Final Report

SECTION 1 - Summary of problem statement, data and findings

Problem statement

The Project goal is to build a pneumonia detection system, to locate the position of lung inflammation in the Patients lungs X-ray Image. Tissues with sparse material, such as lungs which filled with air, do not absorb the X-rays and appear black in the image. However dense tissues such as bones absorb X-rays and appear white in the image. In this project we attempt to detect lungs opacity using Computer Vision. We have leveraged transfer learning methodology using Mask RCNN and attempted to identify lungs opacities by training a model with the DICOM images and there after testing the models and identifying the precision value to find out the model accuracy.

Background - Why is Pneumonia Detection required through Machine learning?

Pneumonia has a high rate of mortality within children below five years of age worldwide. There were several cases leading to emergencies in the United States causing many deaths and justifying that the disease is one of the leading causes of death in the country.

While detecting Pneumonia is a difficult task to work on, it requires highly specialized medical practitioners to review the Chest radiographs and compare the medical history to determine the disease. In the current scenario many developing and underdeveloped countries are also experiencing the high rate of mortality due to a surge in the number of cases and gap in the availability of medical practitioners causing delay in disease diagnosis. Due to this the disease diagnosis gets delayed resulting in the increase in severity level of the patient's health condition.

To curb these situations AI models are developed to identify the presence of the disease through AI models which would then support medical practitioners in faster diagnosis of the disease. Thereby resulting in medication in early stages and reducing the risk of disease severity.

Data and findings

We are using the RSNA Pneumonia Detection Challenge dataset from kaggle Link of the dataset: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data

In the data, some of these are labeled "Not Normal No Lung Opacity". This extra class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia.

- DICOM original images: - 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.

We are using 2 .csv file from the dataset

- stage_2_detailed_class_info.csv renamed into detailed_class_info.csv
- stage_2_train_lebels.csv renamed into train_lebels.csv

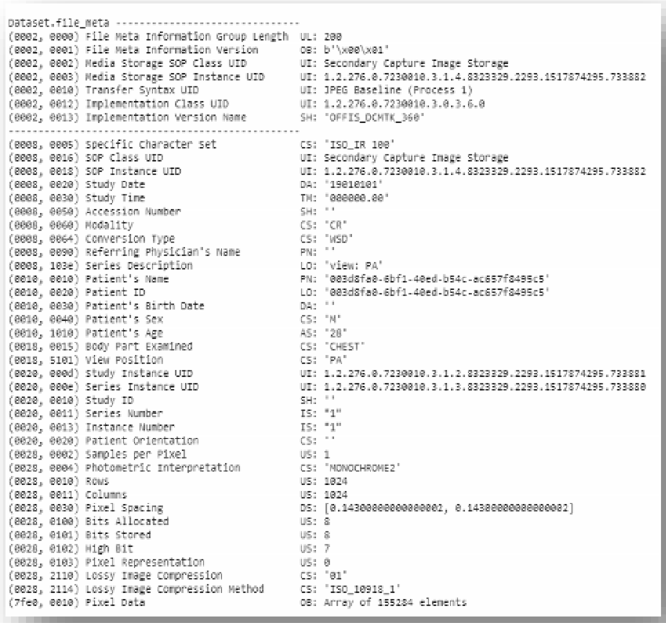
Also we have the set of training images under folder named stage_2_train_image. The training data is a set of **patientIds** and bounding boxes. Bounding boxes are defined in the format of **x-min, y-min, width, height**. There is also a binary target column, indicating pneumonia or non-pneumonia.

Data field’s description -

- **patientId** _- A patientId. Each patientId corresponds to a unique image.
- **x_** - the upper-left x coordinate of the bounding box.
- **y_** - the upper-left y coordinate of the bounding box.
- **width_** - the width of the bounding box.
- **height_** - the height of the bounding box.
- **Target_** - the binary Target, indicating whether this sample has evidence of pneumonia.

In the data, some of these are labeled “Not Normal No Lung Opacity”. This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia.

Medical images are stored in a special format known as DICOM files (*.dcm). They contain a combination of header metadata as well as underlying raw image arrays for pixel data. In Python, one popular library to access and manipulate DICOM files is the pydicom module. To use the pydicom library, first we need to find the DICOM file for a given patientId by simply looking for the matching file in the train_images/ folder, and the use the pydicom.read_file() method to load the data:

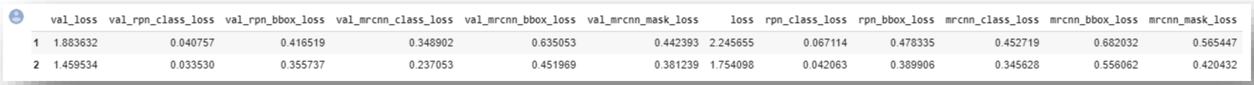


```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 200
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.2293.1517874295.733882
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.2293.1517874295.733882
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '003d8fa0-6bf1-40ed-b54c-ac657f8495c5'
(0010, 0020) Patient ID                          LO: '003d8fa0-6bf1-40ed-b54c-ac657f8495c5'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'M'
(0010, 1010) Patient's Age                       AS: '28'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.2293.1517874295.733881
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.2293.1517874295.733880
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: "1"
(0020, 0013) Instance Number                     IS: "1"
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 155284 elements
```

Figure 1 DICOM Image Distribution

What was intended in the outset?

The Intention in the outset was to identify Pneumonia through the blockage in the X-ray images of lungs. We used Mask RCNN technique, where we deployed transfer learning process over the model trained in COCO dataset. This model was successful in identifying the result. However, there were limitation in terms of accuracy. We were more able to find the reduction in Loss as shown in the below snip.



	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	1.883632	0.040757	0.416519	0.348902	0.635053	0.442393	2.245655	0.067114	0.478335	0.452719	0.682032	0.565447
2	1.459534	0.033530	0.355737	0.237053	0.451969	0.381239	1.754098	0.042063	0.389906	0.345628	0.556062	0.420432

Figure 2: Losses in various Epochs

Now here in this model the outset was to identify Pneumonia using Mask RCNN, we then passed the validation images to the trained model and found the lung opacity to determine Pneumonia in the patients. As shown in the below image. We have then calculated the IOU value for which we were able to IOU % of 70.7% for the lung opacity.

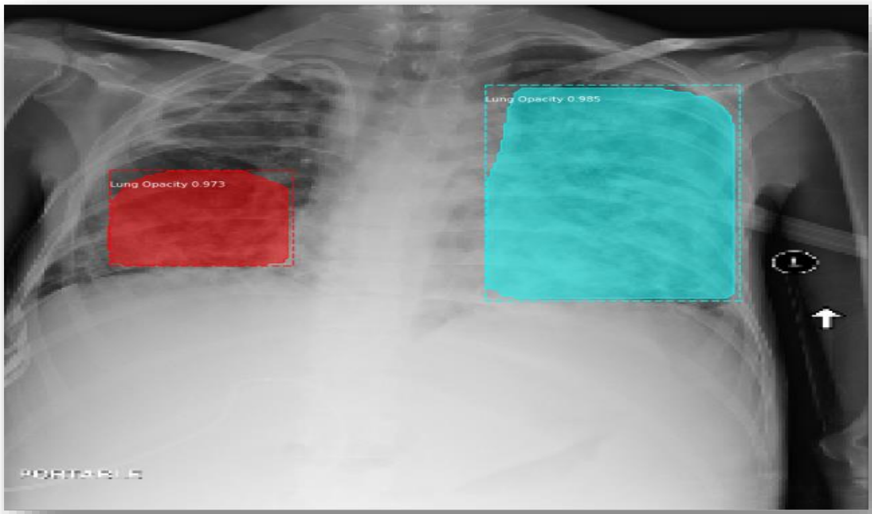


Figure 3: Identifying lung opacity in the validation images

SECTION 2 – Overview of the Final Process

Problem Methodology:

Base Line Model:

We have made our base line model as Mask RCNN. The Mask RCNN has been the new state of art in terms of instance segmentation. Mask RCNN is a deep neural network aimed to solve instance segmentation problem in machine learning or computer vision. In other words, it can separate different objects in an image or a video. You give it an image, it gives you the object bounding boxes, classes and masks.

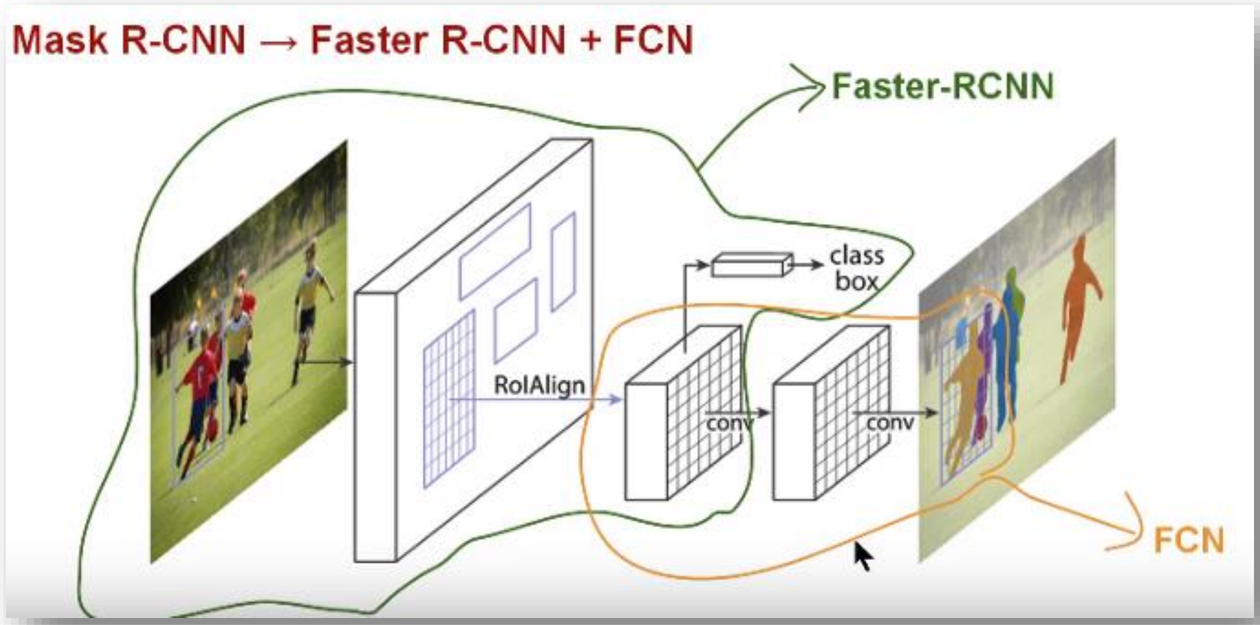
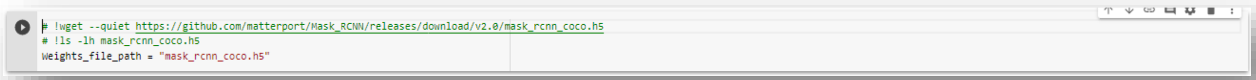


