordinates spread across a wide area showing that pneumonia formations can occur across the area of the lungs.



Explanation:

Plotting the height and width of the bounding boxes

<u>Observation:</u>

- Density of the points is higher in the range of values which means that the pneumonia formations usually have smaller heights and widths and are usually concentrated/ localised in a region.

Loading DICOM files are a few metadata for EDA

We read a DICOM image using the PyDICOM library and checked its values and found there are a lot of metadata associated with each DICOM file which could also be utilised. Hence, we decided to include Age and Gender into our EDA as well.

Please find the list of metadata that each DICOM contains:

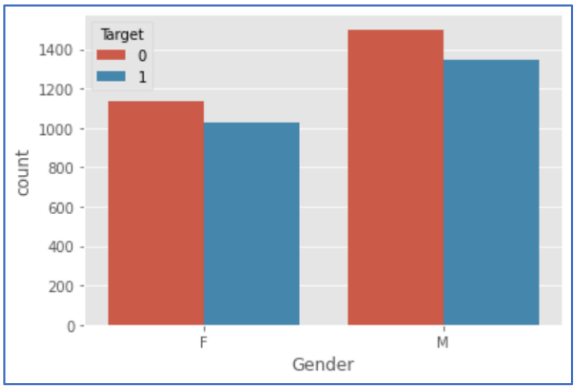
```
Total Image paths stored : 26684

Metadata of the DICOM image stored in the first path :
(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: 1.2.276.0.7230010.3.1.4.8323329.18682.1517874412.981011
(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: 'f6e78734-eed1-449e-9929-ec8ebd157f3a'
(0010, 0020) Patient ID                          LO: 'f6e78734-eed1-449e-9929-ec8ebd157f3a'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '57'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'AP'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.18682.1517874412.981010
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.18682.1517874412.981009
(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 99672 elements
```

Reading Age and Gender Data

As reading all the images was crashing the Colab server we decided to read only 5000 images. We also created bins of ages for a more effective EDA.

```
#insert train image metadata into train labels 2
#-----------------------------------------------------------------
#df_train_labels_2.shape[0]
df_train_labels_2=df_train_labels.copy()
df_train_labels_2["Age"] = np.nan
df_train_labels_2["Gender"] = np.nan
for i in np.arange(0,5000):
  if i%100==0:
    print ("Rows completed : "+str(i))
  for filename_train in lstFilesDCM_train:
    if filename_train=="./stage_2_train_images/"+df_train_labels_2.iloc[i,0]+".dcm" :
      curr_img=pydicom.read_file(lstFilesDCM_train[i])
      df_train_labels_2.iloc[i,6]=int(curr_img.PatientAge)
      df_train_labels_2.iloc[i,7]=curr_img.PatientSex
    else :
      continue

#create age bins
df_train_labels_2['Age_bin']=pd.cut(df_train_labels_2['Age'],bins=9,labels=['4-14','14-24','24-34','34-44','44-54','54-64','64-74','74-84','84-94'])
df_train_labels_2.head()
```

Count of Pneumonia cases W.r.t. Gender and Age bucket

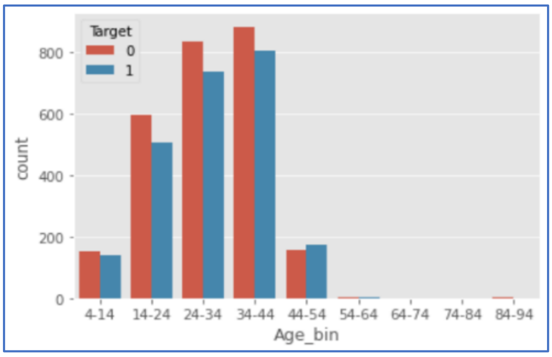


Explanation:

From above plot we can see that the count for pneumonia positive and negative cases as per gender.

Observations:

- We can clearly see males are more prone to lung related diseases and hence more males are have undergone X ray examinations.
- We can also see the proportion of pneumonia positive to pneumonia negative patients is more in case of males.

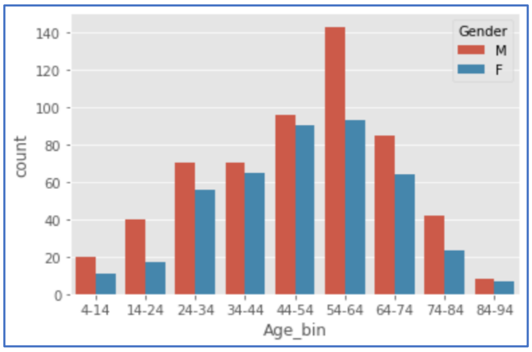


Explanation:

Distribution of X ray results as per age buckets

Observation:

- Most individuals undergoing X ray examinations and also testing positive lie in the age range of 30-44.
- Percentage of positive cases is more in case of children.



Explanation:

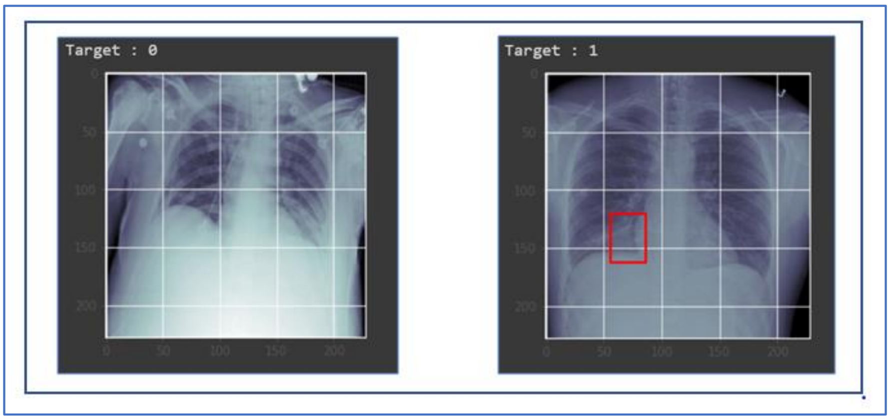
Plotting gender vs age group of patients undergoing X ray examination due to breathing issues.

Observation:

- Significantly more number of male kids face breathing issues compared to female kids.
- In the middle ages, the number of male and female patients is quite similar.
- There is a sudden increase in the number of patients between ages 54 and 64.

Print an x ray image each for Target values 0 and 1

We can observe below both the targets 0 and 1 and also a bounding box for class 1 type.



Note: We have a separate Ipython Notebook for EDA, kindly refer(Final_Report_Pnemonia_EDA.ipynb)

For Model Summary improvement: We have fine-tuned the parameters and we have tried new algorithms such as Faster RCNN, Mask RCNN and YOLO.

3. Deciding Models and Model Building Based on the nature of the problem, decide what algorithms will be suitable and why? Experiment with different algorithms and get the performance of each algorithm.

As part of model building we basically have to build models in which we would be doing Classification, and Localization.

In the Interim report, we are only dealing with classification and as part of our final report we would be adding up the object detection part.

We initially started with building a basic normal sequential model that only have a convolution layer, dense layer and then a flattened layer. This was done in-order to have a brief idea about the base model and then we have developed pertained models for classification, such as:

- VGG16,
- VGG19,
- ResNet,
- CheXnet and
- Mobile net.

We freeze the lower layer and are training the initial top layers. Then, we are using pertained image net weights in most of the model and just tweaking it by training lower layers.

We have calculated the accuracy, precision score, recall score and f1 score. The model with best of these values was VGG16 as it had everything high.

We have decided to finally use VGG16 because it has the best accuracy, high recall and high precision score and also high F1 score when compared to rest other models and also VGG16 does not seem to be over fit model, as we can see in below table.

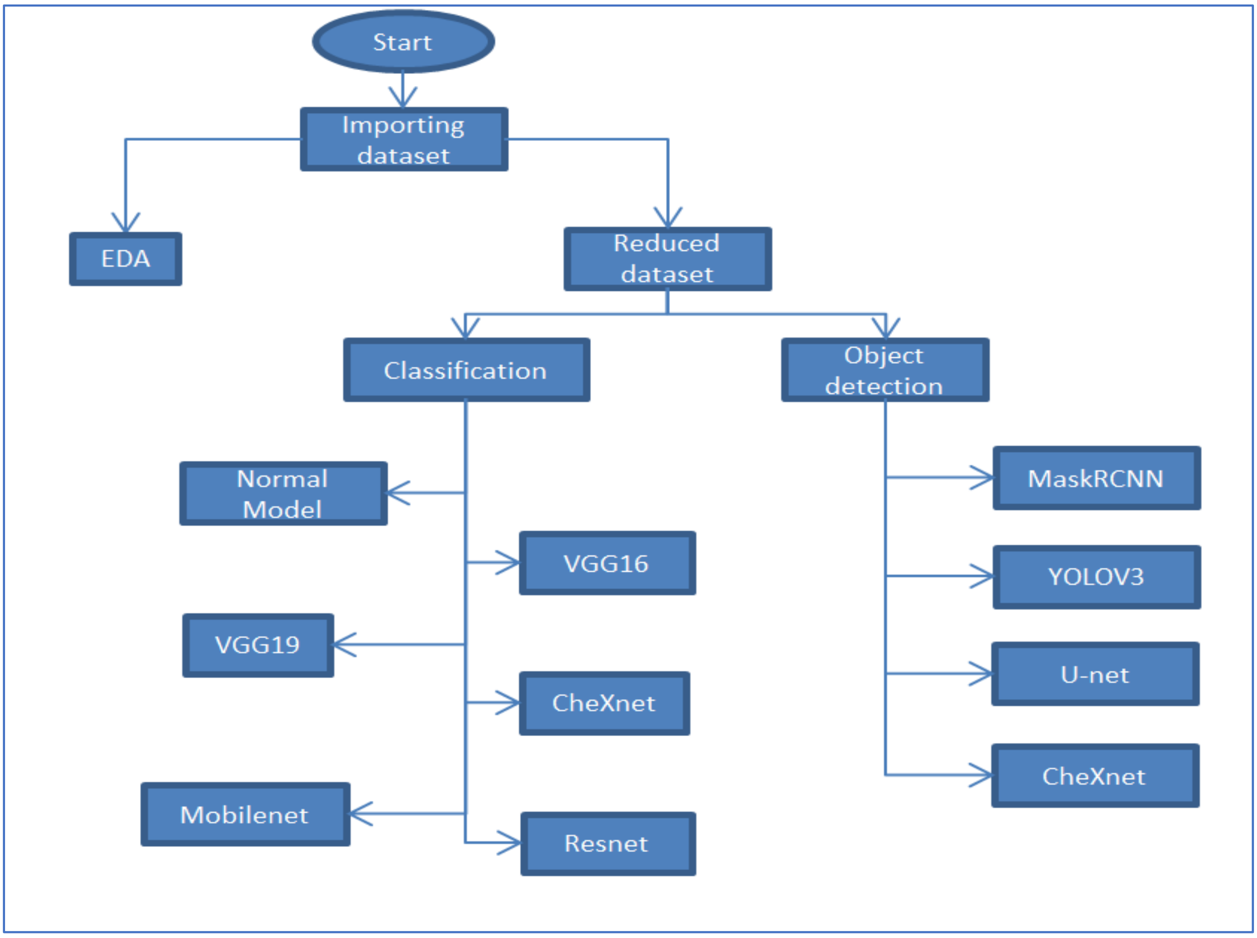
As we can see that MobilNet model also has high precision, high recall, high accuracy and high f1 score but then we can observe that it is an over fit model. Due to which we preferred VGG16 for our classification model.

Also the architecture of VGG16 is light weight so by considering all the factors we decided to use VGG16, which can be seen in a pictorial comparison below.

As part of final report, for the object detection we have used:

- MaskRCNN,
- YOLOv3,
- U-Net,
- CheXnet,
- Mobile Net
- Google Tensor Flow API

Flow chart:



Let us first understand about the pertained models that we have used in our project.

Model Architecture / Understanding the Pertained Models used:

1) VGG16

VGG16 model has a 16 layer deep network. There are 13 convolutional layers, 5 Max Pooling layers and 3 dense layers which sums up to 21 layers but only 16 weight layers. It came in 2014. It did not win the image net competition but was one of the major breakthrough. It had top 5% error rate of 7%.

Reference Link to understand it even better: https://neurohive.io/en/popular-networks/vgg16/

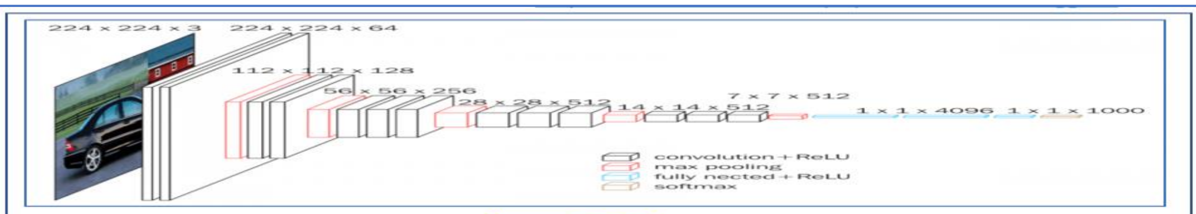


Figure - VGG16

2) VGG19

Is a variant of VGG model which consists of 19 layers i.e. (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). It came in at the end of 2014. It had top 5% error rate is less than 7%

Reference Link to understand it even better:

https://iq.opengenus.org/vgg19-architecture/

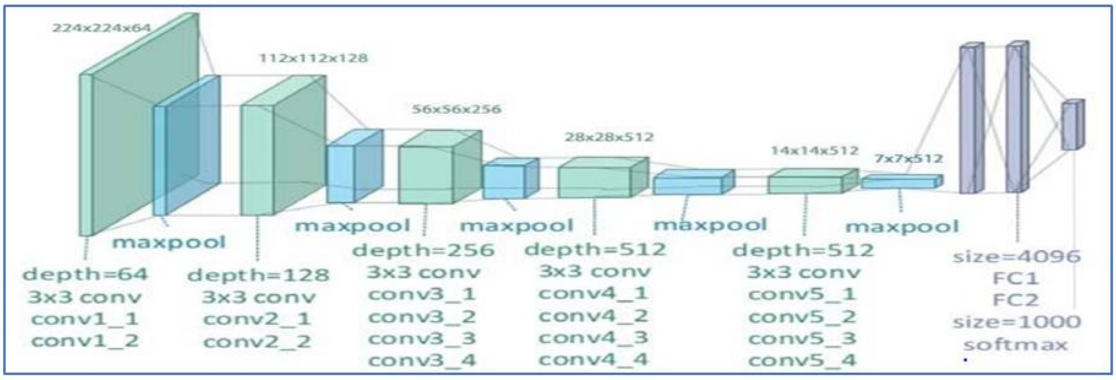


Figure - VGG19

3) ResNet

It was launched in the year2015 and was the winner of image-net competition. Its top 5% error rate was 3.57%. It was first truly deep network with 152 weight layer. It was a major breakthrough in computer vision.

