

A Project Report  
on  
***Pneumonia Detection System***

**Submitted to:**



Post Graduate Program in Artificial Intelligence and Machine  
Learning (2019-2020)

**Submitted By:**

Shraddha Belpande

Akash Jaiswal

Himanshu Birla

Sripriya K S

Niranjan Suresh

**Mentor:**

Sravan Malla

# Abstract

This project's goal is to build a detection system, to determine whether a patient is suffering from pneumonia or not by looking at chest X-ray images with the aim of achieving high accuracy. In this project, we are using the CXR image dataset from Kaggle and are implementing deep-learning based approach to detect lung opacity. The network is trained on the challenging publicly available dataset on Kaggle pneumonia Detection on which an object detection is performed.

<b>Contents</b>	<b>Page no.</b>
1. Introduction.....	5
1.1 What is object detection in Computer Vision?.....	5
1.2 Problem Statement: .....	5
1.2.1 What is Pneumonia? .....	5
1.2.2 Pneumonia Detection .....	5
1.2.3 Application or Business Domain Value.....	5
2. Dataset .....	6
2.1 Data .....	6
2.2 Metadata .....	7
3. EDA (Exploratory Data Analysis).....	8
3.1 Target vs. Record Count.....	8
3.2 Class vs. Record Count .....	8
3.3 Patients vs. Boxes.....	9
3.4 Patient vs. Boxes and Target .....	9
3.5 Patient vs. Target and Class .....	10
3.6 Distribution of patients by age .....	10
3.6.1 Before Removing Outliers .....	10
3.6.2 After Removing Outliers.....	11
3.7 Distribution of patients by sex .....	11
3.8 Distribution by View Position .....	11
3.9 Display some sample images .....	13
3.10 Bounding Box Distribution .....	13
3.10.1 Understanding x and y coordinates using scatterplot and density estimates .....	13
3.10.2 Understanding range of x and y coordinates .....	14
3.10.3 Understanding x and y coordinates based on gender .....	14
3.10.4 Understanding x and y coordinates for females .....	15
3.10.5 Understanding x and y coordinates for males .....	15
3.10.6 Understanding height and width of bounding box coordinates.....	16
3.10.7 Understanding height and width of bounding box coordinates (males and females)..	16
3.11 Understanding Pneumonia Location .....	17
3.11.1 Understanding pneumonia location for the overall dataset .....	17
3.11.2 Understanding pneumonia location for males and females.....	17
3.12 Focus Age Group between 40 to 65 .....	18
3.12.1 Based on gender.....	18
3.12.2 Based on x and y coordinates .....	18
3.12.3 Based on x and y coordinates for females and males .....	19
3.12.4 Based on height and width of bounding box coordinates.....	19
4. Approach.....	20

5.	Models .....	20
5.1	Model Architecture .....	21
5.1.1	Model 1 .....	21
5.1.2	Model 2 .....	22
5.1.3	Model 3 .....	22
5.1.4	Model 4 .....	23
5.1.5	Model 5 .....	24
5.1.6	Model 6 .....	24
5.1.7	Model 7 .....	25
5.2	Model Summary .....	26
5.2.1	Model 1 .....	26
5.2.2	Model 2 .....	26
5.2.3	Model 3 .....	26
5.2.4	Model 4 .....	27
5.2.5	Model 5 .....	28
5.2.6	Model 6 .....	28
5.2.7	Model 7 .....	29
5.3	Implementation Process .....	30
5.4	Model Comparison .....	31
5.5	Model Performance .....	32
5.5.1	Loss Curves .....	32
5.5.2	IOU curves .....	33
5.6	Model Evaluation .....	34
6.	Conclusion .....	35
6.1	Work Summary .....	35
6.2	Implications .....	35
6.3	Limitations .....	35
6.4	Closing Reflections .....	36
6.5	Future Enhancements .....	36
7.	References .....	36

# 1. Introduction

## 1.1 What is object detection in Computer Vision?

Object detection is a computer technology related to computer vision, image processing, neural network that deals with detecting instances of semantic objects of a certain class (such as human, cars etc) in digital image or video. Computer vision is an integrated scientific field that deals with how computers can gain high-level understanding from digital images or videos. It can understand and automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analysing and understanding digital images. The object detection is quite complicated problem because it has to localize the image and predict the class of location of that image to find the bounding box around the given object and class.