Figure 4: Mask R-CNN concept

Model Selection:

We used Mask RCNN with COCO weight file which is supplied to the model. The detailed explanation to the same is shown in the below attached document that Talks about Mask RCNN. Also adding a below snip where we are cloning the GitHub repository to get all the necessary libraries and python file.



```
# !wget --quiet https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5
# !ls -lh mask_rcnn_coco.h5
Weights_file_path = "mask_rcnn_coco.h5"
```

Figure 5: Snip from the code

Simple Understanding of Mask RCNN

Mask RCNN has been the new state of art in terms of instance segmentation. There are rigorous papers, easy to understand tutorials with good quality open source codes around for your reference. Here I want to share some simple understanding of it to give you a first look.

Mask RCNN is a deep neural network aimed to solve instance segmentation problem in machine learning or computer vision. In other words, it can separate different objects in an image or a video. You give it an image, it gives you the object bounding boxes, classes and masks.

There are two stages of Mask RCNN. First, it generates proposals about the regions where there might be an object based on the input image. Second, it predicts the class of the object, refines the bounding box and generates a mask in pixel level of the object based on the first stage proposal. Both stages are connected to the backbone structure.

What is backbone? Backbone is a FPN style deep neural network. It consists of a bottom-up pathway, a top-bottom pathway and lateral connections. Bottom-up pathway can be any ConvNet, usually ResNet or VGG, which extracts features from raw images. Top-bottom pathway generates feature pyramid map which is similar in size to bottom-up pathway. Lateral connections are convolution and adding operations between two corresponding levels of the two pathways. FPN outperforms other single ConvNets mainly for the reason that it maintains strong semantically features at various resolution scales.

Now let's look at the first stage. A light weight neural network called RPN scans all FPN top-bottom pathway (hereinafter referred to feature map) and proposes regions which may contain objects. That's all it is. While scanning feature map is an efficient way, we need a method to bind features to its raw image location. Here come the anchors. Anchors are a set of boxes with predefined locations and scales relative to images. Ground-truth classes (only object or background binary classified at this stage) and bounding boxes are assigned to individual anchors according to some IOU value. As anchors with different scales bind to different levels of feature map, RPN uses these anchors to figure out where of the feature map 'should' get an object and what size of its bounding box is. Here we may agree that convolving, down sampling and up sampling would keep features staying the same relative locations as the objects in original image, and wouldn't mess them around.

At the second stage, another neural network takes proposed regions by the first stage and assign them to several specific areas of a feature map level, scans these areas, and generates objects classes(multi-categorical classified), bounding boxes and masks. The procedure looks similar to RPN. Differences are that without the help of anchors, stage-two used a trick called ROIAlign to locate the relevant areas of feature map, and there is a branch generating masks for each objects in pixel level. Work completed.

The most inspiring things I found about Mask RCNN is that we could actually force different layers in neural network to learn features with different scales, just like the anchors and ROIAlign, instead of treating layers as black box.

Final Model (Final_pneumonia_detection):

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 other architecture, 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.

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

U-Net architecture is separated in 3 parts:

- The contracting/downsampling path
- Bottleneck
- The expanding/upsampling path

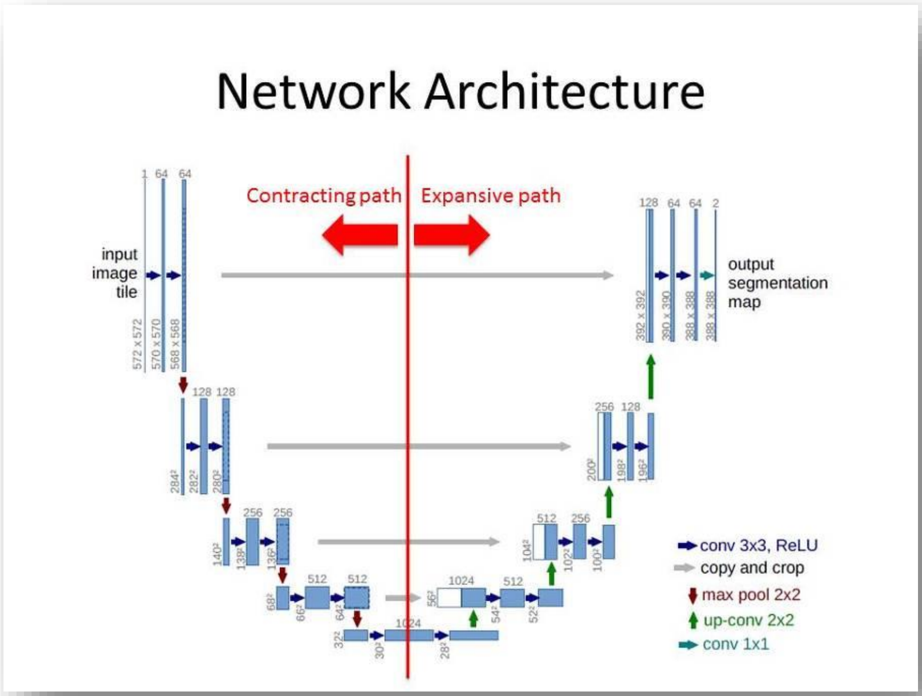


Figure 6: Unet Architecture

Contracting/downsampling path:

The contracting path is composed of 4 blocks. Each block is composed of

- 3x3 Convolution Layer + activation function (with batch normalization)
- 3x3 Convolution Layer + activation function (with batch normalization)
- 2x2 Max Pooling

Note that the number of feature maps doubles at each pooling, starting with 64 feature maps for the first block, 128 for the second, and so on. The purpose of this contracting path is to capture the context of the input image in order to be able to do segmentation. This coarse contextual information will then be transferred to the upsampling path by means of skip connections.

Bottleneck:

This part of the network is between the contracting and expanding paths. The bottleneck is built from simply 2 convolutional layers (with batch normalization), with dropout.

Expanding/upsampling path:

The expanding path is also composed of 4 blocks. Each of these blocks is composed of

- Deconvolution layer with stride 2
- Concatenation with the corresponding cropped feature map from the contracting path
- 3x3 Convolution layer + activation function (with batch normalization)
- 3x3 Convolution layer + activation function (with batch normalization)

The purpose of this expanding path is to enable precise localization combined with contextual information from the contracting path.

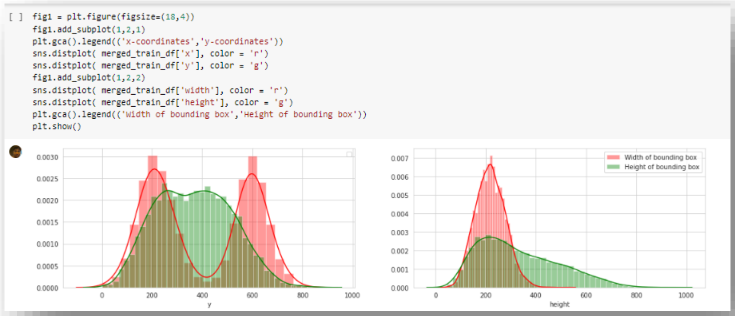
Advantages:

- The U-Net combines the location information from the downsampling path with the contextual information in the upsampling path to finally obtain a general information combining localization and context, which is necessary to predict a good segmentation map.
- No dense layer, so images of different sizes can be used as input (since the only parameters to learn on convolution layers are the kernel, and the size of the kernel is independent from input image' size).
- The use of massive data augmentation is important in domains like biomedical segmentation, since the number of annotated samples is usually limited.

Salient Feature of Data. (Milestone_1_Pneumonia Detection)

Checking the distribution of bounding boxes.

- Y-coordinates are nearly normally distributed where x-coordinates having two picks with almost normal distribution.
- The width of the bounding boxes are a perfect normal distribution but height of the boxes vary and have a long tail on right side.



```
[ ] fig1 = plt.figure(figsize=(18,4))
    fig1.add_subplot(1,2,1)
    plt.gca().legend(('x-coordinates','y-coordinates'))
    sns.distplot( merged_train_df['x'], color = 'r')
    sns.distplot( merged_train_df['y'], color = 'g')
    fig1.add_subplot(1,2,2)
    sns.distplot( merged_train_df['width'], color = 'r')
    sns.distplot( merged_train_df['height'], color = 'g')
    plt.gca().legend(('Width of bounding box','Height of bounding box'))
    plt.show()
```

Figure 7: Distribution of bounding boxes

In this section we have visualized the distribution of class column. This will help us to understand the insight of this column as well as to find out how evenly the data is distributed across different classes.