Reference Link to understand it even better:

https://cv-tricks.com/keras/understand-implement-resnets/

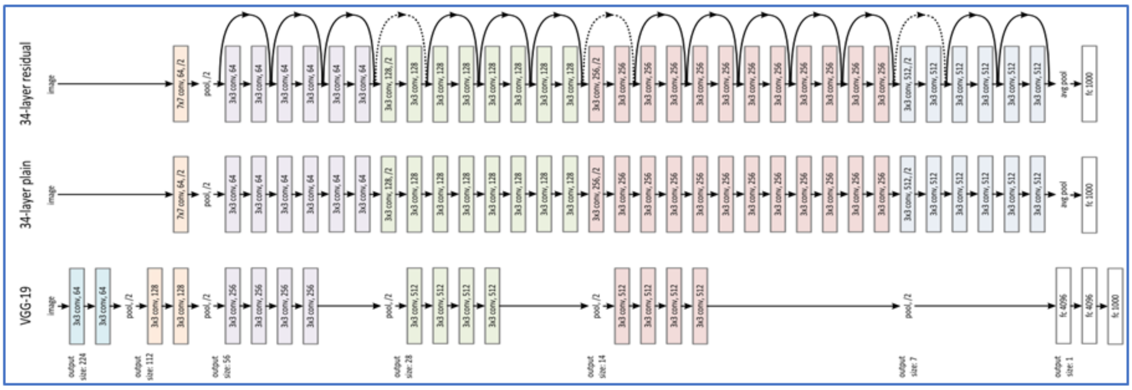


Figure – ResNet

4) MobileNet:

As it is lightweight in its architecture. It uses depthwise separable convolutions which basically means it performs a single convolution on each colour channel rather than combining all three and flattening it.

Reference Link to understand it even better:

https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

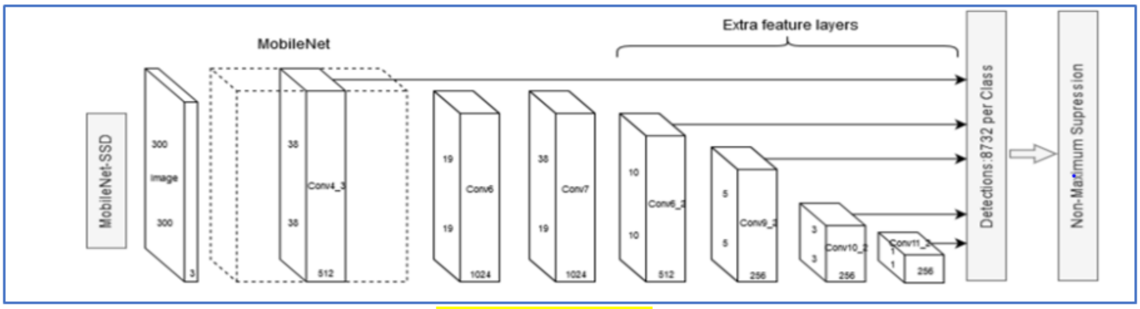


Figure – Mobilenet

5) CheXNet

CheXNet is a 121-layer convolutional neural network that inputs a chest X-ray image and outputs the probability of pneumonia along with a heat map localizing the areas of the image most indicative of pneumonia. It was mainly made for this health care industry especially for pneumonia detecting.

Reference Link to understand it even better:

https://stanfordmlgroup.github.io/projects/chexnet/

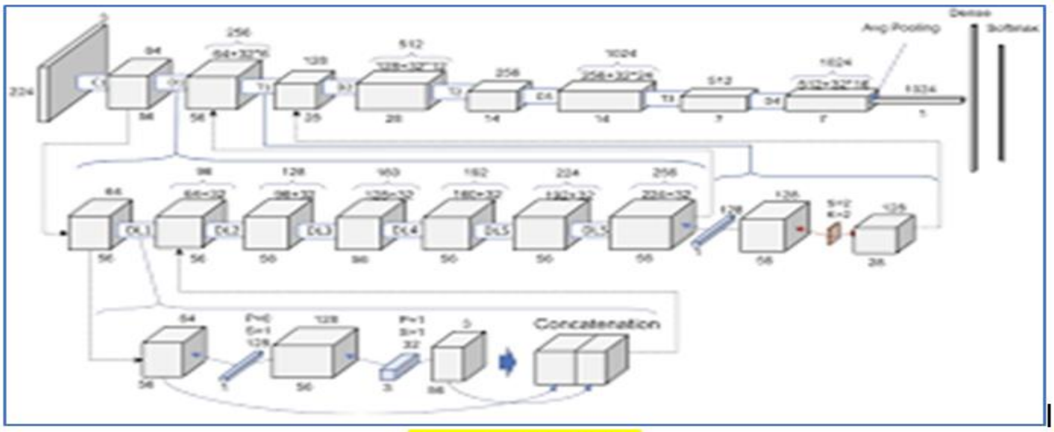


Figure - CheXNet

6) U-Net

The U-Net architecture is built upon the Fully Convolutional Network and modified in a way that it yields better segmentation in medical imaging. Compared to FCN-8, the two main differences are (1) U-net is symmetric and (2) the skip connections between the downsampling path and the upsampling path apply a concatenation operator instead of a sum. These skip connections intend to provide local information to the global information while upsampling. Because of its symmetry, the network has a large number of feature maps in the upsampling path, which allows to transfer information. By comparison, the basic FCN architecture only had number of classes feature maps in its upsampling path.

The U-Net owes its name to its symmetric shape, which is different from other FCN variants.

Reference Link to understand it even better: http://deeplearning.net/tutorial/unet.html

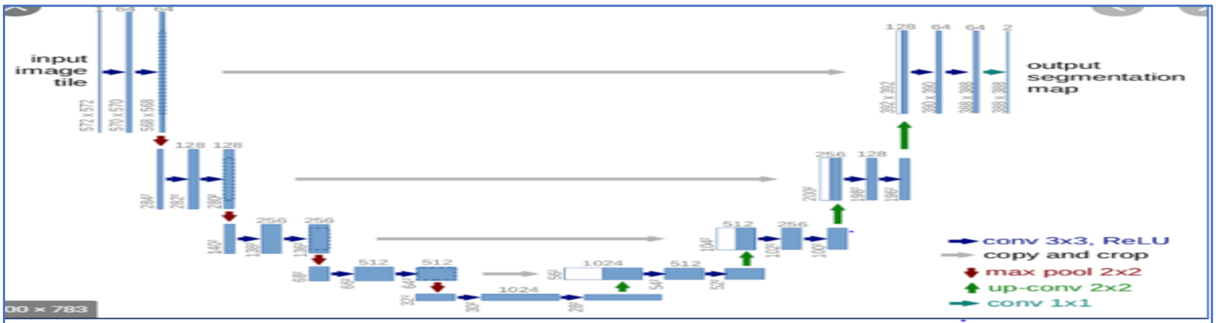
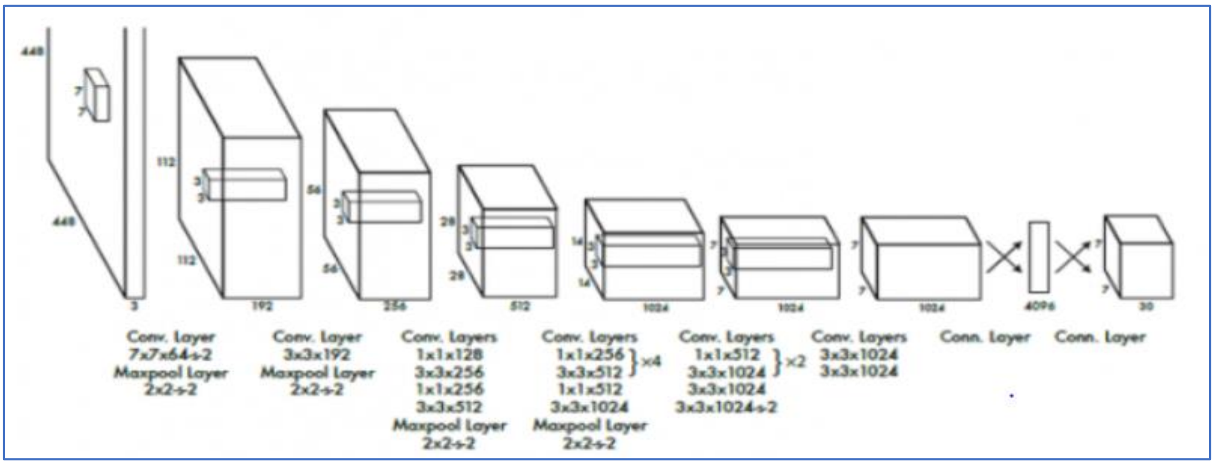


Figure: U Net

7) Yolo

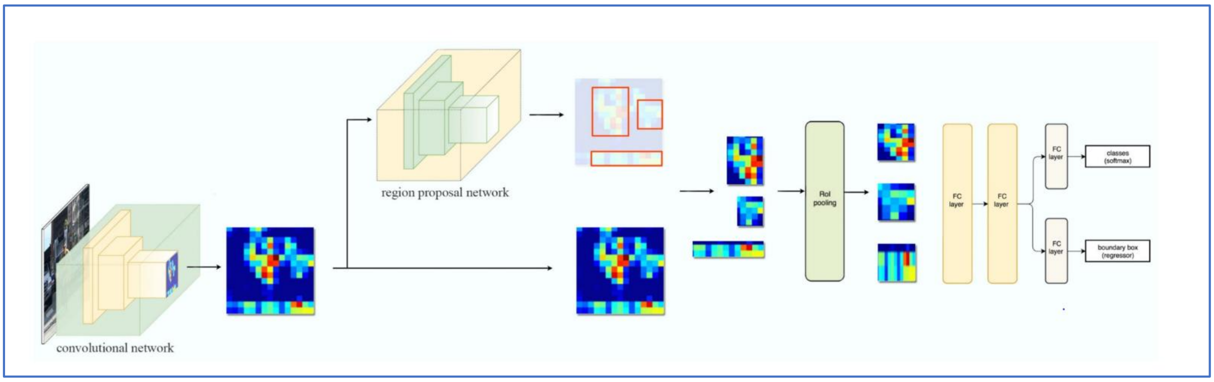
YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region.



Reference Link to understand it even better: https://pjreddie.com/darknet/yolo/

8) Mask RCNN

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.



Reference Link to understand it even better:

https://machinelearningmastery.com/how-to-perform-object-detection-in-photographs-with-mask-r-cnn-in-keras/

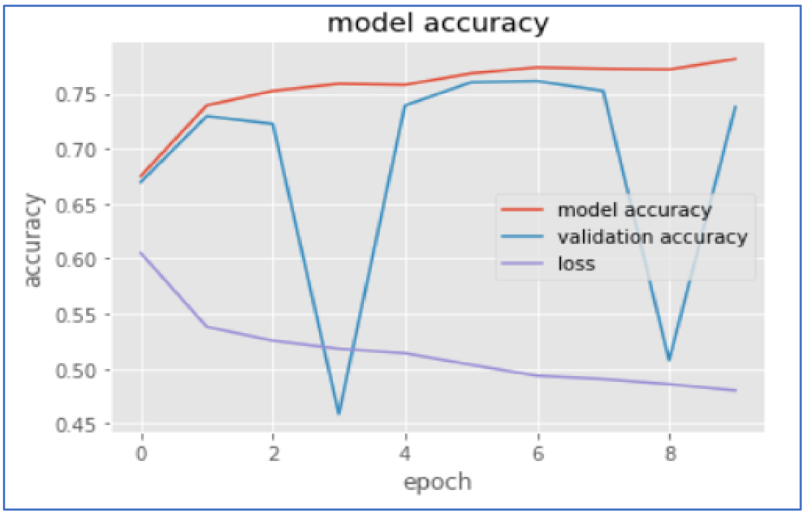
Model Summary:

Please see below screens of the model summaries, as part of classification, that we have built.