## 1.2 Problem Statement:

The goal of this project is to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, your algorithm needs to automatically locate lung opacities on chest radiographs.

### 1.2.1 What is Pneumonia?

Pneumonia is an infection in one or both lungs caused by bacteria, viruses, and fungi. The infection causes inflammation in the air sacs in your lungs, which are called alveoli. In pneumonia air sacs fill with pus and may become solid.

The infection can be life-threatening to anyone, but particularly to infants, children and people over 65 aged people. It requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital signs and laboratory exams. Pneumonia usually manifests as an area or areas of increased opacity on CXR. However, the diagnosis of pneumonia on CXR is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, or post-radiation or surgical changes.

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

### 1.2.2 Pneumonia Detection

Now to detect Pneumonia we need to detect Inflammation of the lungs. The goal of this project is to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, the algorithm needs to automatically locate lung opacities on chest radiographs.

### 1.2.3 Application or Business Domain Value

The application of this project in medical field is as below:

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

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

## 2. Dataset

### 2.1 Data

The dataset on which the analysis is done is from the RSNA Pneumonia detection challenge in Kaggle. The original datasets consist of six different files as follows:

1. stage\_2\_test\_images (3000 files)
2. stage\_2\_train\_images (26.7k files)
3. GCP Credits Request Link - RSNA.txt (55 B)
4. stage\_2\_detailed\_class\_info.csv (1.57 MB)
5. stage\_2\_sample\_submission.csv (155.3 KB)
6. stage\_2\_train\_labels.csv (1.42 MB)

Out of these six datasets given we have considered only three of the datasets available to train the model.

The included datasets are:

1. stage\_2\_train\_labels.csv (1.42 MB): CSV file containing training set patient Ids and labels (including bounding boxes)
2. stage\_2\_train\_images (26.7k files): directory containing training set raw image (DICOM) files
3. stage\_2\_detailed\_class\_info.csv (1.57 MB): CSV file containing detailed labels

#### Analysis of stage\_2\_detailed\_class\_info.csv

	patientId	class
count	30227	30227
unique	26684	3
top	0d5bc737-03de-4bb8-98a1-45b7180c3e0f	No Lung Opacity / Not Normal
freq	4	11821

- There are potentially 3543 repetitive X-Rays for some patients because the patients must have been asked to get his or her x-ray after some interval to see the progress or how worse the situation is

#### Analysis of stage\_2\_train\_labels.csv

	x	y	width	height	Target
count	9555.000000	9555.000000	9555.000000	9555.000000	30227.000000
mean	394.047724	366.839560	218.471376	329.269702	0.316108
std	204.574172	148.940488	59.289475	157.750755	0.464963
min	2.000000	2.000000	40.000000	45.000000	0.000000
25%	207.000000	249.000000	177.000000	203.000000	0.000000
50%	324.000000	365.000000	217.000000	298.000000	0.000000
75%	594.000000	478.500000	259.000000	438.000000	1.000000
max	835.000000	881.000000	528.000000	942.000000	1.000000

- It seems most of the bounding boxes for the X-Rays are not square, probably because of high lung opacity
- This tells us that there are NaN present in the bounding box features, total missing values are 20672.

## 2.2 Metadata

Each image file is .dcm format, which is a DICOM (Digital Imaging and Communications in Medicine) used in the medical field. DICOM file is different from other image files because this file consists some important information about the image. It consists header and image datasets into a single file. By extracting data from these .dcm files, one can access important information about the patients, study parameters, etc. In the interest of patient confidentiality, all information that can be used to identify the patient should be removed. Here in this project we are extracting data from DICOM file which is saved as metadata, The Pydicom package is used to combine the image meta data information to form a detailed data frame. The metadata fields considered for analysis in the project are:

1. Patient Age
2. Patient Sex
3. Body part Examined
4. View Position

	PatientAge	BodyPartExamined	ViewPosition	PatientSex	path	patientId
0	51	CHEST	PA	F	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...	0004cfab-14fd-4e49-80ba-63a80b6bddd6
1	48	CHEST	PA	F	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...	00313ee0-9eaa-42f4-b0ab-c148ed3241cd
2	19	CHEST	AP	M	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...	00322d4d-1c29-4943-afc9-b6754be640eb
3	28	CHEST	PA	M	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...	003d8fa0-6bf1-40ed-b54c-ac657f8495c5
4	32	CHEST	AP	F	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...	00436515-870c-4b36-a041-de91049b9ab4

This patient X-ray information along with the patient metadata shows insights into various perspective for deeper analysis.