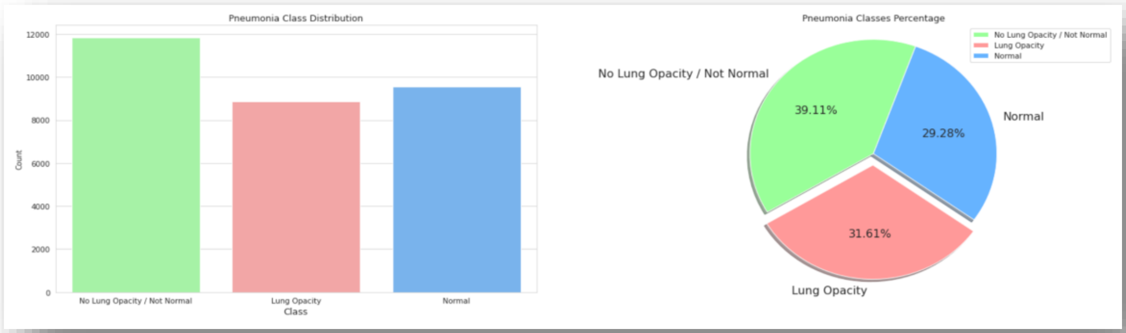
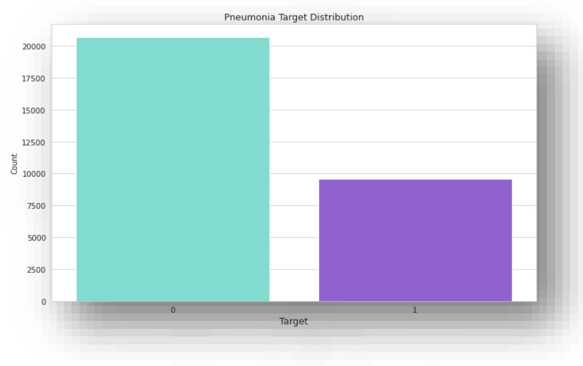


Figure 8: Class Column distribution

Also, we have visualized the distribution of the Target column, this will give us insight on how evenly the dataset is distributed. That will later be consumed for decision making on the utilization of various validation techniques to deal with class imbalance. Techniques like K fold cross validation and use the right evaluation metrics in the end to determine the results. We have shown below the distribution target column.



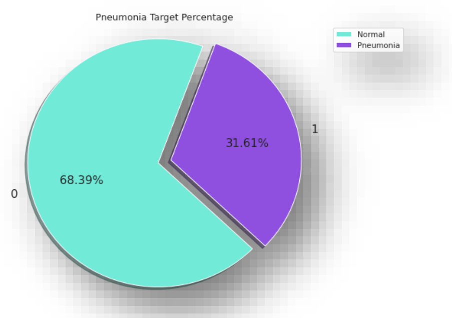
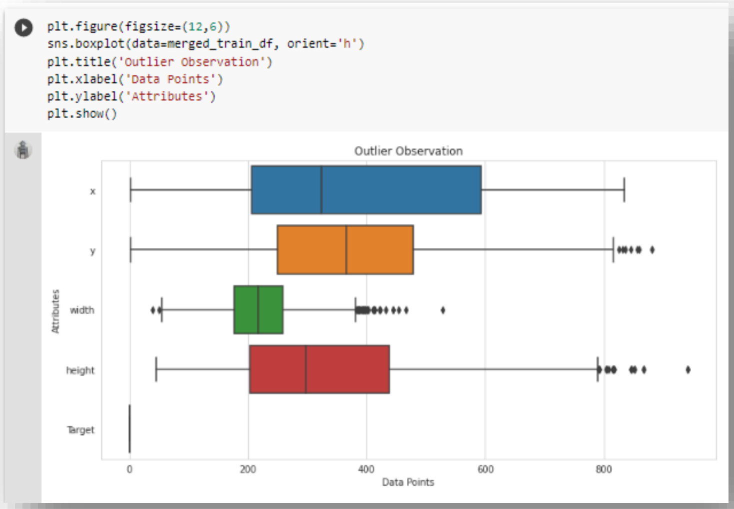


Figure 9: Target Column Distribution

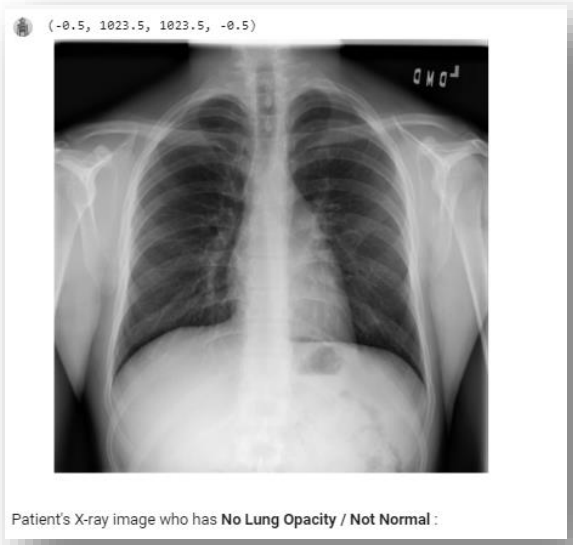
Observe outliers in the bounding boxes as below We can see the outliers for bounding box co-ordinates in below map. At this stage we are not handling outliers since our data for target column is already biased towards Non-Pneumonia. We will handle outliers if required later in performance tuning stage.



```
plt.figure(figsize=(12,6))
sns.boxplot(data=merged_train_df, orient='h')
plt.title('Outlier Observation')
plt.xlabel('Data Points')
plt.ylabel('Attributes')
plt.show()
```

Figure 10: Outlier in Bounding Boxes

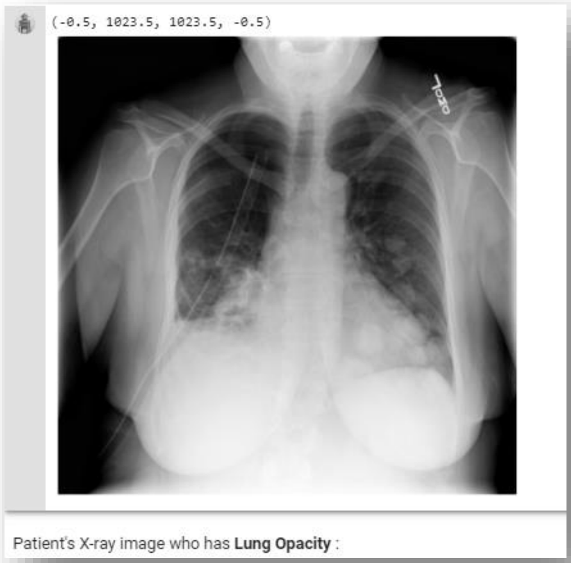
We observed the sample DICOM image as shown below.



```
(-0.5, 1023.5, 1023.5, -0.5)
```

Patient's X-ray image who has **No Lung Opacity / Not Normal** :

Figure 11: X-ray Image with No lung Opacity



```
(-0.5, 1023.5, 1023.5, -0.5)
```

Patient's X-ray image who has **Lung Opacity** :

Figure 12: X-ray Image with lung opacity

We can say from three different X-ray images of three different classes (Normal, No Lung Opacity / Not Normal and Lung Opacity) that X-Ray image with Lung Opacity is looks quite different even lung tissues with sparse material absorbing the X-Rays and appearing white in image.

Let's observe bounding boxes of a patient who has pneumonia in the lungs.

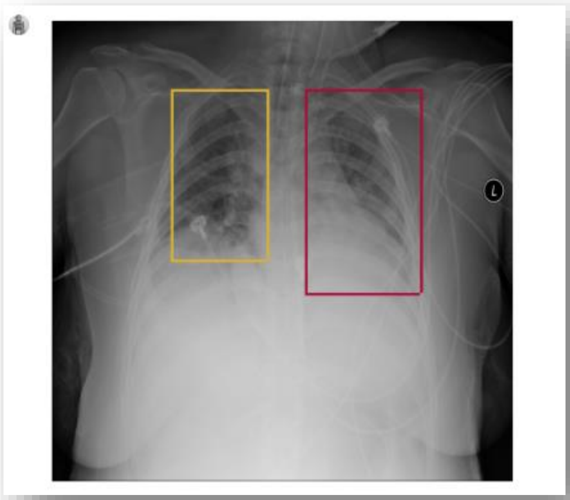


Figure 13: Bounding boxes in the patient having Pneumonia

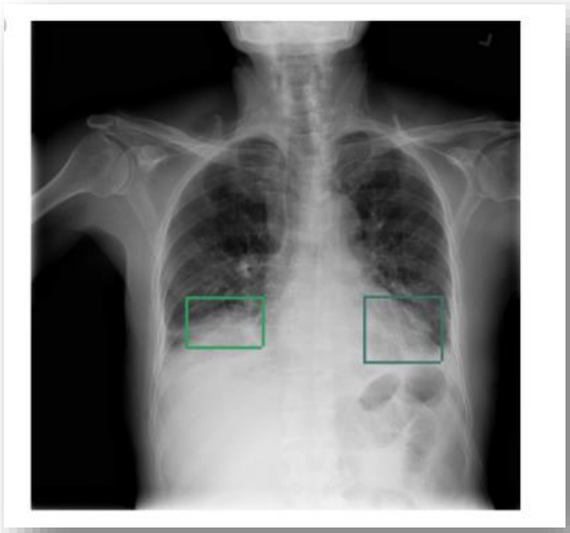


Figure 14: Bounding boxes in the patient having Pneumonia

Visualizing the patient with single row and multiple row.

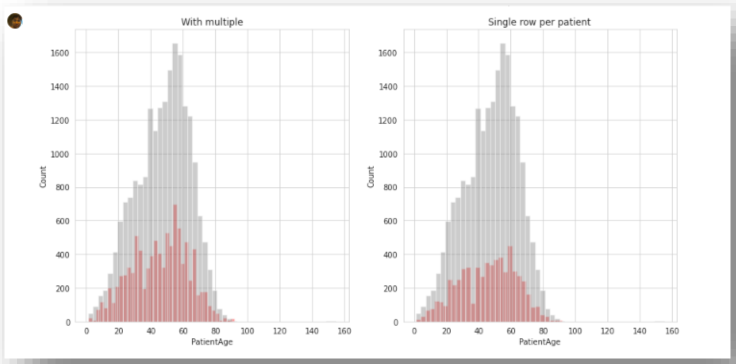


Figure 15: Patient with single row and multiple row

Observing distribution of patient sex with Single row and multiple row.