1. CNN Initial Model

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 226, 226, 128)     1280
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 224, 224, 64)      73792
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 108, 108, 32)      51232
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 52, 52, 16)        4624
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 26, 26, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 23, 23, 8)         2056
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 392)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 392)               1568
_________________________________________________________________
dense_4 (Dense)              (None, 64)                25152
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)                256
_________________________________________________________________
dense_5 (Dense)              (None, 16)                1040
_________________________________________________________________
dropout_4 (Dropout)          (None, 16)                0
_________________________________________________________________
batch_normalization_6 (Batch (None, 16)                64
_________________________________________________________________
dense_6 (Dense)              (None, 4)                 68
_________________________________________________________________
dropout_5 (Dropout)          (None, 4)                 0
_________________________________________________________________
batch_normalization_7 (Batch (None, 4)                 16
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 5
=================================================================
Total params: 161,153
Trainable params: 160,201
Non-trainable params: 952
_________________________________________________________________
```

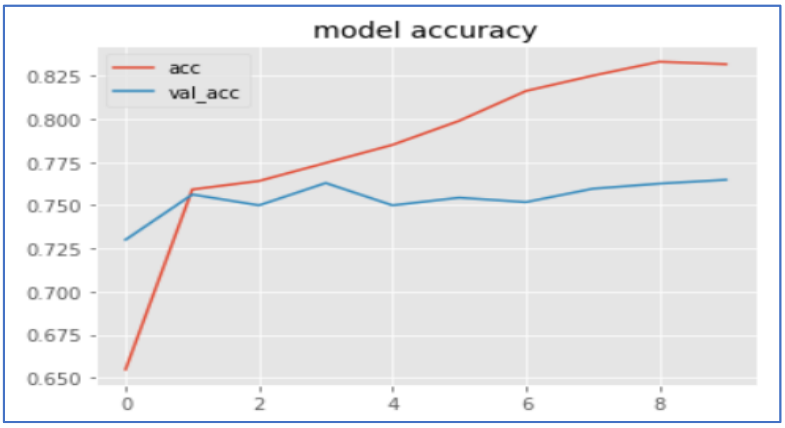
Loss and accuracy:

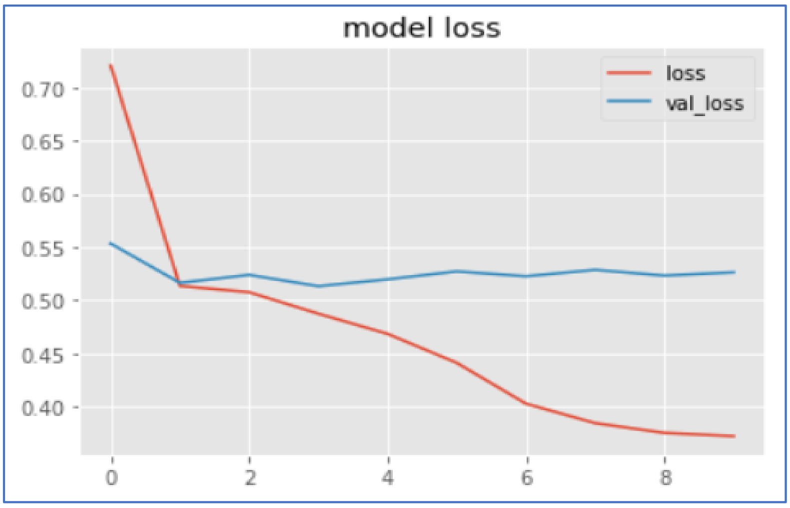


2. CNN Improved Model

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 226, 226, 64)      640
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 113, 113, 64)      0
_________________________________________________________________
dropout_12 (Dropout)         (None, 113, 113, 64)      0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 111, 111, 32)      18464
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 55, 55, 32)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 55, 55, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 96800)             0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               12390528
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 129
=================================================================
Total params: 12,409,761
Trainable params: 12,409,761
Non-trainable params: 0
_________________________________________________________________
```

Loss and accuracy:

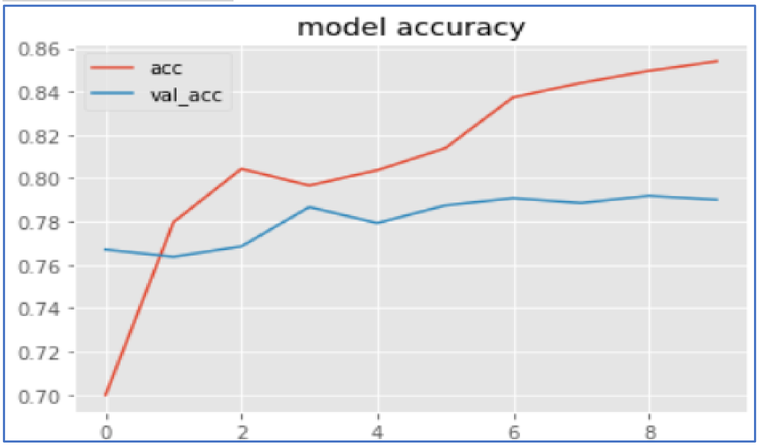




3. VGG16 Model

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 228, 228, 1)]     0
_________________________________________________________________
coords1 (Conv2D)             (None, 228, 228, 3)       30
_________________________________________________________________
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
coords2 (Conv2D)             (None, 2, 2, 3)           55299
_________________________________________________________________
dropout_14 (Dropout)         (None, 2, 2, 3)           0
_________________________________________________________________
flatten_6 (Flatten)          (None, 12)                0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               1664
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 129
=================================================================
Total params: 14,771,810
Trainable params: 57,122
Non-trainable params: 14,714,688
_________________________________________________________________
```

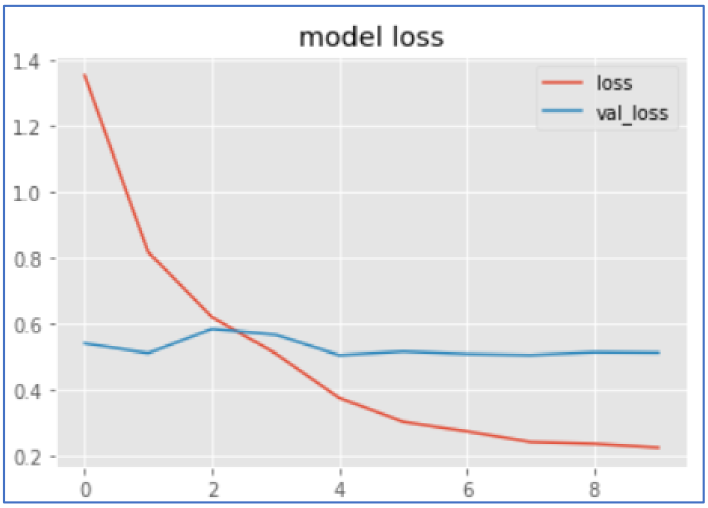
Loss and accuracy:

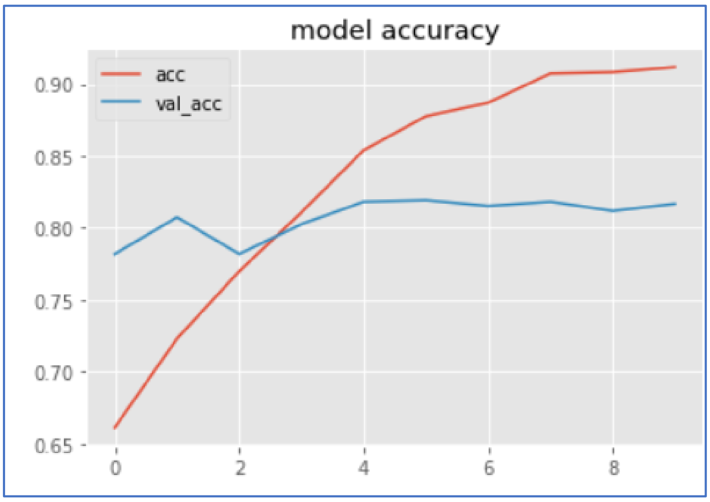


4. Mobile Net Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_22 (InputLayer)        [(None, 228, 228, 1)]     0
_________________________________________________________________
coords1 (Conv2D)             (None, 228, 228, 3)       30
_________________________________________________________________
mobilenet_1.00_224 (Model)   (None, 7, 7, 1024)        3228864
_________________________________________________________________
coords2 (Conv2D)             (None, 2, 2, 3)           110595
_________________________________________________________________
dropout_59 (Dropout)         (None, 2, 2, 3)           0
_________________________________________________________________
flatten_20 (Flatten)         (None, 12)                0
_________________________________________________________________
dense_65 (Dense)             (None, 128)               1664
_________________________________________________________________
dropout_60 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_66 (Dense)             (None, 1)                 129
=================================================================
Total params: 3,341,282
Trainable params: 112,418
Non-trainable params: 3,228,864
_________________________________________________________________
```

Loss and accuracy:

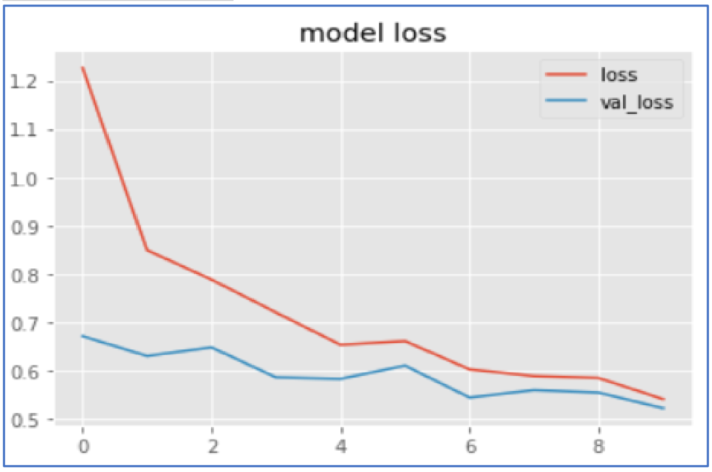


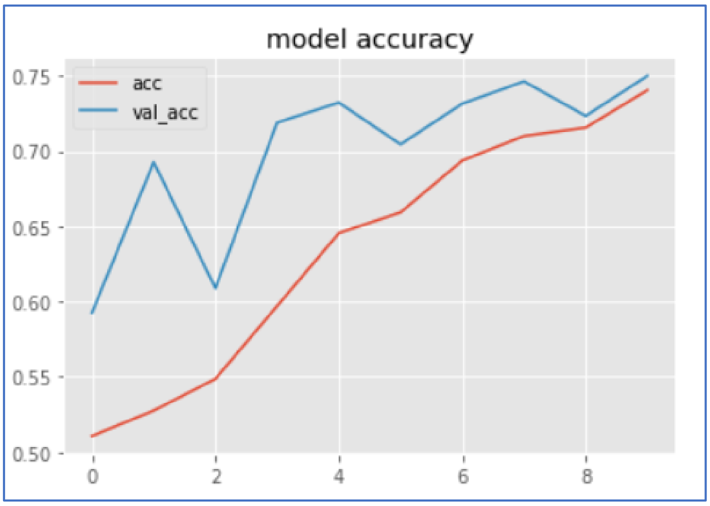


5. Resnet Model

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 228, 228, 1)]     0
_________________________________________________________________
coords1 (Conv2D)             (None, 228, 228, 3)       30
_________________________________________________________________
resnet50 (Model)             (None, 8, 8, 2048)        23587712
_________________________________________________________________
coords2 (Conv2D)             (None, 3, 3, 3)           221187
_________________________________________________________________
dropout_20 (Dropout)         (None, 3, 3, 3)           0
_________________________________________________________________
flatten_9 (Flatten)          (None, 27)                0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               3584
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 129
=================================================================
Total params: 23,812,642
Trainable params: 224,930
Non-trainable params: 23,587,712
_________________________________________________________________
```

Loss and accuracy:

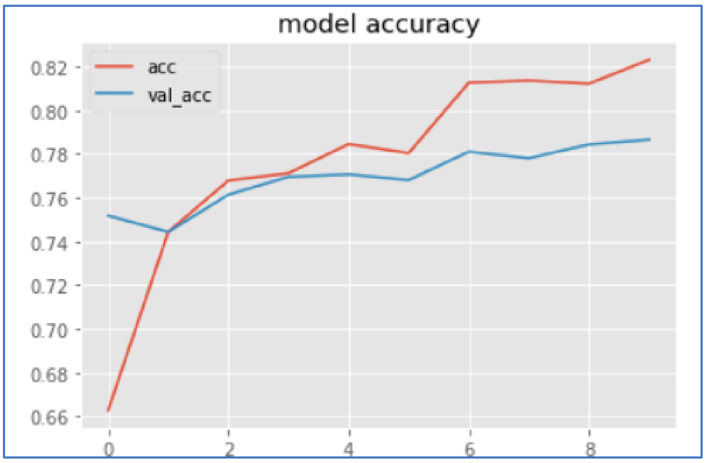




6. VGG19 Model

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 228, 228, 1)]     0
_________________________________________________________________
coords1 (Conv2D)             (None, 228, 228, 3)       30
_________________________________________________________________
vgg19 (Model)                (None, 7, 7, 512)         20024384
_________________________________________________________________
coords2 (Conv2D)             (None, 2, 2, 3)           55299
_________________________________________________________________
dropout_22 (Dropout)         (None, 2, 2, 3)           0
_________________________________________________________________
flatten_10 (Flatten)         (None, 12)                0
_________________________________________________________________
dense_24 (Dense)             (None, 128)               1664
_________________________________________________________________
dropout_23 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 129
=================================================================
Total params: 20,081,506
Trainable params: 57,122
Non-trainable params: 20,024,384
_________________________________________________________________
```

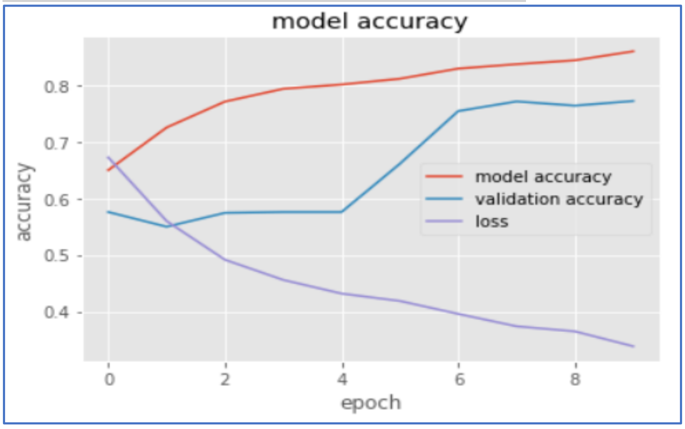
Loss and accuracy:



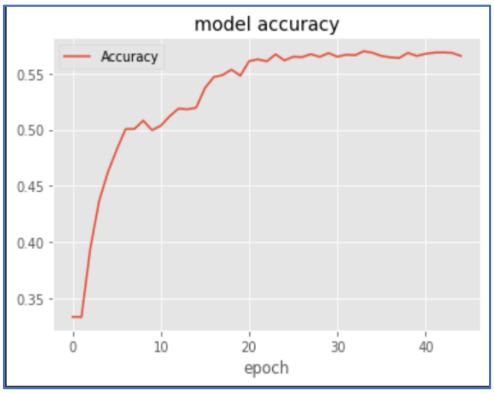
7. CheXNet Model

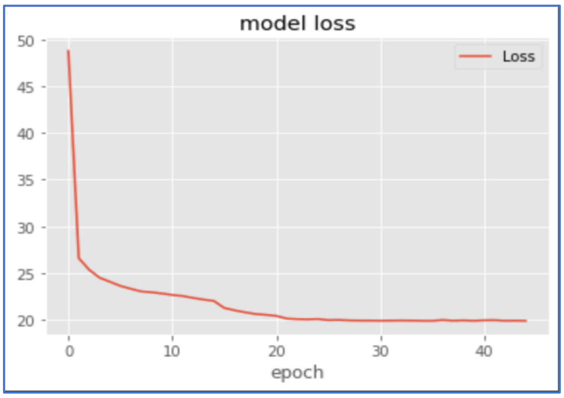
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_30 (Conv2D)           (None, 226, 226, 3)       30
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 224, 224, 3)       84
_________________________________________________________________
densenet121 (Model)          (None, 7, 7, 1024)        7037504
_________________________________________________________________
flatten_17 (Flatten)         (None, 50176)             0
_________________________________________________________________
batch_normalization_41 (Batc (None, 50176)             200704
_________________________________________________________________
dense_56 (Dense)             (None, 64)                3211328
_________________________________________________________________
dropout_48 (Dropout)         (None, 64)                0
_________________________________________________________________
batch_normalization_42 (Batc (None, 64)                256
_________________________________________________________________
dense_57 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_49 (Dropout)         (None, 32)                0
_________________________________________________________________
batch_normalization_43 (Batc (None, 32)                128
_________________________________________________________________
dense_58 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_50 (Dropout)         (None, 16)                0
_________________________________________________________________
batch_normalization_44 (Batc (None, 16)                64
_________________________________________________________________
dense_59 (Dense)             (None, 4)                 68
_________________________________________________________________
dropout_51 (Dropout)         (None, 4)                 0
_________________________________________________________________
batch_normalization_45 (Batc (None, 4)                 16
_________________________________________________________________
dense_60 (Dense)             (None, 1)                 5
=================================================================
Total params: 10,452,795
Trainable params: 10,268,563
Non-trainable params: 184,232
```

Accuracy and Loss for Image Classification:



Accuracy and Loss for Object Detection:

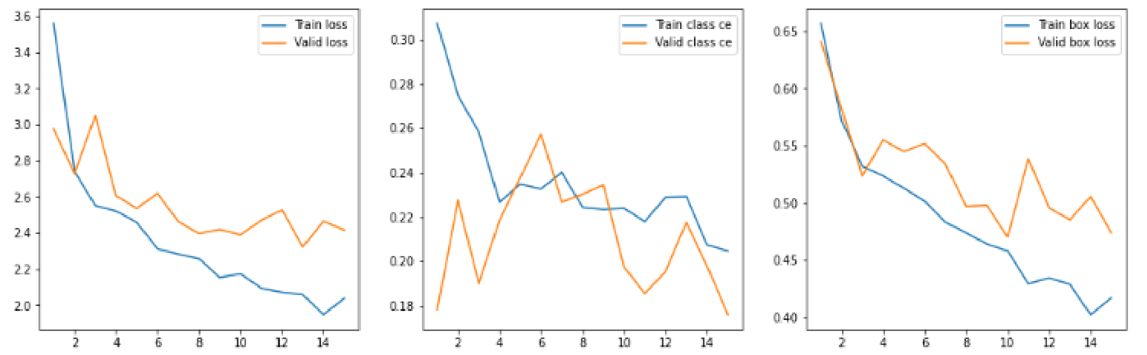




8. Mask RCNN

```
Configurations:
BACKBONE                       resnet50
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     8
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        3
DETECTION_MIN_CONFIDENCE       0.78
DETECTION_NMS_THRESHOLD        0.01
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 8
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  64
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  64
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [64 64  3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}


MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               3
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           pneumonia
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    16
STEPS_PER_EPOCH                100
TOP_DOWN_PYRAMID_SIZE          16
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           16
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Accuracy and Loss:



Epochs	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss	rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
1	2.975009	0.403969	1.188390	0.178276	0.640973	0.563382	3.558728	0.629081	1.334952	0.307143	0.656653	0.630879
2	2.726907	0.331663	1.036669	0.227559	0.582540	0.548457	2.741418	0.368791	0.945038	0.274829	0.572149	0.580593
3	3.049220	0.452373	1.274594	0.190049	0.523431	0.608755	2.551130	0.339341	0.865433	0.258185	0.531949	0.556203
4	2.605792	0.295334	1.024181	0.218651	0.554999	0.512608	2.521012	0.337245	0.884397	0.226699	0.523610	0.549042
5	2.535554	0.325270	0.937983	0.237720	0.544833	0.489729	2.458056	0.309780	0.875331	0.234808	0.512762	0.525356
6	2.618927	0.290956	1.027531	0.257374	0.551646	0.491401	2.311417	0.278667	0.783358	0.232623	0.501574	0.515177
7	2.462571	0.286155	0.919370	0.226765	0.533981	0.496282	2.282546	0.287721	0.767512	0.240179	0.483156	0.503959
8	2.396509	0.292845	0.912154	0.230249	0.497032	0.464211	2.257514	0.288146	0.764128	0.224388	0.473845	0.506988
9	2.418903	0.280946	0.935773	0.234396	0.497873	0.469896	2.152728	0.269219	0.709288	0.223433	0.464076	0.486693
10	2.389229	0.305353	0.957173	0.197564	0.470257	0.458862	2.174160	0.271248	0.731531	0.223974	0.458080	0.489309
11	2.469944	0.293554	0.973597	0.185513	0.538429	0.478831	2.093622	0.244915	0.726468	0.217926	0.429493	0.474802
12	2.526627	0.293884	1.066018	0.195531	0.495749	0.475426	2.071270	0.249347	0.689211	0.228982	0.434421	0.469289
13	2.322730	0.276068	0.903966	0.217495	0.485098	0.440085	2.057689	0.255275	0.674339	0.229200	0.428942	0.469914
14	2.465410	0.292208	1.014796	0.197940	0.505451	0.454996	1.947395	0.242487	0.643762	0.207510	0.402351	0.451266
15	2.415111	0.280030	1.041665	0.176046	0.474076	0.443276	2.038694	0.243738	0.710427	0.204686	0.416955	0.462869

Model Comparison for Image Classification:

A	B	C	D	E	F	G	H	I	J	K	L	M
Model Name	**Epochs**	**Batch Size**	**Learning Rate**	**Optimizer**	**F1 score**	**Accuracy**	**Precision Score**	**Recall Score**	**Train Loss**	**Train Accuracy**	**Val Loss**	**Val Accuracy**
Sequential	10	70	0.01	Adam	0.76	0.77	0.77	0.76	0.37	0.83	0.52	0.76
VGG16	**10**	**70**	**0.01**	**Adam**	**0.8**	**0.8**	**0.8**	**0.8**	**0.34**	**0.85**	**0.43**	**0.8**
VGG19	10	70	0.01	Adam	0.78	0.78	0.78	0.78	0.4	0.82	0.45	0.78
MobilNet	10	70	0.01	Adam	0.8	0.8	0.8	0.79	0.22	0.91	0.51	0.81
ResNet	10	70	0.01	Adam	0.73	0.74	0.75	0.73	0.54	0.74	0.52	0.75
ChexNet	10	63	0.01	Adam	0.73	0.75	0.78	0.73	0.33	0.85	0.57	0.77

Note: We have a separate I python Notebook for all the models, kindly refer (Final_Report_Pneumonia_models.ipynb)

Conclusion for Classification: Amongst all the models built, **VGG16** has performed better of all with a F1 Score of 0.80, as shown above with rest other parameters and other models.

Model Comparison for Object Detection:

U Net:

loss: 0.0617 , accuracy: 0.9218

```
5410/5410 [==============================] - 56s 10ms/step - loss: 0.0214 - accuracy: 0.9712 - val_loss: 0.0616 - val_accuracy:
0.9218
Epoch 30/30
5410/5410 [==============================] - 56s 10ms/step - loss: 0.0215 - accuracy: 0.9710 - val_loss: 0.0617 - val_accuracy:
0.9218
```

Mobile-net:

loss: 917.9167, accuracy: 0.4097, LOSS name-mean_squared_error

```
Epoch 50/50
64/64 [==============================] - ETA: 0s - loss: 190.0780 - accuracy: 0.4819
Epoch 00050: val_accuracy did not improve from 0.26025
64/64 [==============================] - 79s 1s/step - loss: 190.0780 - accuracy: 0.4819 - val_loss: 917.9167 - val_accuracy:
0.4097 - lr: 1.0000e-06
```

Yolo:

iou: 0.75, LOSS name-IOU

```
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.0000
00, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.000001, iou_loss = 0.000000, total_loss = 0.000
001
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 82 Avg (IOU: 0.726848, GIOU: 0.719680), Class: 0.994251, Obj: 0.16417
4, No Obj: 0.001061, .5R: 1.000000, .75R: 0.500000, count: 6, class_loss = 1.148280, iou_loss = 0.659660, total_loss = 1.8079
39
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 94 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.00000
0, No Obj: 0.000066, .5R: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.005619, iou_loss = 0.000000, total_loss = 0.0056
19
v3 (mse loss, Normalizer: (iou: 0.75, cls: 1.00) Region 106 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.0000
00, No Obj: 0.000001, .5R: 0.000000, .75R: 0.000000, count: 1, class_loss = 0.000001, iou_loss = 0.000000, total_loss = 0.000
001
```

Faster RCNN

Loss=1.36

```
0602 11:41:54.968904  5896 learning.py:507] global step 5902: loss = 1.0271 (1.179 sec/step)
NFO:tensorflow:global step 5903: loss = 1.4717 (1.123 sec/step)
0602 11:41:56.093916  5896 learning.py:507] global step 5903: loss = 1.4717 (1.123 sec/step)
NFO:tensorflow:global step 5904: loss = 1.3634 (1.260 sec/step)
0602 11:41:57.354523  5896 learning.py:507] global step 5904: loss = 1.3634 (1.260 sec/step)
```

ChecXNet

Loss = mean_absolute_error, Loss = 19.9030

```
Epoch 45/50
81/81 [==============================] - ETA: 0s - loss: 19.9030 - accuracy: 0.5658Restoring model weights from the end of the best epoch.
81/81 [==============================] - 36s 450ms/step - loss: 19.9030 - accuracy: 0.5658 - val_loss: 22.4031 - val_accuracy: 0.5343 - lr: 1.0000e-
Epoch 00045: early stopping
```

```
sgd_optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)

model_chex.compile(optimizer=sgd_optimizer, loss='mean_absolute_error', metrics='accuracy')
model_chex.summary()
```

Mask RCNN

```
Epoch 13/15
100/100 [==============================] - 1242s 12s/step - loss: 2.0577 - rpn_class_loss: 0.2553 -
rpn_bbox_loss: 0.6743 - mrcnn_class_loss: 0.2292 - mrcnn_bbox_loss: 0.4289 - mrcnn_mask_loss: 0.4699 -
val_loss: 2.3227 - val_rpn_class_loss: 0.2761 - val_rpn_bbox_loss: 0.9040 - val_mrcnn_class_loss: 0.2175 -
val_mrcnn_bbox_loss: 0.4851 - val_mrcnn_mask_loss: 0.4401
```

Conclusion for Object Detection: Amongst all the Object detection models that we have built, **U-Net** seems to have performed better in terms of reliability and detection, since we have a few false positives but no false negatives.

4. Model Performance:

Model Summary Notes:

We have tried to optimize the model by following few steps for each model:

- We have iterated the model to 10 epochs each, to understand the performance metrics and gauge accordingly.
- We did tweak the learning rate to understand the losses for each model. So optimized it to the best at 0.01
- We have used the Adam optimization algorithm to reduce the loss, after comparing it with SGD, RMSE prop.
- We could see that the model was getting over fitted for all the pre-trained models, in-order to get it précised, we added drop out layers to optimize the model.
- We have also added the call backs, to reduce the learning rates, model check point and early stopping for regularization of our model.
- We also have tweaked the number of layers and the unit in each layer to increase the model accuracy when our model could achieve right depth.
- We have finally decided to use VGG16, basis our understanding and behaviour because it had the best accuracy, high recall and high precision score also the architecture of VGG16 is light weight so by considering all the factors we decided to use VGG16.

5. Comparison to Benchmark:

Classification Process:

Stanford has already developed an algorithm that can detect pneumonia from chest X-rays at a level exceeding that of practicing radiologist. Their algorithm, CheXNet, is a 121-layer convolutional neural network trained on ChestX-ray14, currently the largest publicly available chest X-ray dataset, containing over 100,000 frontal-view X-ray images with 14 diseases. Four practicing academic radiologists annotate a test set, on which they have compared the performance of CheXNet to that of radiologists. They found that CheXNet exceeds average radiologist performance on the F1 metric. Later they extend CheXNet to detect all 14 diseases in ChestX-ray14 and achieve state of the art results on all 14 diseases. We find that the model exceeds the average radiologist performance on the pneumonia detection task.

We have developed various models which we have discussed above, in-order to gauge the scores basis various parameters, amongst which, the major parameter that we relied upon is the F1 Score for better classification, in-order to have a comparison benchmark to that of Stanford. The reason being that, they have been much inclined on F1 Score, we have done a similar comparison to establish a better model. Amongst all our models VGG16 has proven to be the best model, for it has a F1 score of 80%.

Detection Process:

PYolo uses double K-means to produce the anchor box of a lesion. PYolo uses DarkNet53 to extract features, uses MaskFPN to fuse features of different levels, and uses a multi-branch convolution module to obtain multi-perception field information. Unlike Yolov3, PYolo only detects the features of the module output. The input image size of DarkNet53 is 416 × 416 pixels, and the output features are {F1, F2, F3} with the sizes of {13 × 13, 26 × 26, 52 × 52}, respectively. In the experiment, the input image was scaled to 416 × 416 pixels in the pre-processing stage. The difference between MaskFPN and FPN is that MaskFPN uses the information of high-level feature as prior knowledge to generate a weight map, and then multiplies the weight map with low-level features linearly to suppress the output of inaccurate semantic information of low-level features. By contrast, FPN directly combines high-level features and low-level features, directly overcoming the problem of inaccurate semantic information in low-level features.

In an object detection algorithm, the ratios of positive and negative samples are critical to the performance of the algorithm. Yolov3, PYolo corresponds to the feature points by dividing the image into grid cells. In the training phase, the real bounding box is mapped to the corresponding coordinates on the feature map by dividing by the stride; in the detection phase, the predicted bounding box on the feature map is mapped to the corresponding coordinates on the original image by multiplying the stride. The dimensions of the output features of PYolo are [S, S, A * (B + Conf + Cls)]. S × S is the number of grid cells; B is the predicted bounding box; Conf is the confidence level of the output object; Cls is the class of the dataset; and A is the number of scales for each anchor. With respect to the selection of positive and negative samples, anchors with the Intersection over Union (IOU) with the ground-truth bounding boxes were used for evaluation. Anchors that have IOU with any ground-truth box greater than 0.5 were included as training samples. The canter points of the anchor and ground-truth bounding boxes that fall on the same grid were designated as positive samples, and other anchors were designated as negative samples.