### 3. EDA (Exploratory Data Analysis)

The project is implemented in python 3. Tensorflow and Keras was used for training the deep network and OpenCV was used for image pre-processing.

#### 3.1 Target vs. Record Count

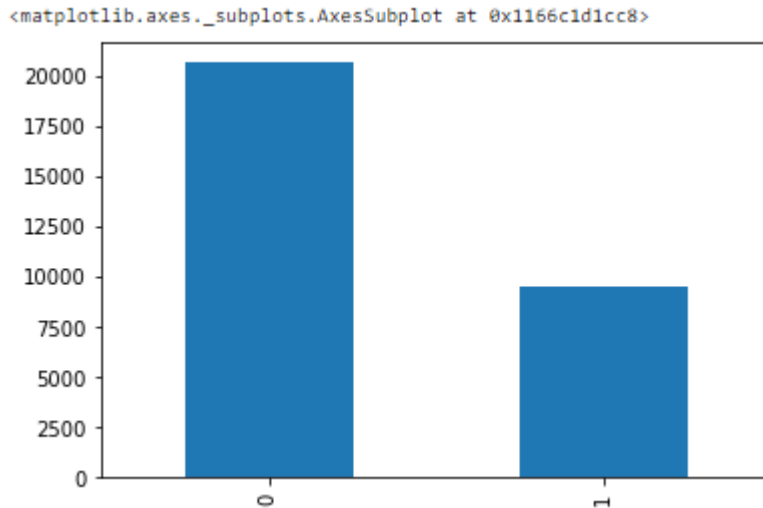


Fig No. 3.1

Fig no. 3.1 shows that out of 30227 , 20672 do not have pneumonia , hence this is the reason behind missing values in bounding box features.

This also shows the number of records of the class 0 are higher than the number of records of class 1. There is an imbalance in the classes, **we need to treat these imbalances**

#### 3.2 Class vs. Record Count

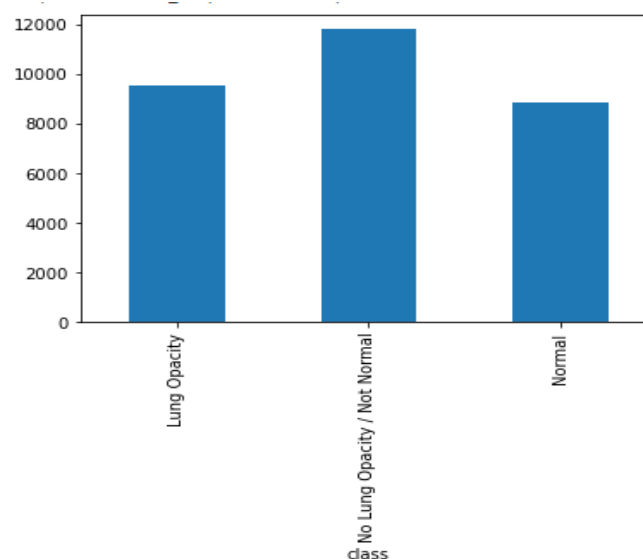


Fig no. 3.2



The graph in Fig no. 3.2 delineates that classes are highly imbalanced. We have target column which is the part of our interest in the pneumonia detection. The class column we can there is three classes which is Lung Opacity, No Lung Opacity and Normal. In target column we can see they have ignored the third class not normal but no lung opacity and considered as 0.

We have incorporated a few changes in the data structure for the convenience of usage as follows:

1. **Concatenated** the two above mentioned data frames with respect to the Patient IDs using inner join
2. **Replaced** the missing values of x, y, h and w columns with zeroes as the missing values were denoting the class of no lung opacity and hence do not have of the bounding boxes.
3. **Added two new columns:**
  - a. **boxes** which indicates the count of the number of records for each patient id.
  - b. **path** which indicates the path of the .dcm image from the train images folder

	patientId	x	y	width	height	Target	class	boxes	Path
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	0.0	0.0	0.0	0.0	0	No Lung Opacity / Not Normal	1	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...
28989	000924cf-0f8d-42bd-9158-1af53881a557	0.0	0.0	0.0	0.0	0	Normal	1	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...
28990	000db696-cf54-4385-b10b-6b16fbb3f985	316.0	318.0	170.0	478.0	1	Lung Opacity	2	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...
28991	000db696-cf54-4385-b10b-6b16fbb3f985	660.0	375.0	146.0	402.0	1	Lung Opacity	2	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...
28992	000fe35a-2649-43d4-b027-e67796d412e0	570.0	282.0	269.0	409.0	1	Lung Opacity	2	C:\Users\Skillup 08\Desktop\Capstone_Greatlear...