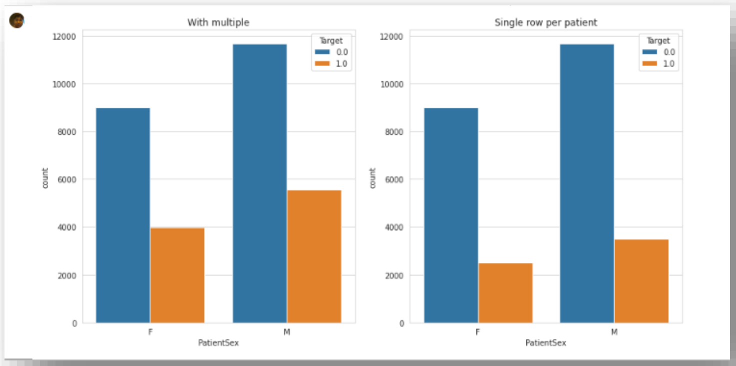


Figure 16: Distribution based on patient Sex

Observing lung opacity with multiple and single row vs. classes of opacity.

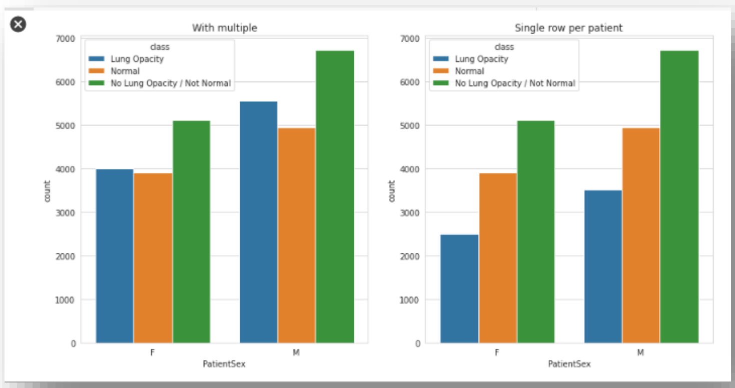


Figure 17: Distribution based on the class

Visualizing the bounding boxes of the target column, some outliers we can see in the box plot in both male and female patients but only for no pneumonia diagnosis patients.

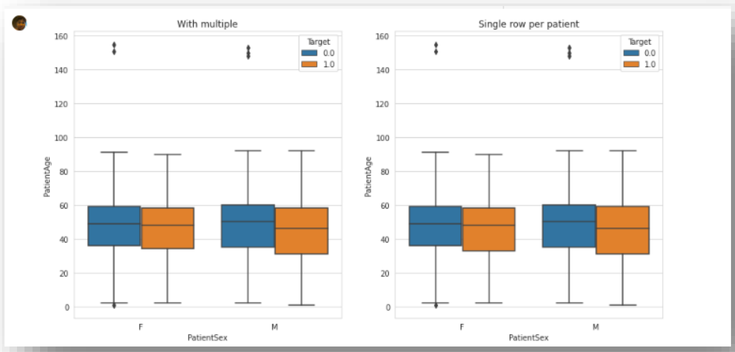


Figure 18: Box plot for Target vs. Patient sex

Visualizing the box plot of patient over three different classes.

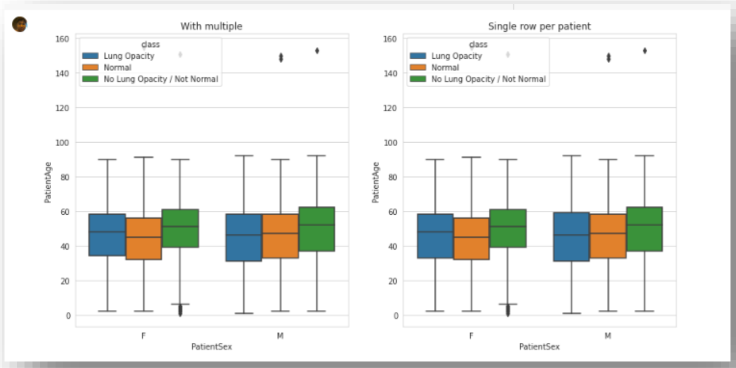


Figure 19: Box plot for different classes vs. Patient Sex

Visualizing the view position with different view position.

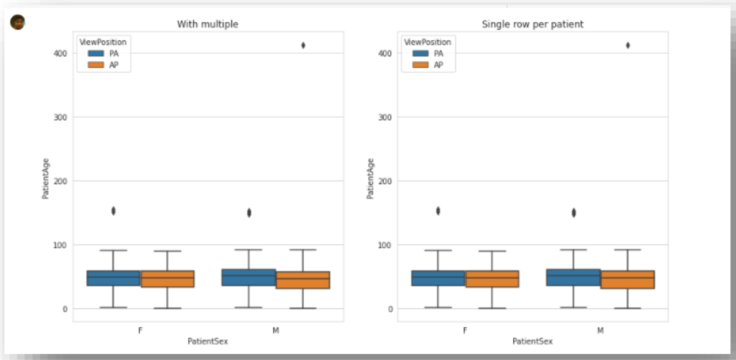


Figure 20: Box plot for various view position

Checking the density of the bounding boxes using contour plot.

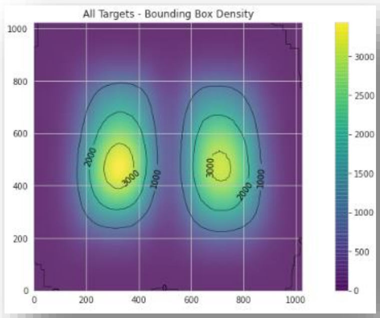


Figure 21: Contour plot for density of bounding boxes

Checking the density of PA and AP. When we checked the bounding box density based on the PA and AP view of x-ray, for PA view the left lung is less affected than right lung and with less area.

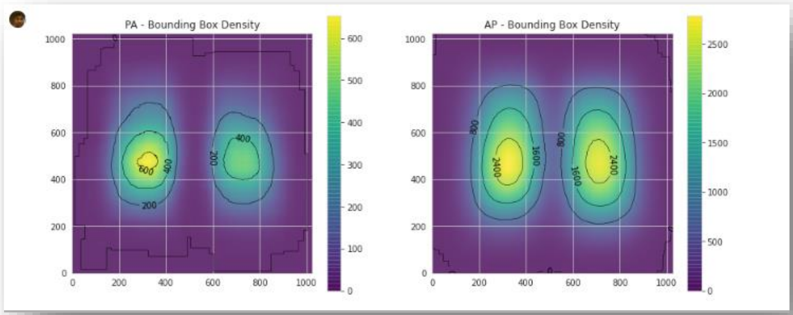


Figure 22: Contour plot in different view position

Checking contour graph and identified that right lungs of the male are more impacted then that of females.

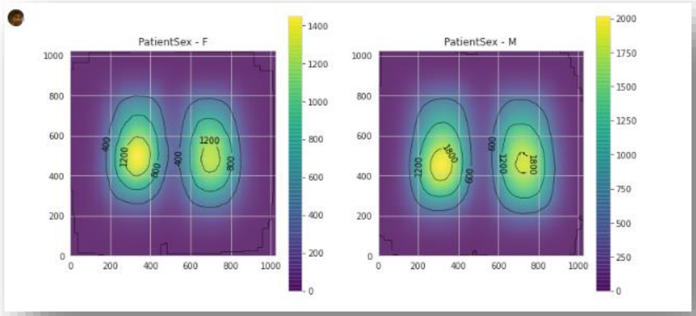


Figure 23: Contour plot showing the impact as per patient sex

Checking contour graph of the patient age as below.

- Patients age greater than 60 have more density of bounding boxes on right lung.
- Patients age between 40 to 60 have same density of bounding boxes but on left lung the area of inflammation is bit more.
- Patients age less than 40 also have more density of bounding boxes on right lung.

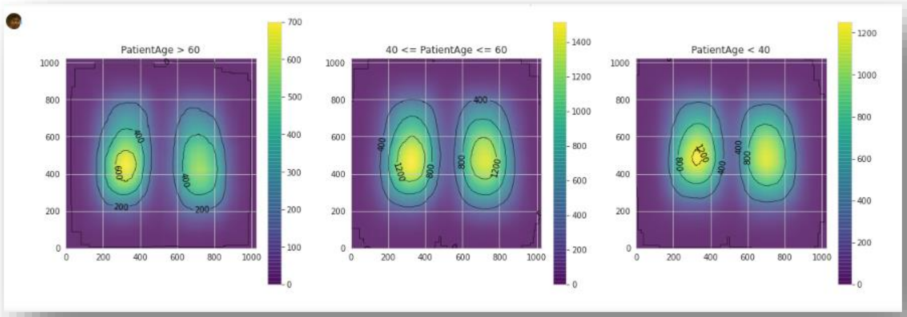
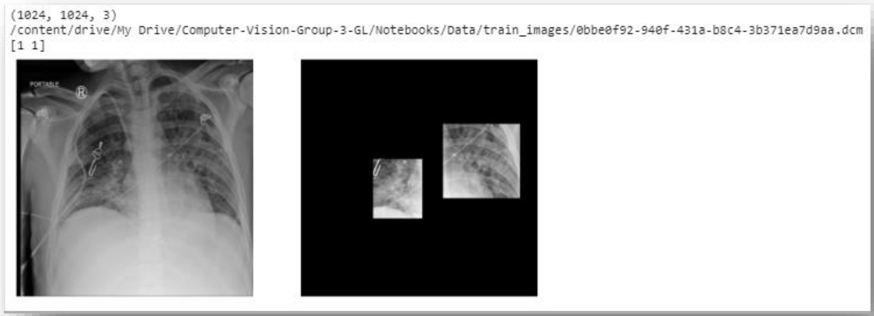


Figure 24: Contour plot for various age group.