There is still a problem of imbalance between positive and negative samples in the screened sample set. To overcome the problem of imbalance between positive and negative samples [21], a hyper-parameter $\lambda = 200$ is introduced in the loss function to strengthen the learning intensity for negative samples and accelerate the speed of the convergence of the model. The localization loss function is different from the function in Yolov3. Smooth L1 loss was adopted as the localization loss function as it has a higher level of smoothness compared to others. The loss functions of the model are as follows:

$$L_{loc} = \sum_{i \in Pos}^{N} \sum_{m \in \{x,y,w,h\}} \mathrm{smooth}_{L1}(g_m - p_m)$$

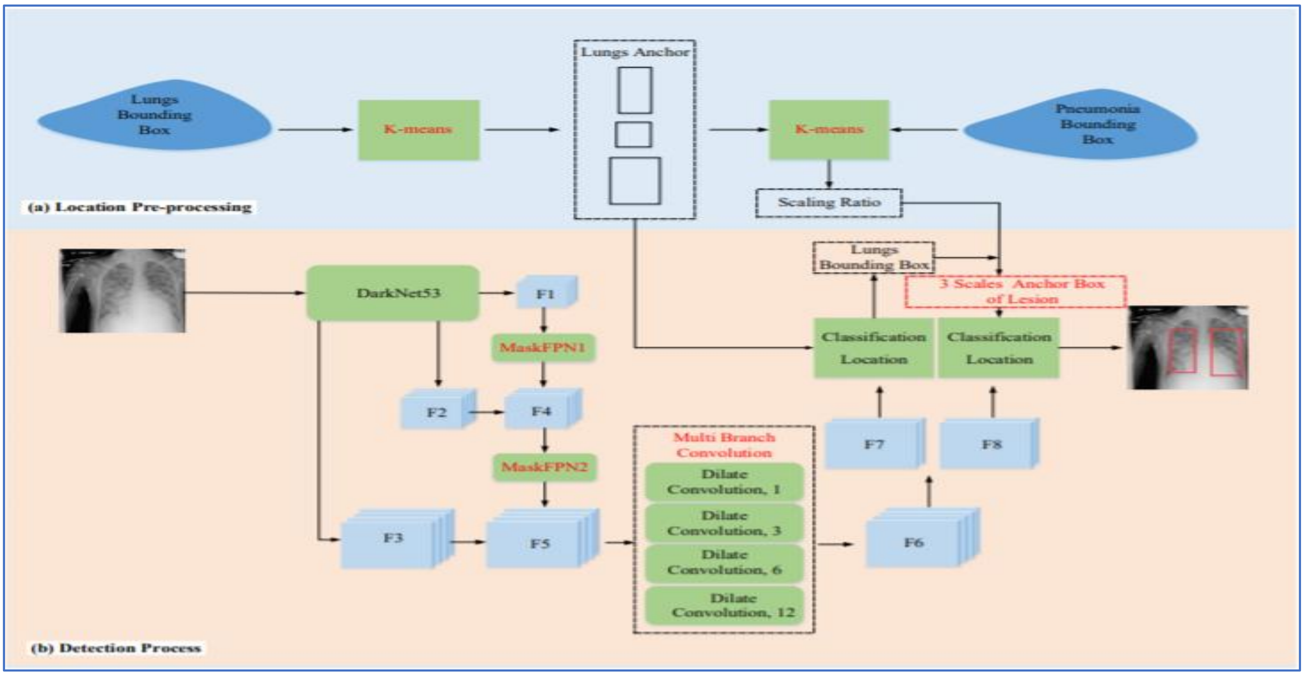
$$L_{cls} = -\sum_{i \in Pos} C_i \log(X_i)$$

$$L_{pos} = -\sum_{i \in Pos} M_i \log(Y_i)$$

$$L_{neg} = -\sum_{i \in Neg} M_i \log(Y_i)$$

$$L_{total} = L_{loc} + L_{cls} + L_{pos} + \lambda L_{neg}$$

Here LLoc, Lcls, Lpos, and Lneg represent localization loss, classification loss, positive sample loss, and negative sample loss, respectively, and g, p, C, X, M, and Y refer to the actual coordinates, predicted coordinates, probability of the actual class, and probability of the predicted class, actual set of positive.



Link for Stanford paper for Object Classification (CheXNet):
https://stanfordmlgroup.github.io/projects/chexnet/

Link for Object Detection technique:

https://www.mdpi.com/2076-3417/10/5/1818/pdf

Amongst all the Object detection models that we have built, the U-Net seems to have performed better in terms of reliability and detection, since we have a few false positives but no false negatives.

6. Visualisation:

We have done visualisations to show a comparison of the ground truth and predictions that our model has made. We would see below the pictorial images of various models build, that would give us a glimpse of the Actuals VS Predicted.

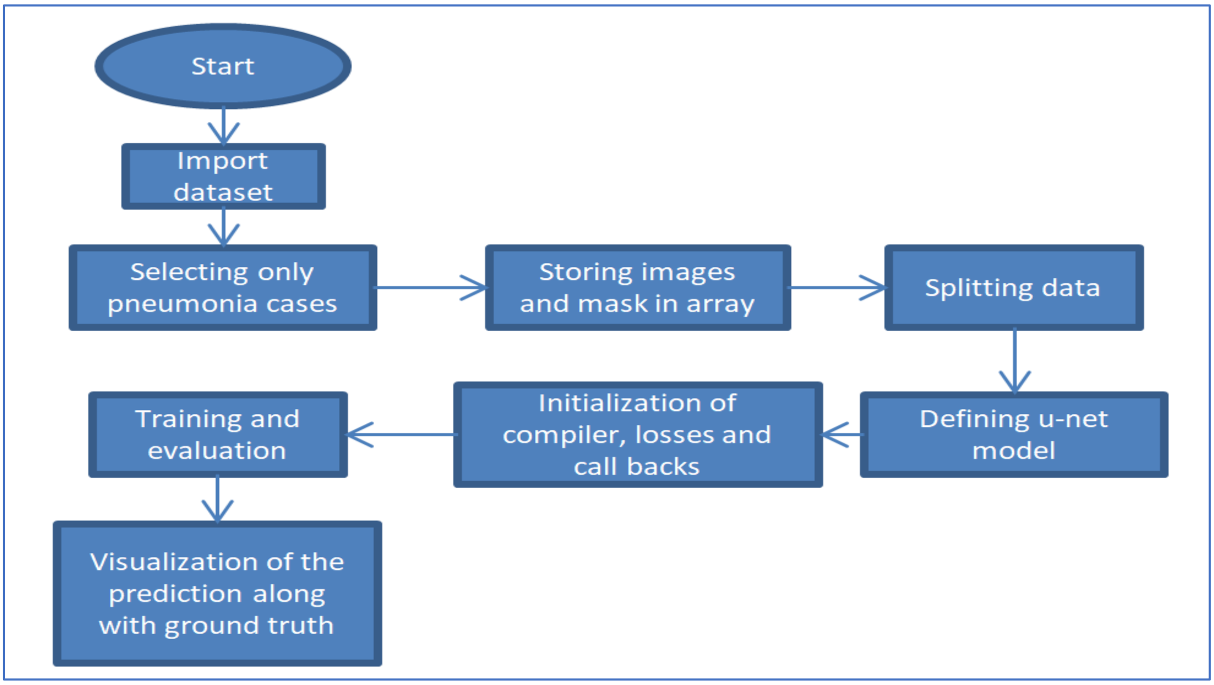
a) U-Net:

Process Flow:

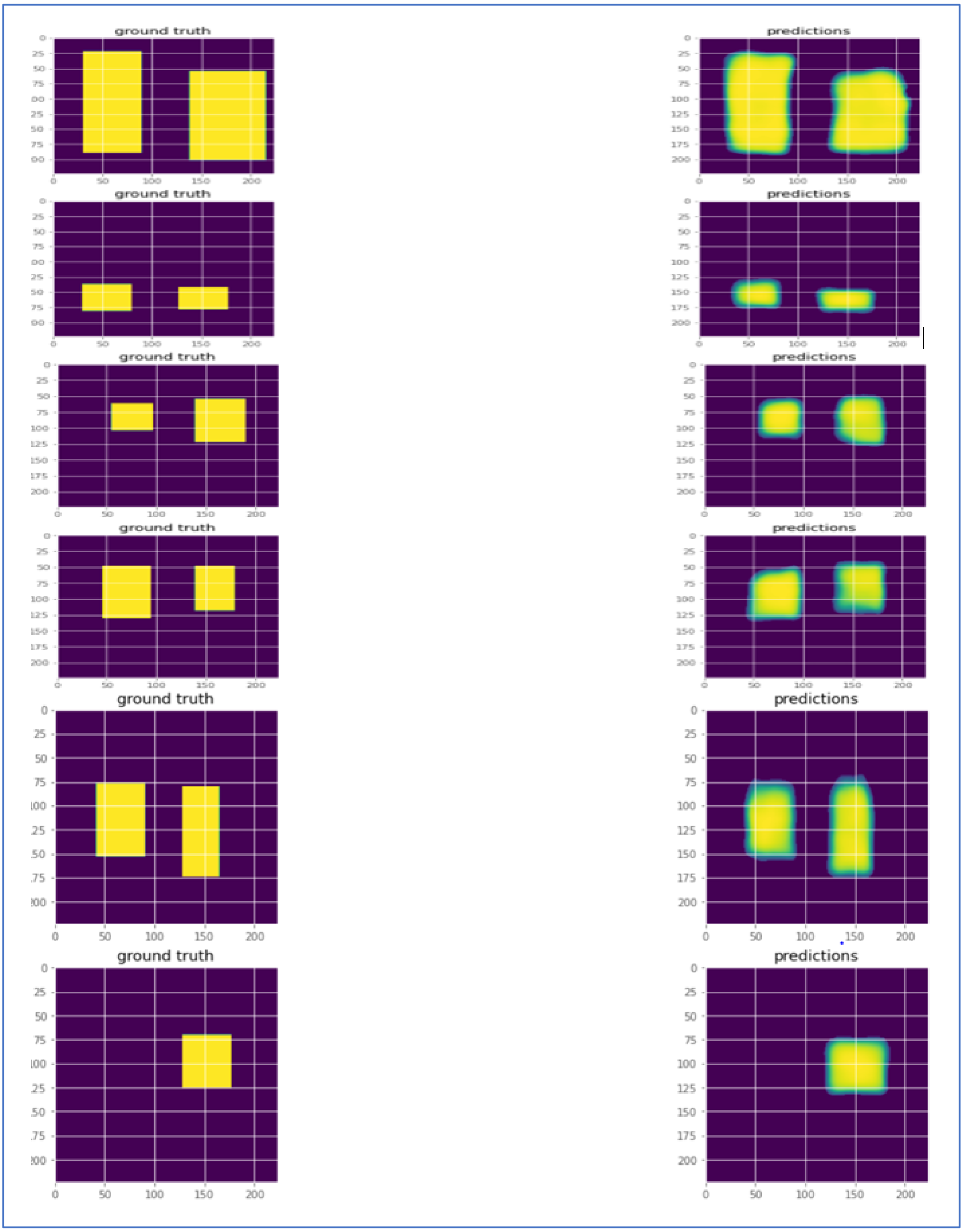
- After loading the whole dataset, we are selecting those records where the target is equals to 1 i.e. pneumonia cases.
- We are storing all the images after resizing it to 224 * 224 into an array and storing all the targets into an array. Here targets are not the 4 coordinates instead they are the whole masks.
- We are splitting the data into train, test and validation sets.
- We are now initializing the u-net and setting the output to the size of our input mask.
- We are now defining some losses, complier and accuracy so that our model can be trained at its best and then finally we are evaluating the model.

Loss	**mean_squared_error**
Optimizer	**adam**
Metrics	**Accuracy**

- We have trained our model with epochs = 30 and then finally we are evaluating the model.
- After everything we are then using visualization tools to see the predictions and the ground truth.
- Here in visualization we are using 20% as a confidence interval.



Ground Truth VS Predictions:



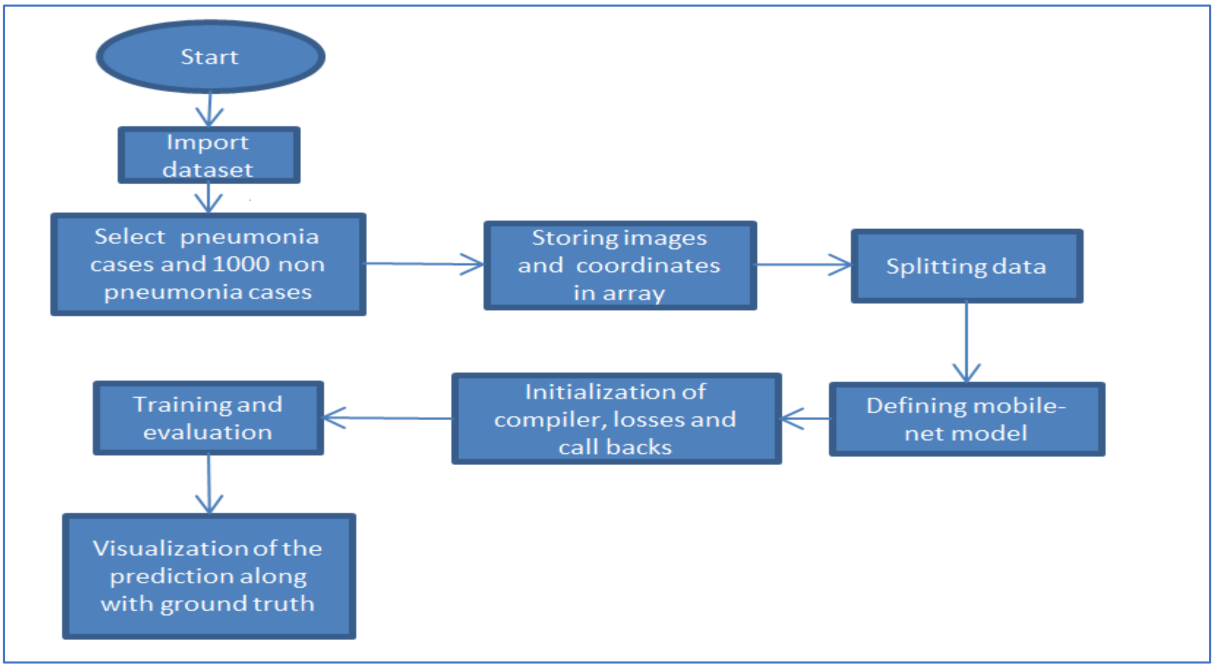
b) Mobile-Net:

Process Flow:

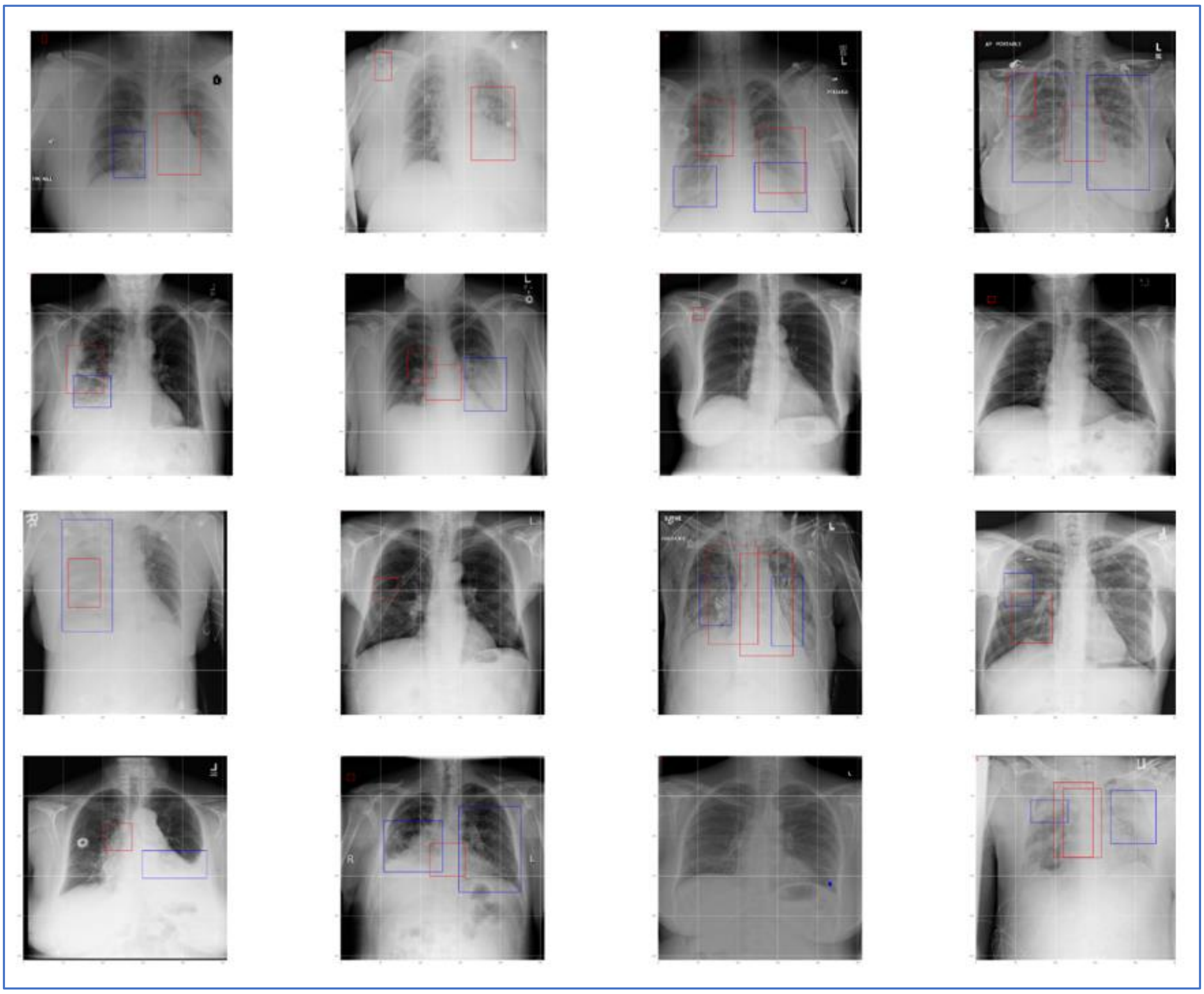
- After loading the whole dataset, we are selecting those records where the target is equals to 1 i.e. pneumonia cases.
- Apart from considering pneumonia cases, we are also considering 1000 non-pneumonia cases for better training.
- We are storing all the images after resizing them to 256*256 into an array and storing all the targets into an array. Here target are the 4 coordinates but not the masks as was in the case of u-net
- Then we are splitting the data into train, test and validation sets
- We are then initialization the mobile-net and setting the output to the size of our target shape
- After that we are defining some losses, complier and accuracy so that our model can train at its best.

Loss	**mean_squared_error**
Optimizer	**adam**
Metrics	**Accuracy**

- Then we are training our model with epochs = 50 and then finally we are evaluating the model.
- After everything we are then using visualization tools to see the predictions and the ground truth.



Ground Truth VS Predictions:

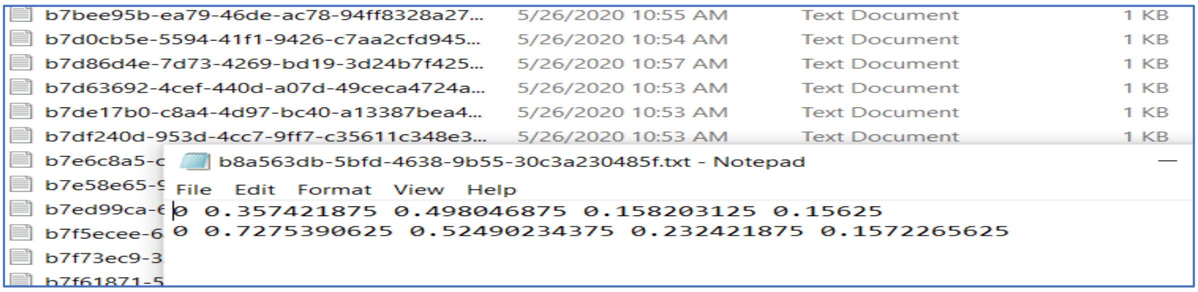


Blue bounding boxes represent ground truth and red bounding boxes represents predictions

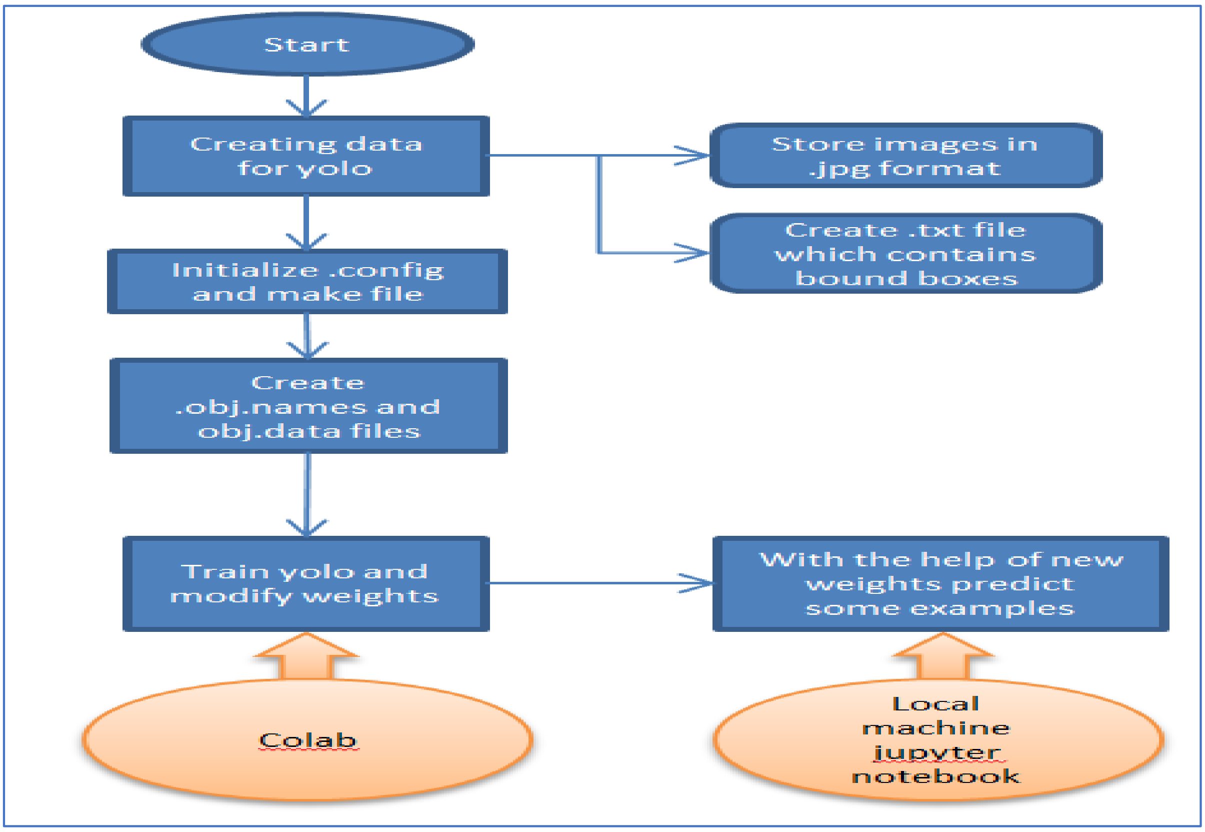
c) Yolo:

Process Flow:

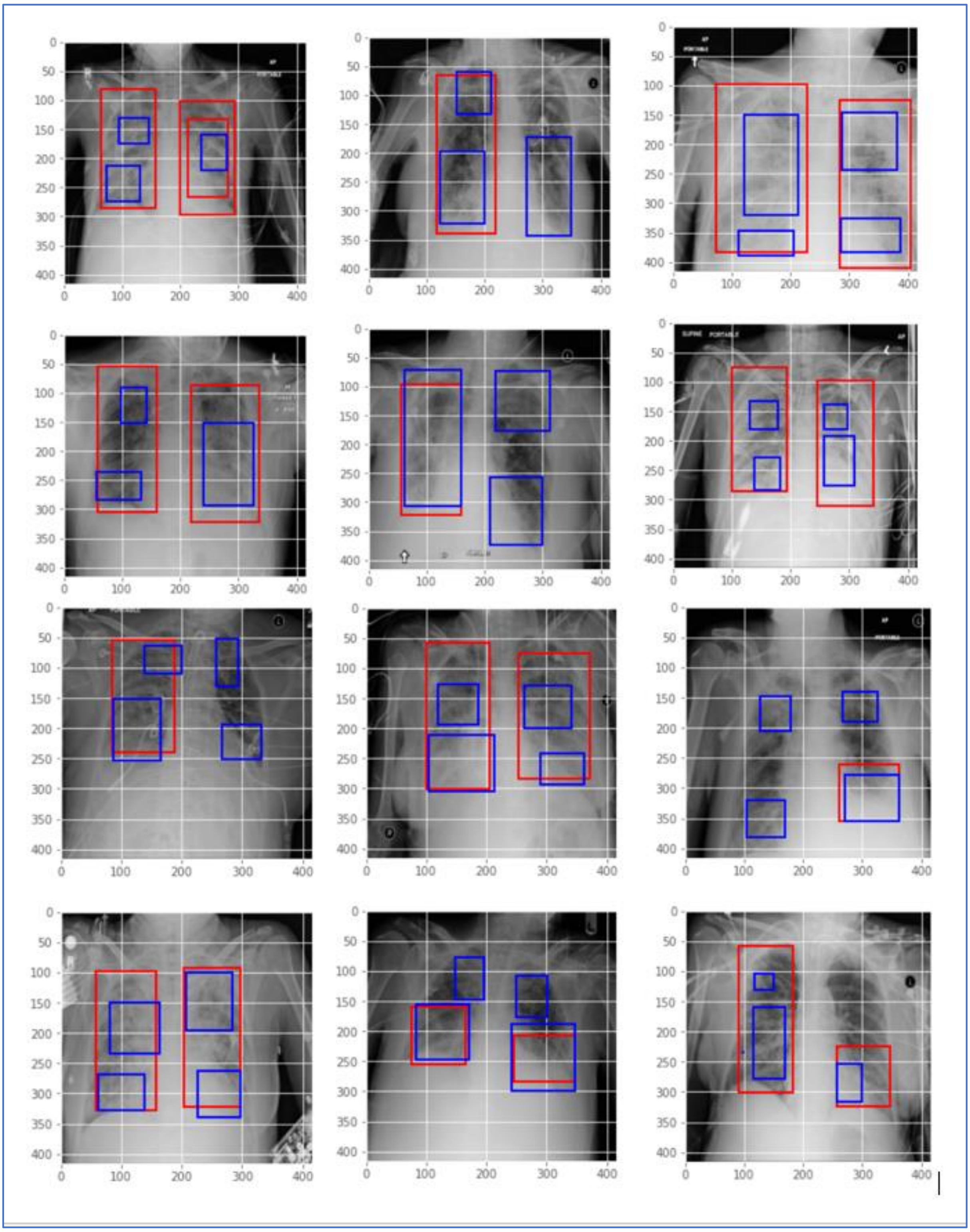
- Here we are first creating data for the yolo
- We are resizing all the images to 416*416 and are storing them in a .jpg format to a folder
- We are taking all the coordinates for pneumonia cases and strong them in a text file
- Yolo takes input as a .jpg file and target as txt file
- We are only considering pneumonia cases.



- We have then trained the yolo in colab for that we make initialize Yolo to create file and config file.
- We have created obj.name and obj.data files as these files are required by the yolo.
- We are now training yolo model to 2000 epochs and modifying pre-trained coco weights to our dataset and then we are saving the modified weights.
- After the training we are predicting some of the examples and comparing it with the ground truth.
- Here, we have done training in google-colab and predictions in jupyter notebook in local machine.