### 3.3 Patients vs. Boxes

	boxes	patients
0	1	23286
1	2	3266
2	3	119
3	4	13

Table No. 3.3.1

In Table no. 3.3.1 we can see majority of patients have one bounding box. This 23286 number does not match to number of patients having pneumonia.

### 3.4 Patient vs. Boxes and Target

	boxes	Target	patients
0	1	0	20672
1	1	1	2614
2	2	1	3266
3	3	1	119
4	4	1	13

In Table no. 3.4.1 we could see that out of 23286 patients with one bounding box, only 2614 patients have pneumonia and 20672 patients who do not have pneumonia have their bounding box coordinates as zero.

Table no. 3.4.1

### 3.5 Patient vs. Target and Class

	class	Target	Patient Count
0	Lung Opacity	1	9555
1	No Lung Opacity / Not Normal	0	11821
2	Normal	0	8851

The table shows that there are 9555 records in the dataset with pneumonia.

Table no. 3.5.1

### 3.6 Distribution of patients by age

#### 3.6.1 Before Removing Outliers

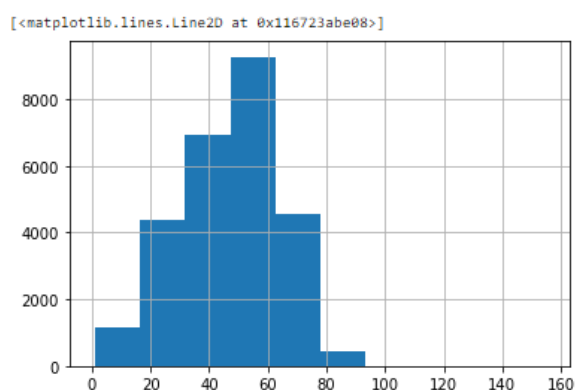


Fig no. 3.6.1

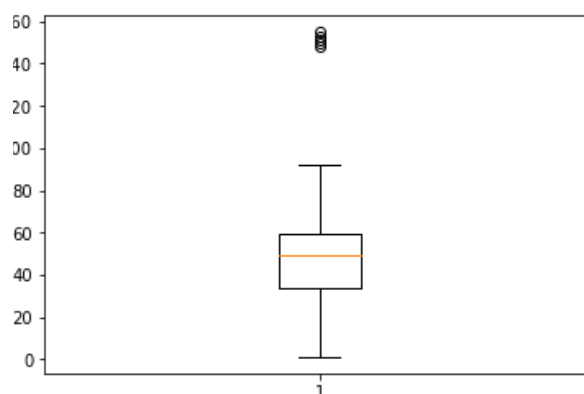


Fig no. 3.6.2

In above Fig no. 3.6.1 we can see that patients in majority we of age between 35-65 and Fig no 3.6.2 shows the patients age distribution in Box plot which shows there is outliers are present. Now we will see the distribution of patient's age with respect to their gender below:

#### Age Distribution for males

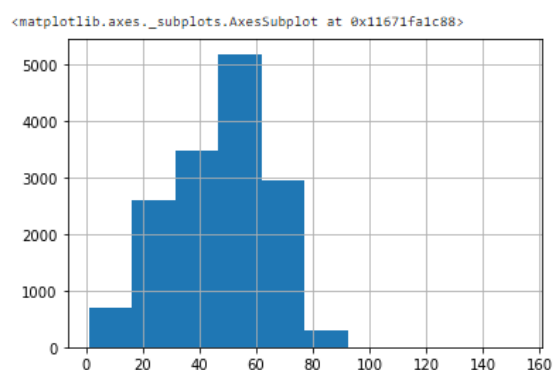


Fig no. 3.6.3

#### Age Distribution for females

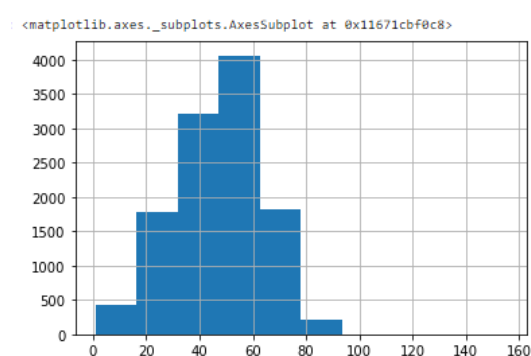


Fig no. 3.6.4

- When we split it on basis of Gender (Fig no. 3.6.3 and Fig no. 3.6.4) we found out that for females majority lies between 35 - 65, whereas for males its between 45-65
- We can also see that there is an empty gap after age of 99, there seems to be potential outliers.