Next to this we will view the images and the position of bounding boxes in the images that will help us determine if the opacity is visible in the lungs and therefore will help in determining Pneumonia in the patients. We will be able to find different images as shown below for the Pneumonia detection.



```
(1024, 1024, 3)
/content/drive/My Drive/Computer-Vision-Group-3-GL/Notebooks/Data/train_images/0bbe0f92-940f-431a-b8c4-3b371ea7d9aa.dcm
[1 1]
```

Figure 25: Sample image with masked area

SECTION 3 – Step by Step Walkthrough the solution

Detailed Walkthrough Base Model (Milestone1_pneumonia-detection.ipynb):

Now let’s dive into the Model in depth to find out how the mask RCNN is working to identify the opacity in lungs.

First we load the DICOM images and then extract the DICOM files in a list. Post that we will create a class that will be used for Configuration for training Lung Opacity location detection on the RSNA pneumonia dataset. Created a sub-class that inherits from config class and override properties that need to be changed.

**Here we will Name the configurations. Override in sub-classes give the NUMBER OF GPUs to use. (When using only a CPU, this needs to be set to 1.) We will then give number of images to train with on each GPU like a 12GB GPU can typically handle 2 images of 1024x1024px. Adjust based on GPU memory and image sizes. Use the highest. Number that GPU can handle for best performance. Train on 2 GPU and 8 images per GPU. We can put multiple images on each GPU because the images are small and Batch size is 8 (GPUs * images/GPU) number of training steps per epoch. This doesn't need to match the size of the training set. Tensor board updates are saved at the end of each epoch, so setting this to a smaller number means getting more frequent Tensor Board updates. Validation stats are also calculated at each epoch end and they might take a while, so don't set this too small to avoid spending a lot of time on validation stats.*

Backbone network architecture supported values are: resnet50, resnet101.We can also provide a callable that should have the signature of model.resnet_graph. We need to supply a callable to COMPUTE_BACKBONE_SHAPE as well. Here, we leave it as resnet 101. Length of square anchor side in pixels. Number of classification classes (including background) background and 1 pneumonia classes, Input image resizing. Generally, use the "square" resizing mode for training and predicting and it should work well in most cases. In this mode, images are scaled up such that the small side is = IMAGE_MIN_DIM, but ensuring that the scaling doesn't make the *long side > IMAGE_MAX_DIM. Then the image is padded with zeros to make it a square so multiple images can be put in one batch.*

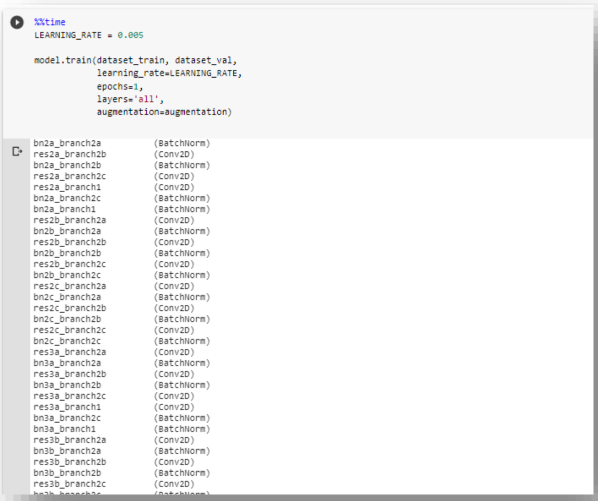
Available resizing modes

- *None: No resizing or padding. Return the image unchanged.*
- *Square: Resize and pad with zeros to get a square image of size [max_dim, max_dim].*
- *Pad64: Pads width and height with zeros to make them multiples of 64. If IMAGE_MIN_DIM or IMAGE_MIN_SCALE are not none, then it scales up before padding. IMAGE_MAX_DIM is ignored in this mode. The multiple of 64 is needed to ensure smooth scaling of feature maps up and down the 6 levels of the FPN pyramid (2**6=64).*
- *Crop: Picks random crops from the image. First, scales the image based on IMAGE_MIN_DIM and IMAGE_MIN_SCALE, then picks a random crop of size IMAGE_MIN_DIM x IMAGE_MIN_DIM. Can be used in training only. IMAGE_MAX_DIM is not used in this mode.*

Number of ROIs per image to feed to classifier/mask heads. The Mask RCNN paper uses 512 but often the RPN doesn't generate enough positive proposals to fill this and keep a positive: negative ratio of 1:3. You can increase the number of proposals by adjusting the RPN NMS threshold. Maximum number of ground truth instances to use in one image. Max number of final detection. Minimum probability value to accept a detected instance ROIs below this threshold are skipped. Non-maximum suppression threshold for detection.

Post this we are creating another Dataset class for training automated location detection for lung opacities on the RSNA pneumonia dataset. For overriding the Base dataset class in Matter port's Mask R-CNN. Then we extract Metadata of the DICOM file. Then we have divide the training, test and validation Dataset. Then we will prepare the training and validation dataset and show annotation of bounding boxes in the DICOM image. After this we will be doing some data Augmentation.

Now we are creating a model for image training is created using Mask R CNN and we will be loading weights in the model. Then define a learning rate 0.005 and then start model training for 2 epochs, to test model feasibility it is important to check if the model is converging at all and hence we are training the model for two epochs here.



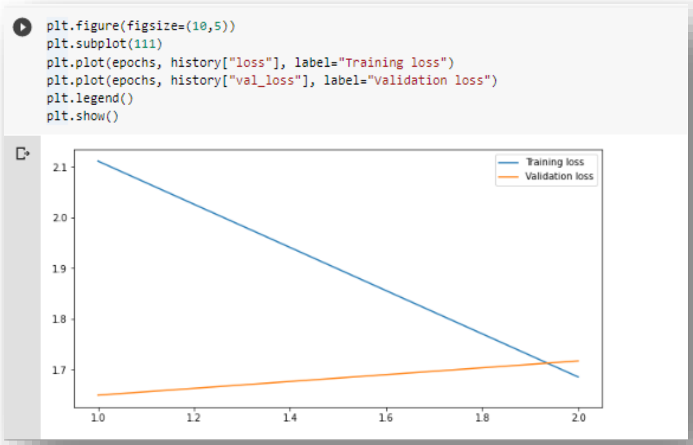
```
%%time
LEARNING_RATE = 0.005

model.train(dataset_train, dataset_val,
            learning_rate=LEARNING_RATE,
            epochs=1,
            layers='all',
            augmentation=augmentation)
```

```
bn2a_branch2a          (BatchNorm)
res2a_branch2b         (Conv2D)
bn2a_branch2b          (BatchNorm)
res2a_branch2c         (Conv2D)
res2a_branch1          (Conv2D)
bn2a_branch2c          (BatchNorm)
bn2a_branch1           (BatchNorm)
res2b_branch2a         (Conv2D)
bn2b_branch2a          (BatchNorm)
res2b_branch2b         (Conv2D)
bn2b_branch2b          (BatchNorm)
res2b_branch2c         (Conv2D)
bn2b_branch2c          (BatchNorm)
res2c_branch2a         (Conv2D)
bn2c_branch2a          (BatchNorm)
res2c_branch2b         (Conv2D)
bn2c_branch2b          (BatchNorm)
res2c_branch2c         (Conv2D)
bn2c_branch2c          (BatchNorm)
res3a_branch2a         (Conv2D)
bn3a_branch2a          (BatchNorm)
res3a_branch2b         (Conv2D)
bn3a_branch2b          (BatchNorm)
res3a_branch2c         (Conv2D)
res3a_branch1          (Conv2D)
bn3a_branch2c          (BatchNorm)
bn3a_branch1           (BatchNorm)
res3b_branch2a         (Conv2D)
bn3b_branch2a          (BatchNorm)
res3b_branch2b         (Conv2D)
bn3b_branch2b          (BatchNorm)
res3b_branch2c         (Conv2D)
```

Figure 26: Snip from the code for model training

After this we will tabulate out the different losses in various epochs and compare training vs. validation loss. Here in the below snip we can notice the comparison between training loss and validation were we can training loss is decreasing and validation is increasing. However, in reality we will try to reduce both training and validation loss in the test datasets. Hence, we will cover this further in the Final report.



```
plt.figure(figsize=(10,5))
plt.subplot(111)
plt.plot(epochs, history["loss"], label="Training loss")
plt.plot(epochs, history["val_loss"], label="Validation loss")
plt.legend()
plt.show()
```

Figure 27: Snip for the loss validation

Post this we will feed the validation data to the model and test them to identify lung opacity which resulted in the identification of opacity in lungs in the below as shown in the below snip.