Ground Truth VS Predictions:

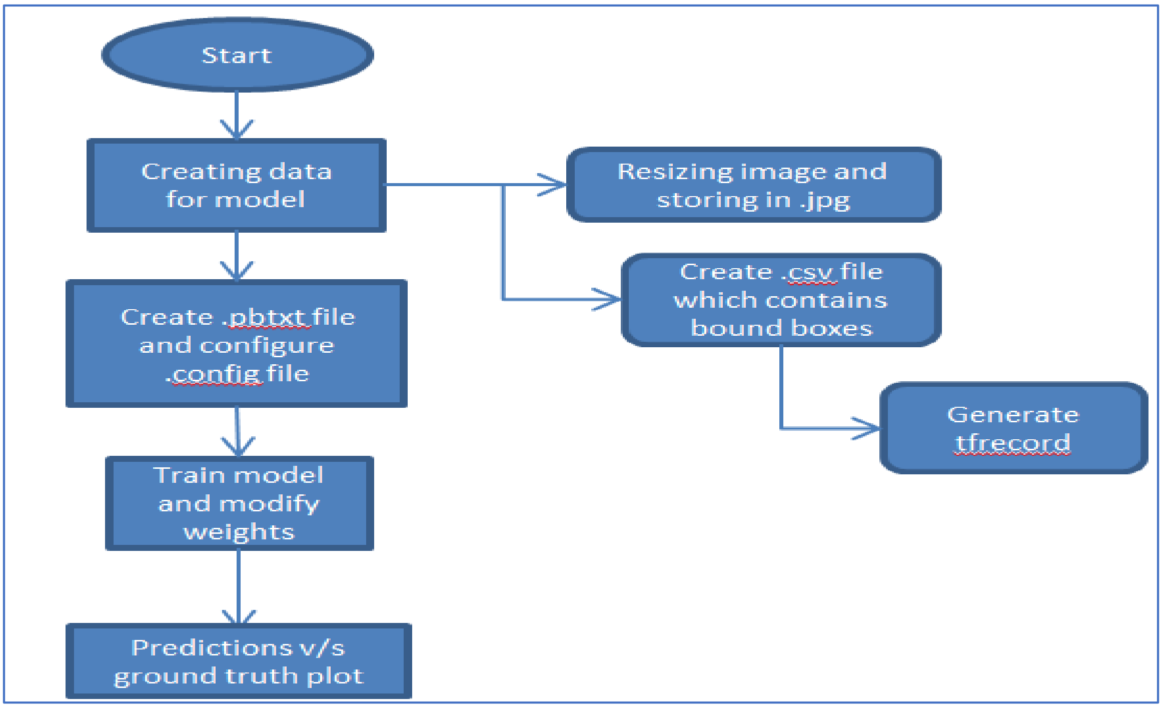


Blue bounding boxes represents ground truth and red boxes represents predictions

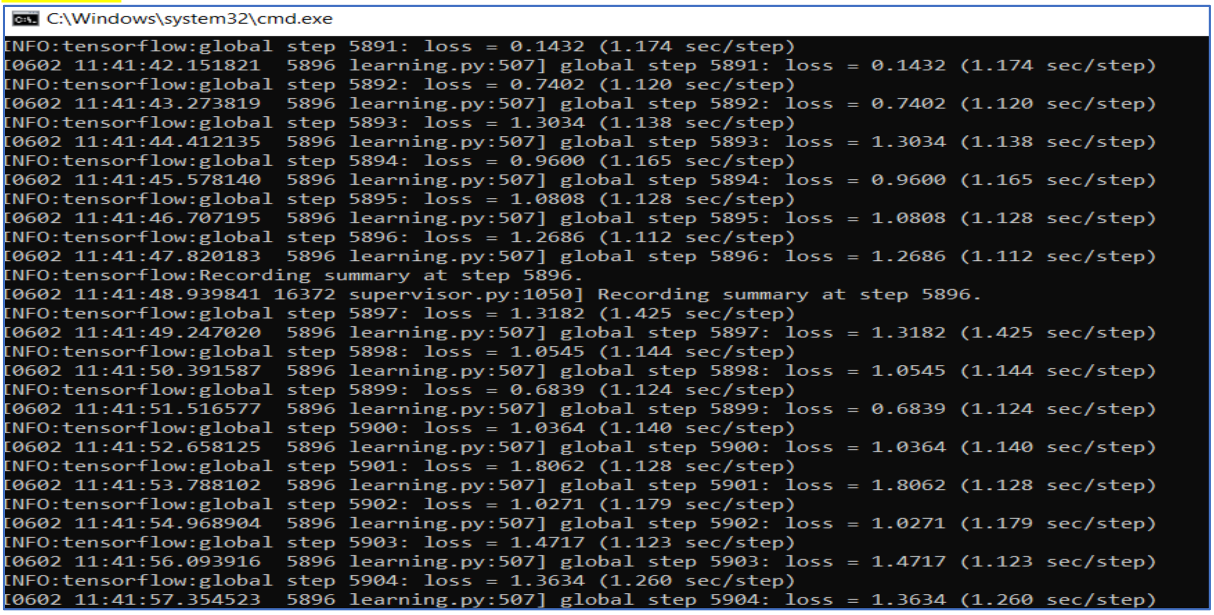
d) Faster- RCNN:

Process Flow:

- **Note:** Here we have use tensor flow obj detection API for Faster RCNN.
- Here we are first resizing images to 560*560 and storing them in .jpg format.
- Then we are considering only pneumonia cases and storing coordinates in a specific way i.e. Xmin, ymin, ymax, ymin into .csv file
- Now we are transforming .csv file to tf record file which will be used by the model
- Later we are creating a .pbtxt file and manipulating .config file of the model.
- We are then training the model till 5900 epochs and updating the pre trained weights of coco dataset to our data.
- Then we are predicting some examples and plot them along with the ground truth

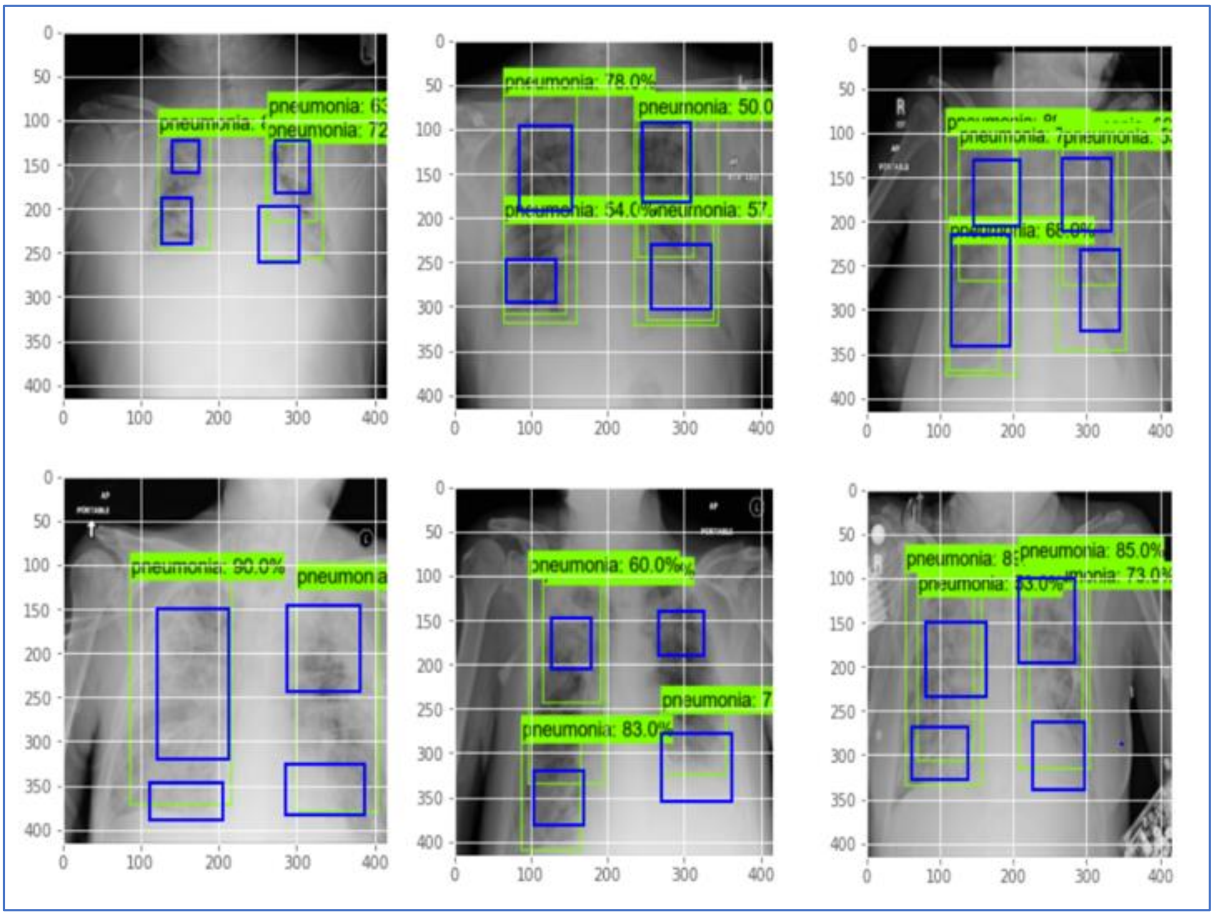


Training:



Losses:

Ground Truth VS Predictions:



Note: Blue bound box represents ground truth and Green boxes represents predicted boxes

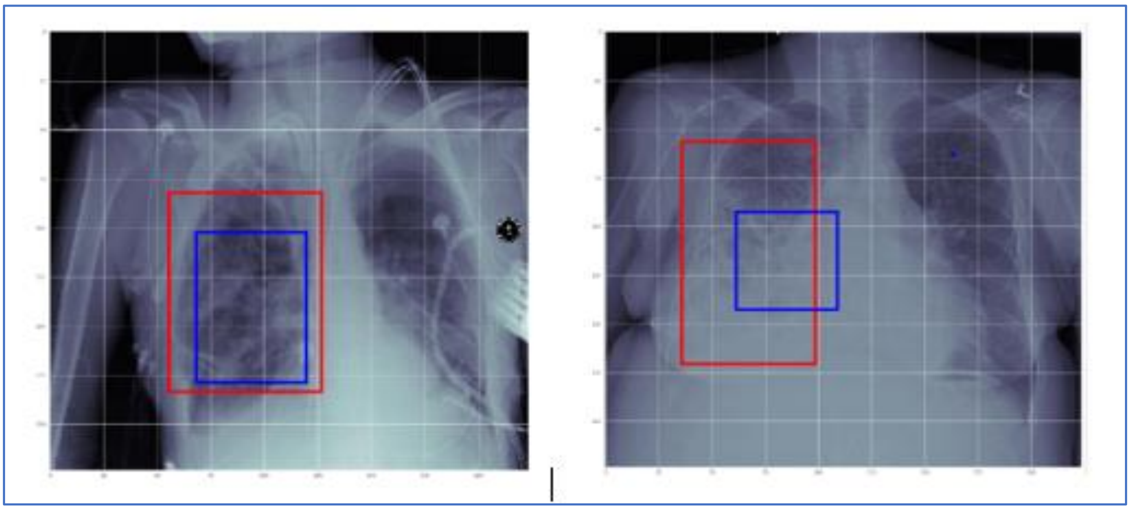
e) CheXNet (DenseNet - 121)

Process Flow:

- We are loading dataset only the images with Target = 1 for this prediction. We have also noticed that there were multiple images for some patients hence we selected only one unique image for such patients.
- Finally, we came up with 9000 unique patient ids and 1 image corresponding to each patient. This dataset was further broken into Training, Validation and Test sets.
- We are storing all the images after resizing it to 224 * 224 into an array and storing all the targets into an array.
- We decided to train the model on Dense net 121 architecture and replace the top layer with a regression that would predict the 4 dimensions of the bounding box i.e. x and y coordinates of the top left corner of the box, its width and its height.
- Depth of the top layers: Initially we started we a model where the top layer was a dense layer with 4 output nodes and no activation and slowly we started increasing the depth of the top node while monitoring the overall accuracy and eventually finalized a top layer which contains a few normalization layers, 2D convolution layers with Relu-activations and a final regressor layer.
- Regressor Layer: We started with a dense layer having 4 output nodes for the regression but it did not provide a very good accuracy but what provided a better accuracy and final bounding boxes was a 2D convolution layer with 4 output feature maps and kernel size same as the output of the layer previous to this. So, we finally used a 2D convolution layer for the regression.

Loss	**mean_absolute_error**
Optimizer	**adam**
Metrics	**Accuracy**

Ground Truth VS Predictions:



Note: Red bound box represents ground truth and Blue boxes represents predicted boxes

f) Mask RCNN

Process Flow:

- 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.
- Let's first quickly understand how Faster R-CNN works. This will help us grasp the intuition behind Mask R-CNN as well.
- Faster R-CNN first uses a Convent 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.
- Once you understand how Faster R-CNN works, understanding Mask R-CNN will be very easy. So, let's understand it step-by-step starting from the input to predicting the class label, bounding box, and object mask.
- There are few custom settings that can be adjusted to better train our model.

```
# These parameters are selected to reduce running time
class DetectorConfig(Config):
    """Configuration for training pneumonia detection on the RSNA pneumonia dataset.
    Overrides values in the base Config class.
    """

    NAME = 'pneumonia'
    # Train on 1 GPU and 8 images per GPU. We can put multiple images on each
    # GPU because the images are small. Batch size is 8 (GPUs * images/GPU).
    GPU_COUNT = 1
    IMAGES_PER_GPU = 8
    BACKBONE = 'resnet50'
    NUM_CLASSES = 2  # background + 1 pneumonia classes
    # Use small images for faster training. Set the limits of the small side
    # the large side, and that determines the image shape.
    IMAGE_MIN_DIM = 64
    IMAGE_MAX_DIM = 64
    RPN_ANCHOR_SCALES = (32, 64)
    TRAIN_ROIS_PER_IMAGE = 16
    MAX_GT_INSTANCES = 3
    DETECTION_MAX_INSTANCES = 3
    DETECTION_MIN_CONFIDENCE = 0.78
    DETECTION_NMS_THRESHOLD = 0.01
    RPN_TRAIN_ANCHORS_PER_IMAGE = 16
    STEPS_PER_EPOCH = 100
    TOP_DOWN_PYRAMID_SIZE = 16
    STEPS_PER_EPOCH = 100

config = DetectorConfig()
config.display()
```