### 3.6.2 After Removing Outliers

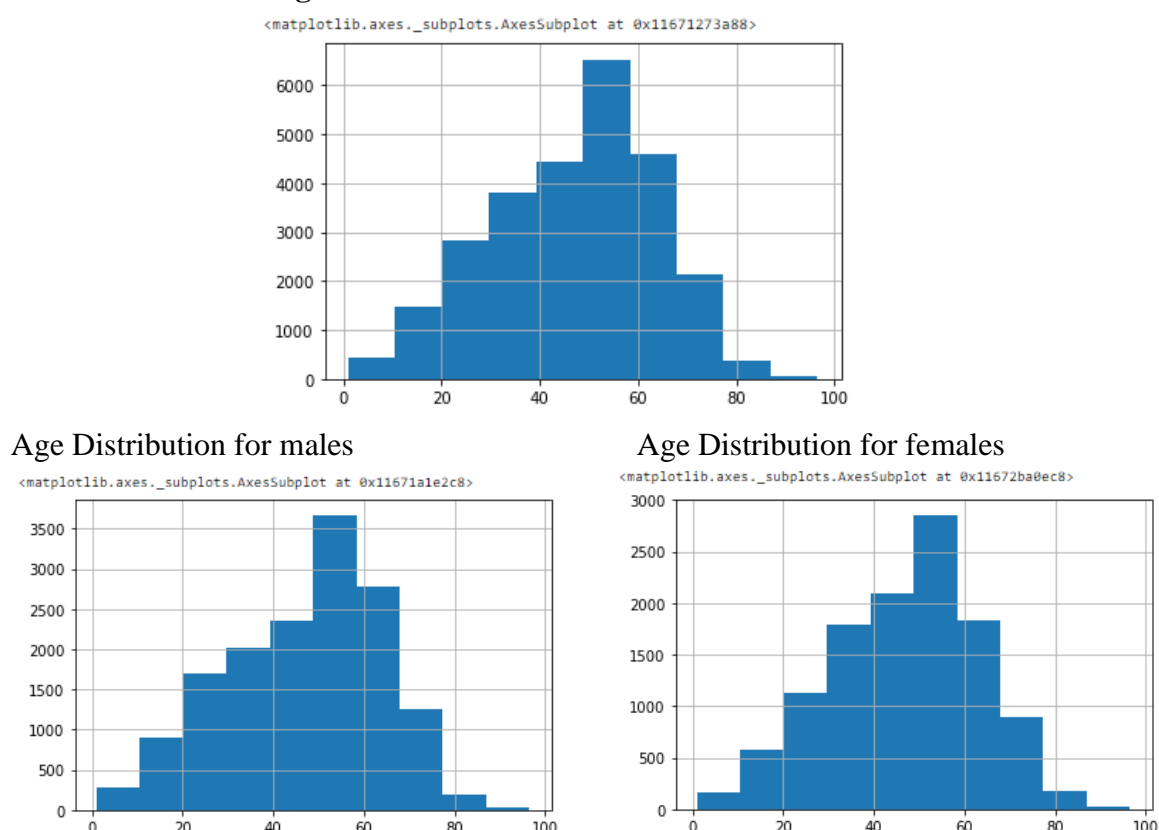


Fig no. 3.6.5

Above Fig no. 3.6.5 shows age data distribution after removing outliers

### 3.7 Distribution of patients by sex

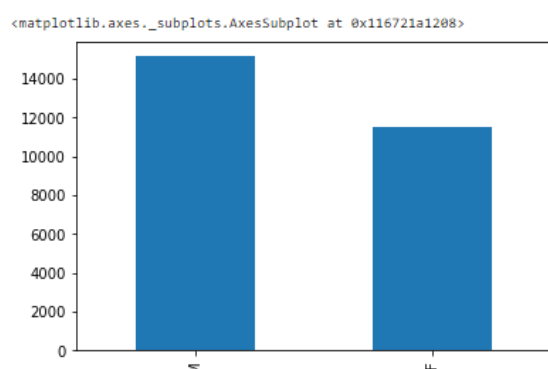


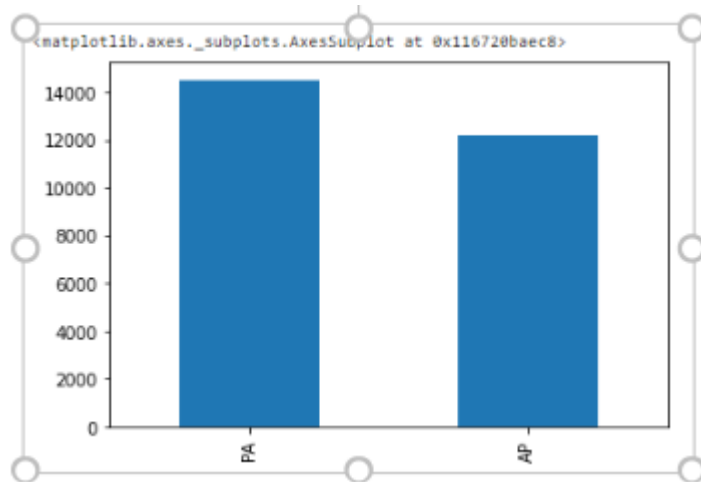
Fig. No. 3.7.1

Fig no. 3.7.1 shows the distribution of patient's by sex here we can see that Males are in majority whose X-rays are being conducted, we can say that approximately every 3rd person going for XRAY is Female

### 3.8 Distribution by View Position

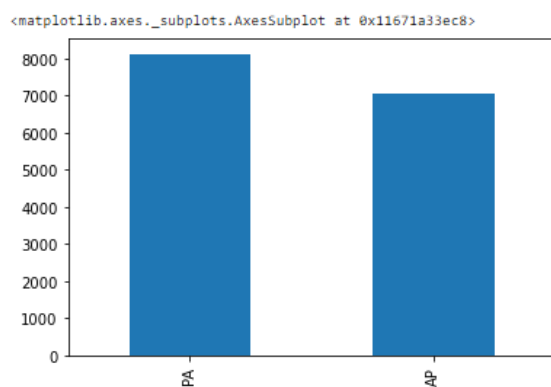
The use of body positioning requires an under-standing of terminology that refers to the relationship of the body to the x-ray equipment and to anatomical references. A projection refers to the path the x rays take through the body, from entrance to exit. Position describes the body and its relationship to the x-ray film device. For example, the physician orders an upright chest x ray with two views of an ambulatory patient. Here we are having two type of

view positions AP(Anteroposterior), which shows from front to back view and PA (Posteroanterior) which shows from back to front view.