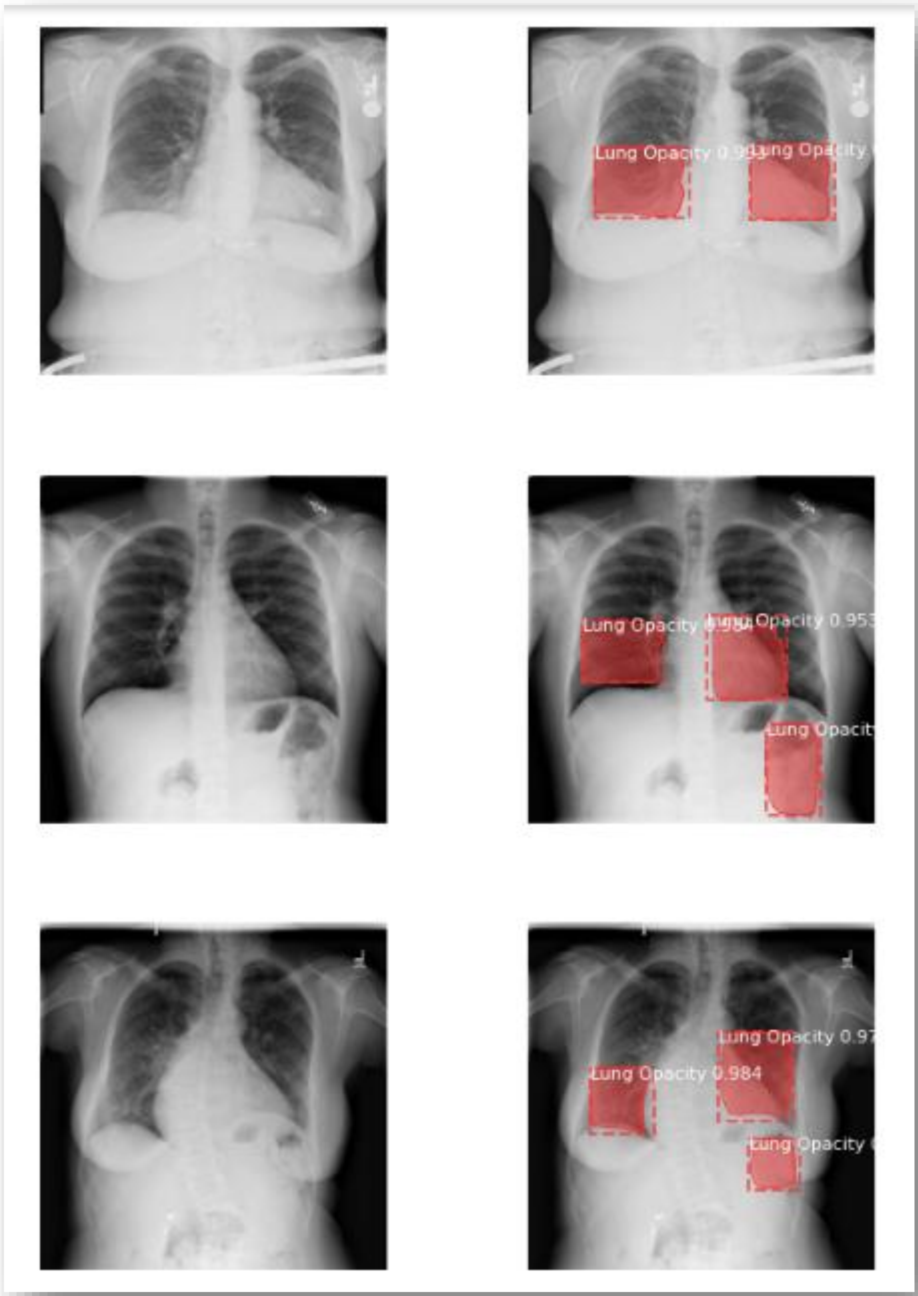


Figure 28: Identifying the opacity and masking the region

Model selection and Why using Unet architecture?

We planned using Yolo V3 where we ran the model for few epochs and obtained below results. This model results are for the loss comparison of total vs. the average loss. However we have limited resource in processing this model and therefore we would use another model as our final model for this project. We used Unet architecture rather than using Yolo V3 as we have attained better accuracy in the model with less loss percentage. Hence used the same in final model building.

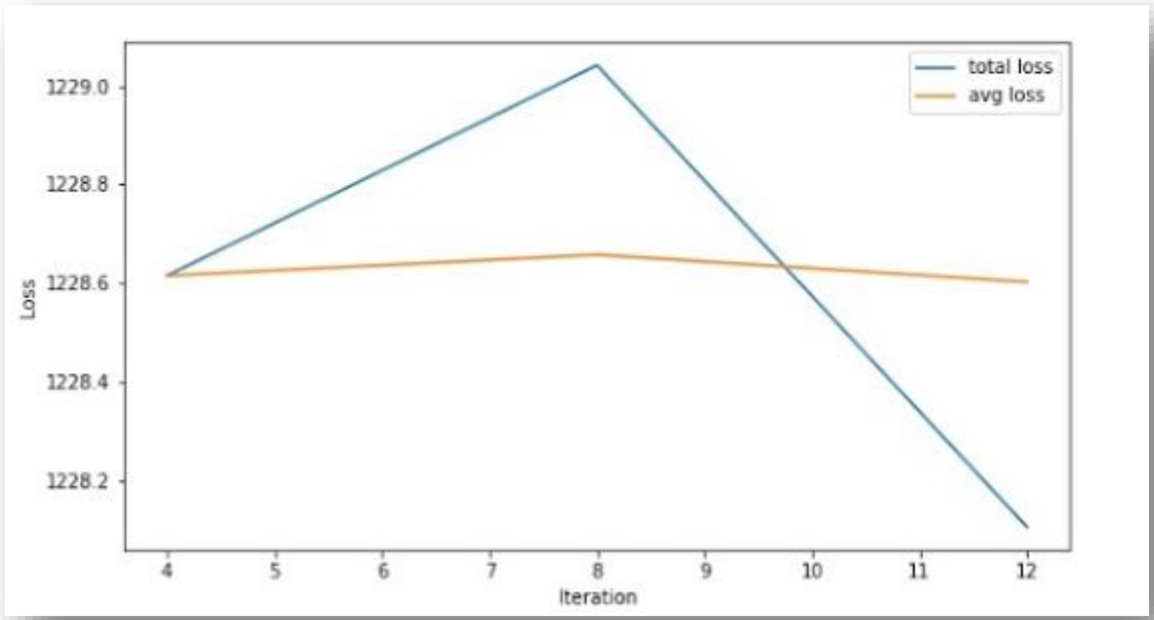


Figure 29: Yolo V3 Loss comparison

Detailed Walkthrough of Final Model (using Unet Architecture in the Final_pneumonia_detection.ipynb)

Final Model detailed walkthrough is as such, we have imported the necessary libraries in the notebook. Post importing the library. We will first build the dictionary and put all the files in it that contain bounding boxes, description and DICOM images in the dictionary.

Then we have created dictionary for the Pneumonia locations. This would be used to add label to the data of patients who have Pneumonia. Post this we will load the filenames and will separate test and train datasets.

Moving on to the next step for data generator part. We would need to create a data generator for the following reason.

The dataset is too large to fit into memory, so we need to create a generator that loads data on the fly.

- The generator takes in some filenames, batch size and other parameters.

- The generator outputs a random batch of numpy images and numpy masks.

Post the data generator part we have built a Unet architecture network. Where we have trained below listed number of parameter.

Total params: 3,149,217
Trainable params: 3,144,673
Non-trainable params: 4,544

Then we have fixed the data in the generator with 32 as batch size and 256 as image size. For both Train and validation dataset. Then we have fit the dataset in the model for training and run it for 10 Epochs. While running for 10 Epochs we found the best accuracy score to be 0.9749 and the mean IOU value to be 0.7222 as better than our outset result which was 0.7071 previously. Moving to the next level we have plotted the graph against validation loss and train loss, validation accuracy and train accuracy and validation iou and test iou.

Next to this we will predict in the batch of images. Here we will create an iteration were we will fetch images and masks from valid gen. Then we created figure for the images. Next we create an iteration were we will extract image, masks, prediction. Post this we will plot images and create threshold for the true and predicted masks where we will show the true masks with blue and predicted masks with blue just to show the visual difference. This part will be later covered in the visualization part. Also, then we perform the model tuning in 6 epochs using cosine annealing.

We then predict the test images in which we have loaded the filenames and created the test generator. Next to this we built submission dictionary just, loop through the test set. Next to this we will predict the batch of images and loop through the batch. Post that we will resize the mask and apply a bounding box for the images. Then we will add file name and prediction string to the dictionary.

SECTION 4 – Model Evaluation?

1. Train/Validation Loss, Accuracy and IOU graph

We have shown the train/validation Loss, Accuracy and IOU graph in the Section 6. Where we have collected the value of Loss, Accuracy and IOU in various and depicted the same in various epoch.

2. Confusion matrix (Final_pneumonia_detection.ipynb)

We have attempted to show confusion matrix for the model evaluation. However let's first understand what is confusion matrix?

Well, it is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve.

Let's understand TP, FP, FN and TN in terms of pregnancy analogy.

True Positive:

Interpretation: You predicted positive and it's true.

You predicted that a woman is pregnant and she actually is.

True Negative:

Interpretation: You predicted negative and it's true.

You predicted that a man is not pregnant and he actually is not.

False Positive: (Type 1 Error)

Interpretation: You predicted positive and it's false.

You predicted that a man is pregnant but he actually is not.

False Negative: (Type 2 Error)

Interpretation: You predicted negative and it's false.

You predicted that a woman is not pregnant but she actually is.

Just Remember, We describe predicted values as Positive and Negative and actual values as true and false.

How to Calculate Confusion Matrix for a 2-class classification problem? Let's understand confusion matrix through math.

Recall

Out of all the positive classes, how much we predicted correctly. It should be high as possible.

Precision

Out of all the positive classes we have predicted correctly, how many are actually positive.