- When the Model is Trained on 15 EPOCHS with 100 Steps Per Epoch, the results are as follows:

	val_loss	val_rpn_class_loss	val_rpn_bbox_loss	val_mrcnn_class_loss	val_mrcnn_bbox_loss	val_mrcnn_mask_loss	loss	rpn_class_loss	rpn_bbox_loss	mrcnn_class_loss	mrcnn_bbox_loss	mrcnn_mask_loss
1	2.975009	0.403969	1.188390	0.178276	0.640973	0.563382	3.558728	0.629081	1.334952	0.307143	0.656653	0.630879
2	2.726907	0.331663	1.036669	0.227559	0.582540	0.548457	2.741418	0.368791	0.945038	0.274829	0.572149	0.580593
3	3.049220	0.452373	1.274594	0.190049	0.523431	0.608755	2.551130	0.339341	0.865433	0.258185	0.531949	0.556203
4	2.605792	0.295334	1.024181	0.218651	0.554999	0.512608	2.521012	0.337245	0.884397	0.226699	0.523610	0.549042
5	2.535554	0.325270	0.937983	0.237720	0.544833	0.489729	2.458056	0.309780	0.875331	0.234808	0.512762	0.525356
6	2.618927	0.290956	1.027531	0.257374	0.551646	0.491401	2.311417	0.278667	0.783358	0.232623	0.501574	0.515177
7	2.462571	0.286155	0.919370	0.226765	0.533981	0.496282	2.282546	0.287721	0.767512	0.240179	0.483156	0.503959
8	2.396509	0.292845	0.912154	0.230249	0.497032	0.464211	2.257514	0.288146	0.764128	0.224388	0.473845	0.506988
9	2.418903	0.280946	0.935773	0.234396	0.497873	0.469896	2.152728	0.269219	0.709288	0.223433	0.464076	0.486693
10	2.389229	0.305353	0.957173	0.197564	0.470257	0.458862	2.174160	0.271248	0.731531	0.223974	0.458080	0.489309
11	2.469944	0.293554	0.973597	0.185513	0.538429	0.478831	2.093622	0.244915	0.726468	0.217926	0.429493	0.474802
12	2.526627	0.293884	1.066018	0.195531	0.495749	0.475426	2.071270	0.249347	0.689211	0.228982	0.434421	0.469289
13	2.322730	0.276068	0.903966	0.217495	0.485098	0.440085	2.057689	0.255275	0.674339	0.229200	0.428942	0.469914
14	2.465410	0.292208	1.014796	0.197940	0.505451	0.454996	1.947395	0.242487	0.643762	0.207510	0.402351	0.451266
15	2.415111	0.280030	1.041665	0.176046	0.474076	0.443276	2.038694	0.243738	0.710427	0.204686	0.416955	0.462869

- Apart from the above-mentioned Model Settings, the model was also trained & tested using the following scenarios to check the Model's performance

Parameters:

DETECTION_MIN_CONFIDENCE = 0.9 and DETECTION_NMS_THRESHOLD = 0.1 and STEPS_PER_EPOCH = 100, for up to 15 EPOCHS

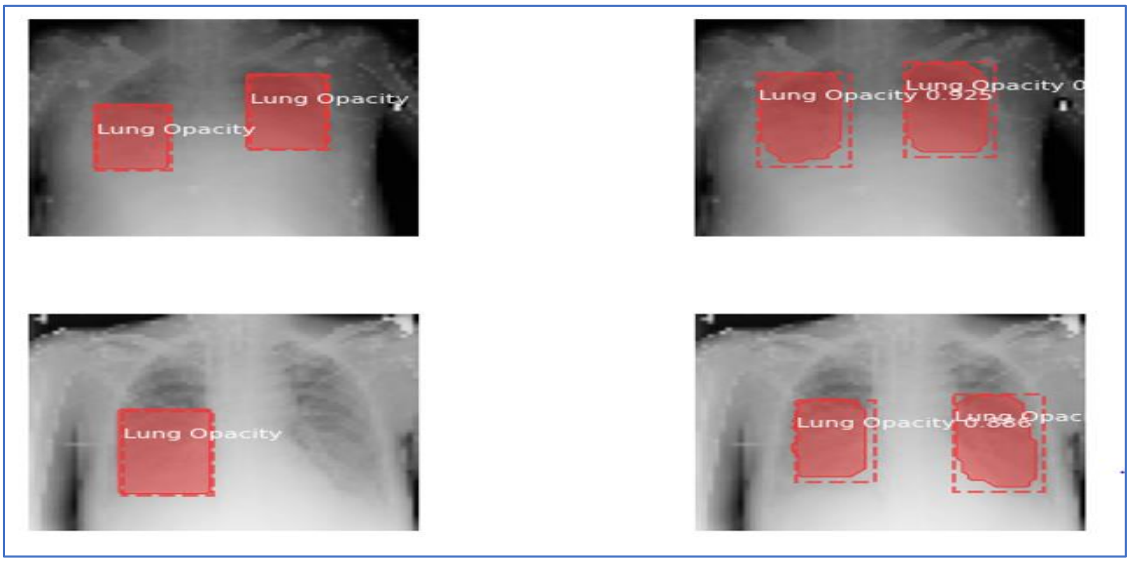
DETECTION_MIN_CONFIDENCE = 0.9 and DETECTION_NMS_THRESHOLD = 0.1 and STEPS_PER_EPOCH = 10 for up to 50 EPOCHS

LEARNING_RATE = 0.006 and STEPS_PER_EPOCH = 200, for up to 16 EPOCHS

The results are as follows when Trained with DETECTION_MIN_CONFIDENCE = 0.9 and DETECTION_NMS_THRESHOLD = 0.1 and STEPS_PER_EPOCH = 100, for up to 15 EPOCHS

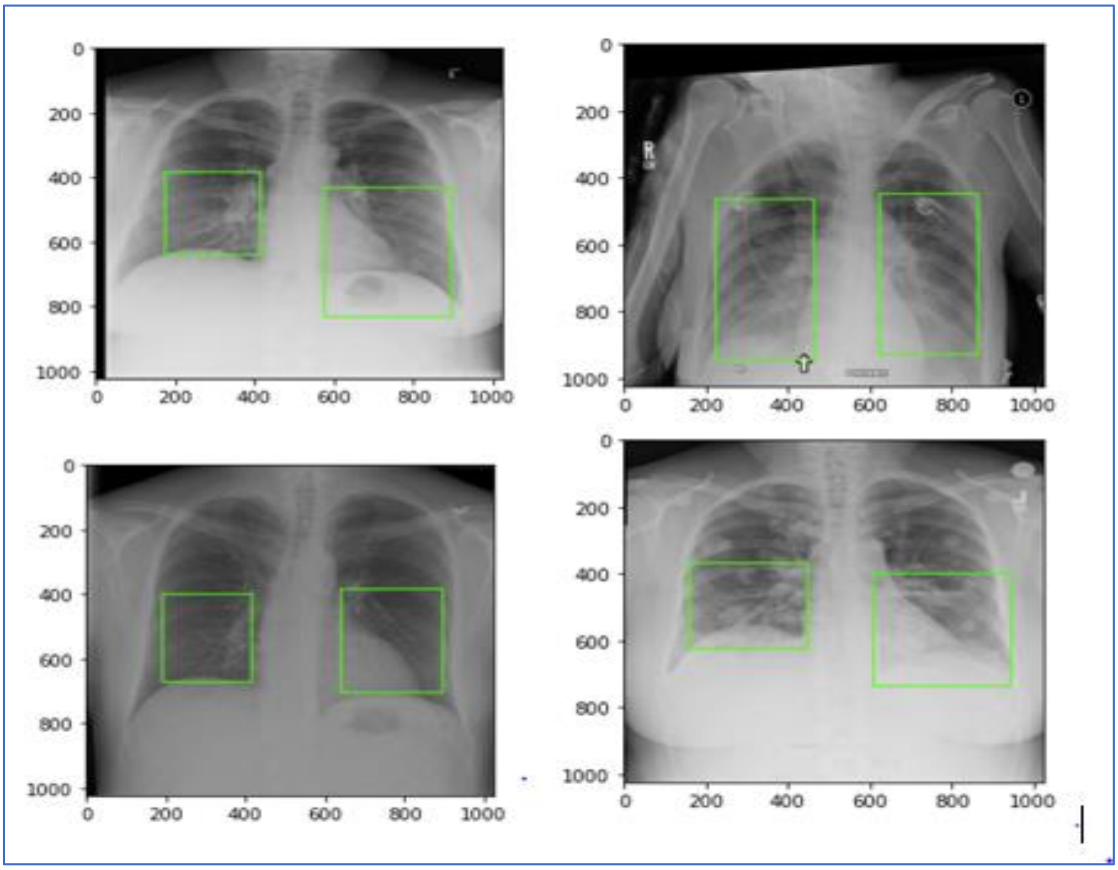
```
# These parameters are selected to reduce running time
class DetectorConfig(Config):
    """Configuration for training pneumonia detection on the RSNA pneumonia dataset.
    Overrides values in the base Config class.
    """

    NAME = 'pneumonia'

    # Train on 1 GPU and 8 images per GPU. We can put multiple images on each
    # GPU because the images are small. Batch size is 8 (GPUs * images/GPU).
    GPU_COUNT = 1
    IMAGES_PER_GPU = 8
    BACKBONE = 'resnet50'
    NUM_CLASSES = 2  # background + 1 pneumonia classes
    # Use small images for faster training. Set the limits of the small side
    # the large side, and that determines the image shape.
    IMAGE_MIN_DIM = 64
    IMAGE_MAX_DIM = 64
    #RPN_ANCHOR_SCALES = (32, 64)
    TRAIN_ROIS_PER_IMAGE = 16
    MAX_GT_INSTANCES = 3
    DETECTION_MAX_INSTANCES = 3
    DETECTION_MIN_CONFIDENCE = 0.9
    DETECTION_NMS_THRESHOLD = 0.1
    RPN_TRAIN_ANCHORS_PER_IMAGE = 16
    STEPS_PER_EPOCH = 100
    TOP_DOWN_PYRAMID_SIZE = 32
    STEPS_PER_EPOCH = 100

config = DetectorConfig()
config.display()
```

Ground Truth VS Predictions:



Note: Left Image is the Ground Truth & Right Image is Prediction

Predictions Made on Test Images:



7. Implications:

- From all the models that we have built, we can see the predicted vs ground truth that is done for object detection for each model to understand it better.
- Basis the threshold value that is being set for the respective models we can see the object detection working.
- However, the object detection varies basis the threshold we have decided, for example for Mask RCNN we have set the threshold to 0.78 and similarly we have different thresholds for other models to detect i.e. U-Net has 20% and Yolo has 10% and Faster RCNN has about 50% confidence.
- If we increase the threshold or confidence level to 0.90, the number of detections being made are going down to one detection, so in-order to have multiple detections we have set it to 0.78 for Mask RCNN. Similarly, for U-Net we have observed the confidence level of 20% but the accuracy is high.
- Our solution gives user a better feasibility to understand if he/she is suffering from Pneumonia or not and we don't miss on False negatives in our detection.
- In our project we have tried various models, since Yolo serves as a benchmark for object detection, but it is used in multiple chest disease detection of 14 classes.
- We have customised our model for object detection, since we are only anticipating one class disease i.e. Pneumonia.
- In case the user wants to have a check-up done and if he can be provided with the image, he can do an object detection right away.
- This would in-turn lead to reducing the "Turn Around Time", and the user can at least have a fair idea if he/she is suffering from pneumonia.

8. Limitations:

We have limitations individually for each model which has been explained below for one and all:

a. U-Net:

- Its architecture is complexed, so it Consumes much of processing time to train.
- It requires input target as a mask. Therefore, it requires lot of processing time and memory.

b. Mobile-Net:

- It is light weight model thus it sometimes leads to overfitting.
- Its accuracy is very less comparing to other models.

c. Yolo:

- Darknet implementation yolo is very time consuming when it comes to train
- It needs lots of processing power as well
- It requires output in some format like it needs .txt file
- We need to configure lot of files like .config and .make files

d. Faster RCNN:

- TensorFlow objection detection Fasterrcnn takes lots of time to train.
- It needs lots of processing power as well, which is again a turn down.
- It needs lots of data preprocessing like first it needs .csv then we need convert that .csv to tf records file.
- We need to configure lot of files like .config and pbtxt files.

e. CheXNet:

- DenseNet is not primarily used for bounding box prediction
- CheXNet originally had 14 classes whereas here we only have 1 class, so this was a major challenge.

f. Mask RCNN:

- MASK-RCNN is not compatible with certain TensorFlow Versions: Training part of the Model didn't work with TensorFlow 2.2.0 version. When downgraded to TensorFlow 1.14.0, it worked fine.

Apart from the model limitations we have also had other issues such as:

- Colab Limit and run time issues due to heavy data.
- Training time has been a huge time taking process where in connectivity issues had also contributed.

Enhancement of Solution:

To enhance our solution, we can make following considerations to have even better solution:

1. We can apply the Oops concepts to have a better programming structure.
2. We can Create different files to overcome reliability such as Setup files and run files and package files can be run individually to maintain code modularity.
3. All this can be achieved using the python scripts which can help us have code in a better fashion and indeed even the maintenance would be easy.
4. We can create API such as Anvil for the user friendliness. Such that the user can upload the image and can get to have an idea if the report has any symptoms of Pneumonia.

9. Closing Reflections:

Learnings from the Process:

We have got to know how the AI could help an individual and the Health Care System to make our lives even better, its contribution seems to be remarkable. How a model would behave while doing the classification and object detection. The resizing of images to have a proper data. Also, how to visualise the models and understand the different losses that would help us improve our solution. Also trying to align our data as per the pre-built architecture for each model, this was a major learning that was very much needed and helped us understand the work pattern. We have also tried a varied number of solutions rather than limiting ourselves to 1 or 2 models. This has helped us explore and understand the developed models in a better fashion.

Different models have different challenges. However, we have listed few difficulties that we have faced during our project work.

1. Firstly, Challenge is to set different inputs for each:
   - In case of **YOLO,** we need to make .txt for every co-ordinate.
   - In case of **U-Net,** we need a mask as an input. So, we must set a limit to the input data as mask is of size of the image and if we supply more data the system was crashing.
   - In case of **Faster-RCNN**, we need coordinates to be in specific format and in .csv file, and then again .csv file needs to be converted into tf record file.
   - CheXNet originally had 14 classes whereas here we only have 1 class

2. Secondly, Challenge is how much to train:
   - Most of the models consumed lot of time to train like **YOLO** and **Faster-RCNN**.
   - They had to be run for higher number of epochs for yolo it was 2000 and for faster-rcnn it was 5900 epochs, which is very time consuming.

3. Thirdly, Challenge was how to Configure files before training:
   - Many models like **YOLO** and **Faster-RCNN** require specific steps to train. One of the steps requires to configure lots of files like .config file, .make file.
   - In both yolo and faster-rcnn, initializing lots of parameter like alpha in mobile net and creating new files like obj.names and obj. data in yolo.

4. Finally, Challenge was Prediction v/s ground-truth plots:
   - As every model has different inputs and different output so plotting prediction v/s ground-truth plots were as challenging
   - Dense Net is not primarily used for bounding box prediction.

Different Next time:

For there are many things that need to be learned apart from what we have accomplished, but we have few ideas in mind such as:

a. We would like to create furthermore new models apart from the ones that are already existing, by developing each individual layer by even understanding the underlying math behind the screen.
b. We can try to Ensemble Models and create hybrid models, which would behave as a collection of many models giving a collective decision.

c. We would like to create a user API for the user to upload the image and understand if he/she has been affected by pneumonia, they can even get to know the prediction percentage while doing so.