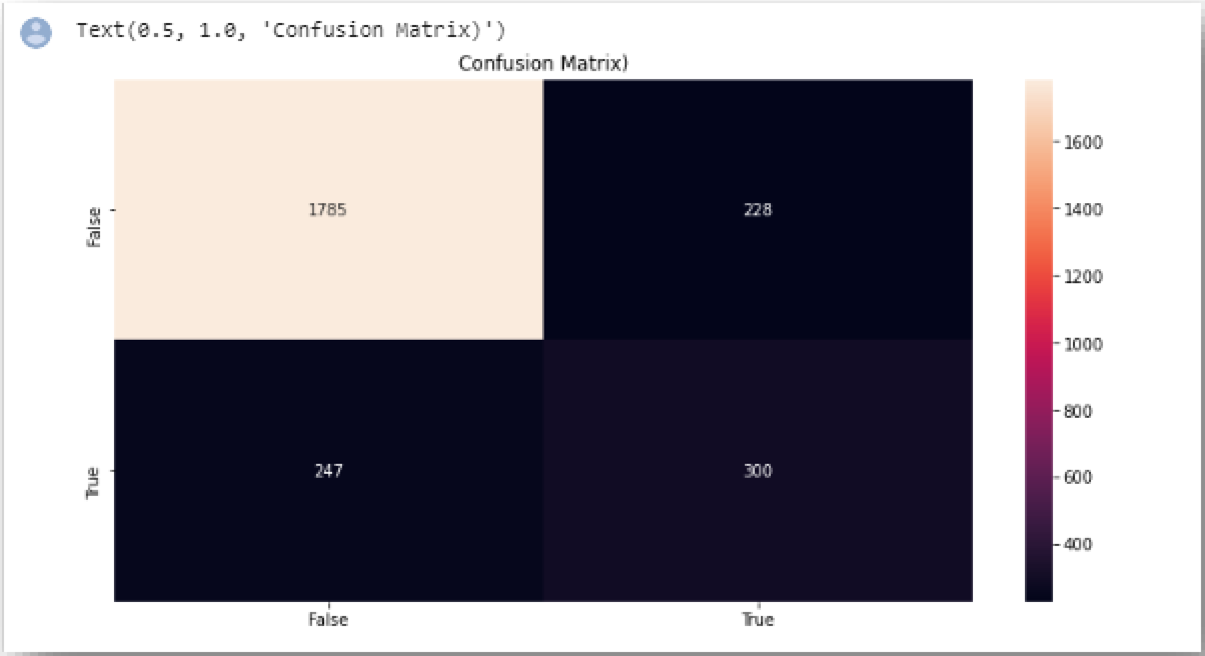
Accuracy

Out of all the classes, how much we predicted correctly, which will be, in this case 4/7. It should be high as possible.

F-measure

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

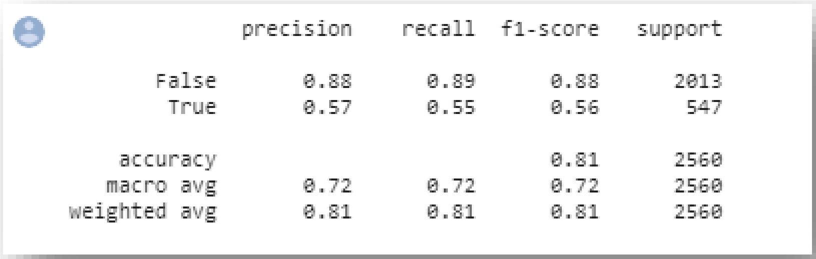
With respect to the project we have calculated the various parameter for the Pneumonia detection and accounted if the prediction is falling under which particular category. However in the medical terminology it is more important to concentrate more on precision therefore we have worked on the mean average precision in the base model and we have shown the same in the section below for mAP.



```
Text(0.5, 1.0, 'Confusion Matrix)')
```

Figure 30: Confusion Matrix

3. Classification Report:



```
              precision    recall  f1-score   support

       False       0.88      0.89      0.88      2013
        True       0.57      0.55      0.56       547

    accuracy                           0.81      2560
   macro avg       0.72      0.72      0.72      2560
weighted avg       0.81      0.81      0.81      2560
```

Figure 31: Classification Report

4. mAP (Mean Average Precision) – (Milestone1_pneumonia_detection)

We will use mAP mean average precision value of multiple epochs in mAP. Here we will first get the image from validation dataset and then we will predict the bounding boxes on top of it. This bounding will look into for the opacity in the lung region and will identify the percentage of region with opacity as shown in the below image.

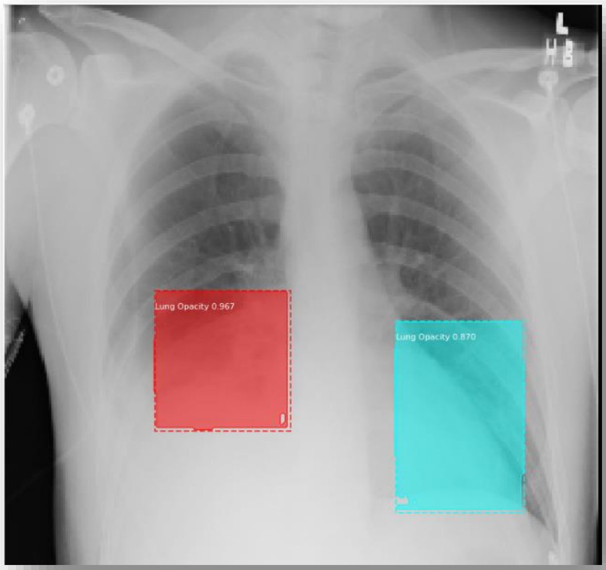


Figure 32: Visualizing the opacity in the validation images

From the utils library we will fetch average precision and the same is shown in the below graph. Where we see the precision/recall curve as below.

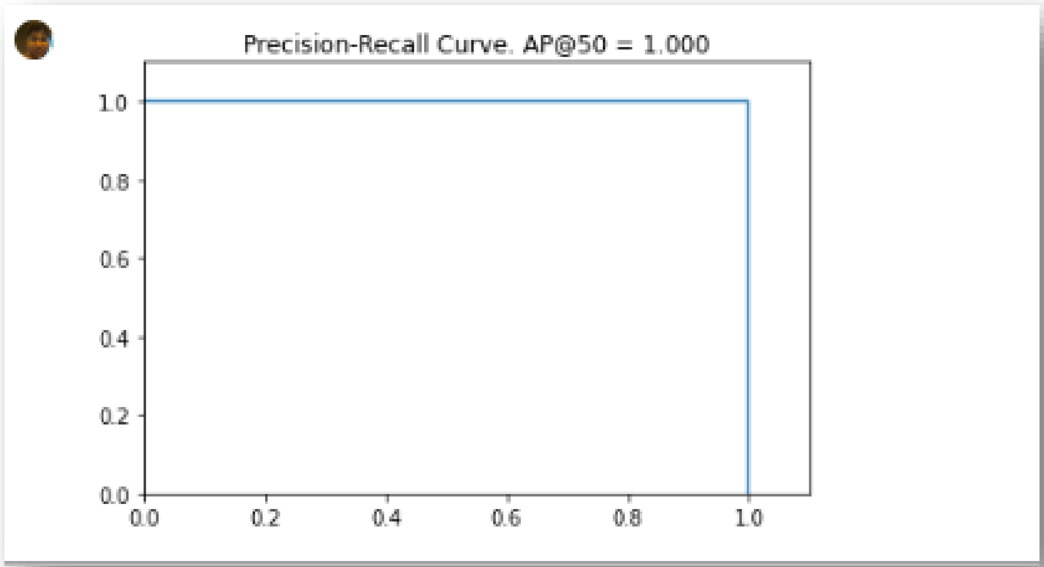


Figure 33: Precision vs. Recall

Now below we will visualize the grid of ground truth object and there prediction.

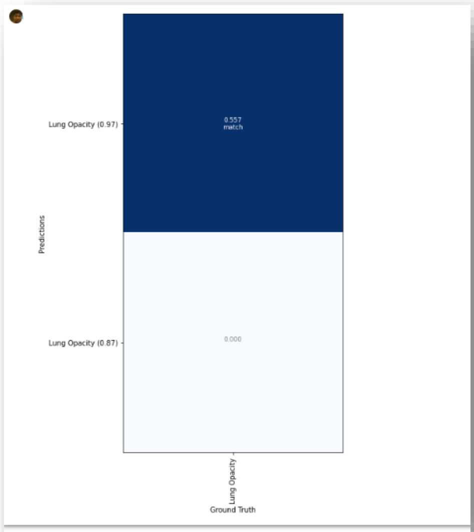
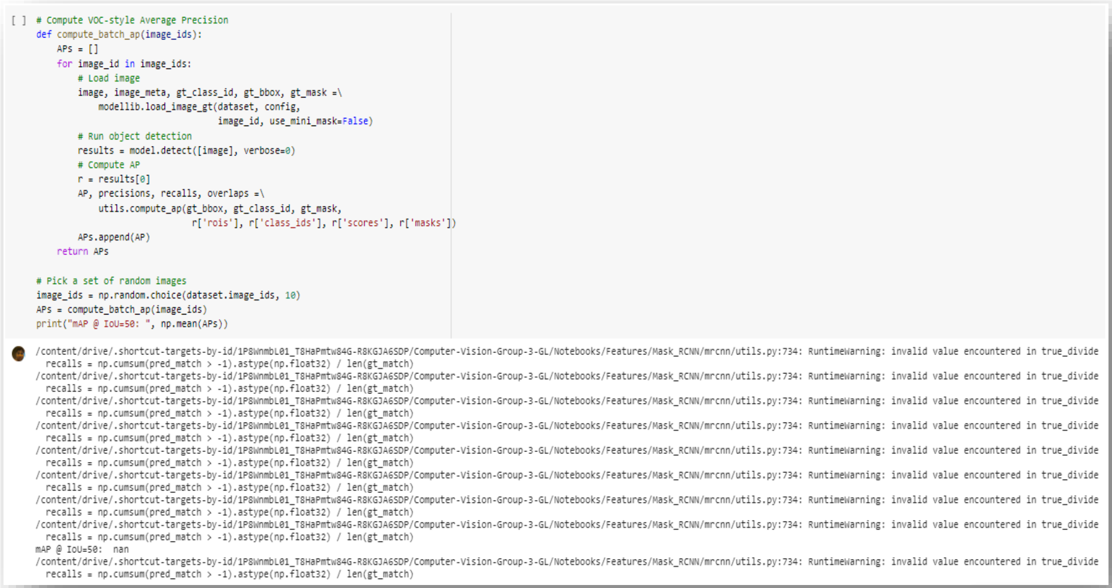


Figure 34: Visualize the grid of ground truth

In the below snip we will pick set of random images to find out the average precision. To find out the mean average precision, we have created a function to bring out the Average precision using utils.compute_ap and then checking the same in the random images.



```
[ ] # Compute VOC-style Average Precision
    def compute_batch_ap(image_ids):
        APs = []
        for image_id in image_ids:
            # Load image
            image, image_meta, gt_class_id, gt_bbox, gt_mask =\
                modellib.load_image_gt(dataset, config,
                                       image_id, use_mini_mask=False)
            # Run object detection
            results = model.detect([image], verbose=0)
            # Compute AP
            r = results[0]
            AP, precisions, recalls, overlaps =\
                utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
                                 r['rois'], r['class_ids'], r['scores'], r['masks'])
            APs.append(AP)
        return APs

    # Pick a set of random images
    image_ids = np.random.choice(dataset.image_ids, 10)
    APs = compute_batch_ap(image_ids)
    print("mAP @ IoU=50: ", np.mean(APs))
```

```
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
mAP @ IoU=50:  nan
/content/drive/.shortcut-targets-by-id/1P8WnmbL01_T8HaPmtw84G-R8KGJA6SDP/Computer-Vision-Group-3-GL/Notebooks/Features/Mask_RCNN/mrcnn/utils.py:734: RuntimeWarning: invalid value encountered in true_divide
  recalls = np.cumsum(pred_match > -1).astype(np.float32) / len(gt_match)
```

`

Figure 35: Snip of the code

Why do we need to use Precision?

In medical terminology a test method is said to be precise when repeated analyses on the same sample give similar results. When a test method is precise, the amount of random variation is small. The test method can be trusted because results are reliably reproduced time after time.

Although a test that is 100% accurate and 100% precise is the ideal, in reality, this is impossible. Tests, instruments, and laboratory personnel each introduce a small amount of variability. This amount of variability does not usually detract from the test's value as it is taken into account.

Why do we need to use mAP?

Precision and recall are commonly used along with other metrics such as accuracy, F1-score, specificity, a.k.a. true negative rate (TNR), receiver operating characteristics (ROC), lift and gain.

However, all these metrics fail when it comes to determining if a model is performing well in information retrieval or object detection tasks. This where mAP comes to the rescue!

5. Test image prediction

We have performed test image prediction as indicated in the below section in the data visualization. In the test image prediction we have compared the bounding boxes of ground truth and the prediction bounding boxes in the red and blue respectively. This would result in the better visualization of result and the bounding boxes as indicated in Section 6 Visualization.

SECTION 5 – Comparison to the Benchmark.

In the benchmark model we did not get better accuracy but in the Final model where we have used the UNET architecture we have identified accuracy rate of 0.97 on train images and 0.97 approximate in validation image as maximum in mere 6 epochs this indicate that we have attained good accuracy in the model. This can prove that our model is not over fitting in the train images.

In addition to this, we have improved the score from benchmark. One of the major reason is we have not used transfer learning here. Therefore it is quiet efficient to say here that since this final model is trained initially with the Chest X-ray training data and the then validation and testing is observed. Which is the reason for reaping better results for train and test and validation data.

SECTION 6 - Visualizations

After exploring Mask R-CNN, Darknet and U-net. We found out U-net is the best model for us considering limitation of computing power with us. With 3,144,673 Trainable parameters we succeeded in reducing Loss, increasing Accuracy and IOU. With our initial training with 25 epochs we got the results shown in below graph.

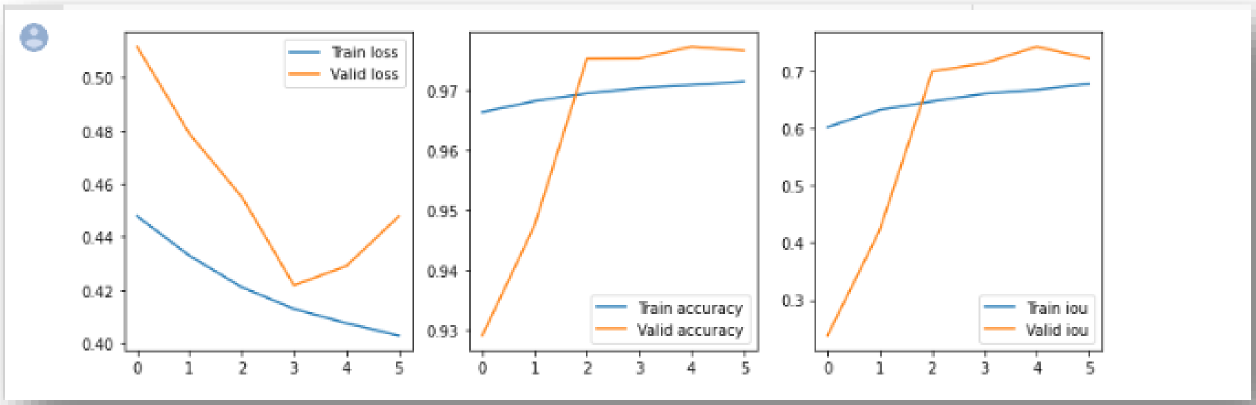


Figure 36: Train Validation comparison loss, accuracy and IOU

We performed bounding box prediction for some of the validation images which resulted in good results.

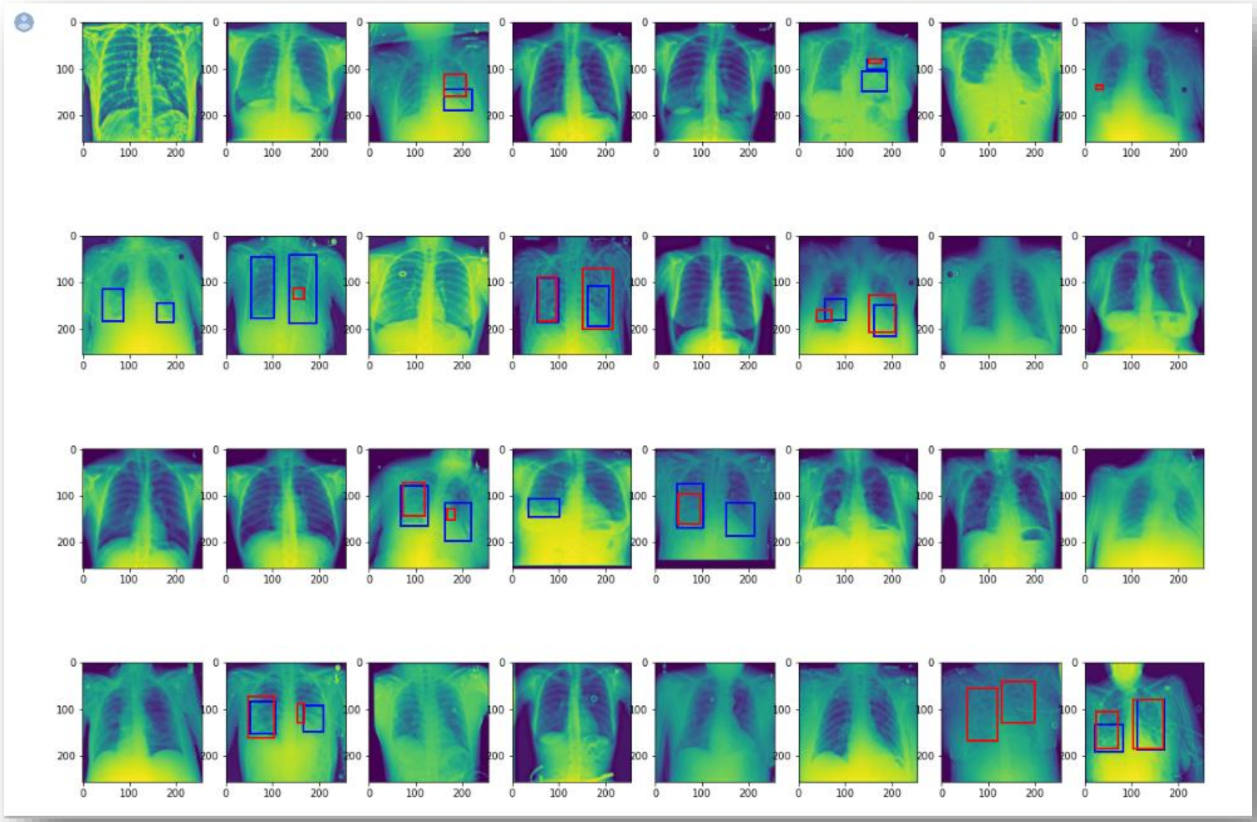


Figure 37: Display the test images

SECTION 7 - Implications

Detecting Pneumonia is difficult task because lung inflammation doesn’t always results in to presence of Pneumonia, there could be few other disease like Tuberculosis, Lung cancer etc. Even doctors with their naked eye can’t guarantee whether it’s Pneumonia or not. This can be used as additional tool when doctors are in dilemma on presence of Pneumonia. Where the doctors can ultimately shift their focus towards bounding boxes instead of looking for complete X-Ray Image. The careful examination of bounding boxes would result into better clarity and prediction of Pneumonia.

However, this also indicates that this field has more scope of research which were out of scope of this project. Where this the images can be used in combination with DICOM data and other datasets which could be available at provider level to decide whether a patient is suffering from Pneumonia. As such the medical history for such situation is boon and would result and clarification on indicating whether if patient is suffering from Pneumonia. And as mentioned the medical history of a patient can be gained from provider (Insurance provider data). Which along with the presence of other comorbidities will be a good model together to present with.

SECTION 8 - Limitation

This model has been trained on images with known diseases. Whenever new kind of opacity comes into picture which this model haven't seen earlier will fall short. To overcome this we will continue to retrained the model whenever there is new lung opacity found in humans and compare the results.

SECTION 9 - Closing Reflections

We learned that we can't say one model which works better for other will always work for same or different kind of problem. It is totally dependent on the data available for training and computing power required for it. Selecting the model for problem depends on following aspects.

1. Training data available
2. Computational power
3. Type of solution required for problem (as assistance / complete solution)

We will always have best options short listed before actually starting working on solution so that whenever required we should have option in hand to switch from one type of model and don't have to redo research activity.