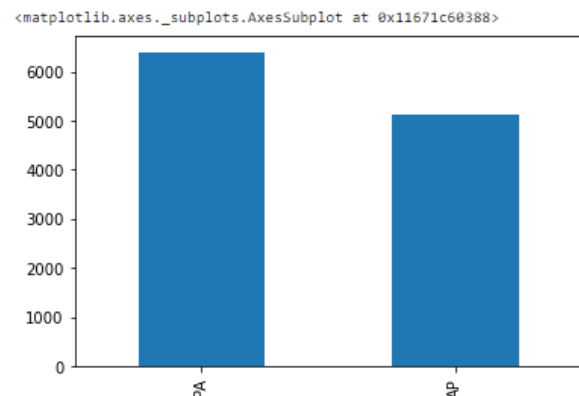
**Fig no. 3.8.1**

View Position for males



**Fig no. 3.8.2**

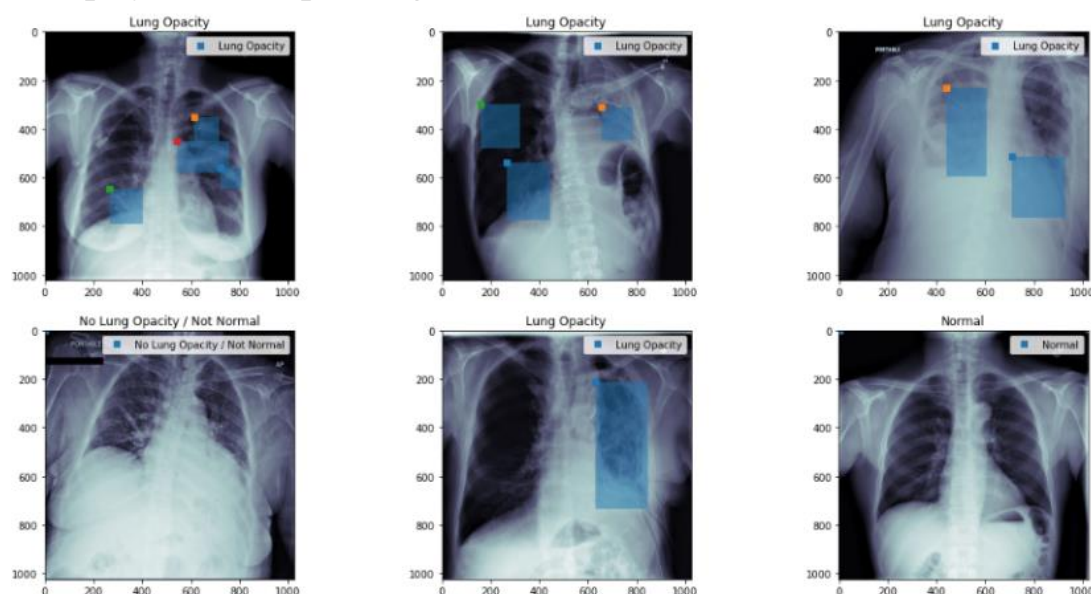
View Position for females



**Fig no. 3.8.3**

In the distribution of view position in fig no. 3.8.1 We can see that there is not much difference between posteroanterior & anteroposterior View Position of X-Rays. If we distribute by gender in Fig no. 3.8.2 and Fig no. 3.8.3 we can see there is no difference as such on based on Gender.

### 3.9 Display some sample images

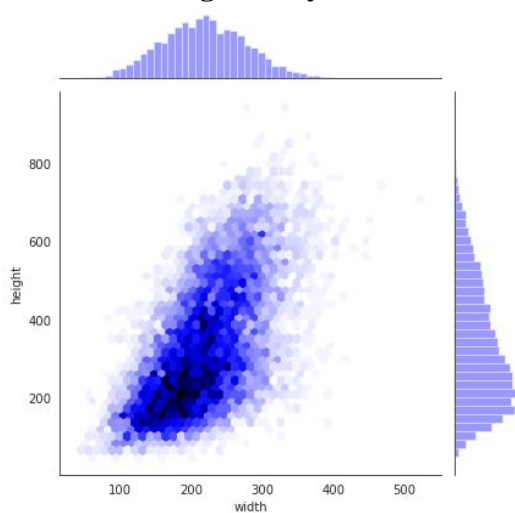


**Fig no. 3.9.1**

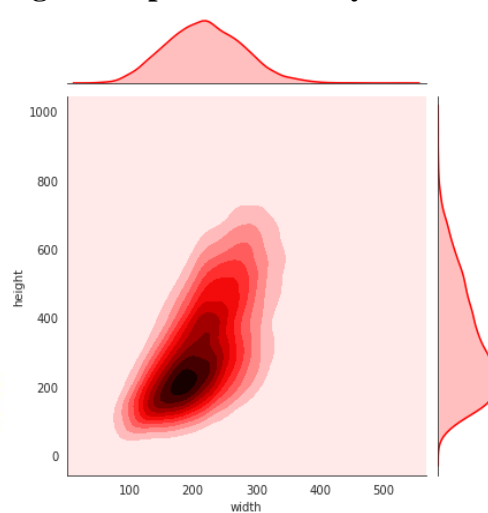
Fig no 3.9.1 shows some X-Ray images from the given data, some of them having lung opacity, No lung opacity/Not normal and normal lung

### 3.10 Bounding Box Distribution

#### 3.10.1 Understanding x and y coordinates using scatterplot and density estimates



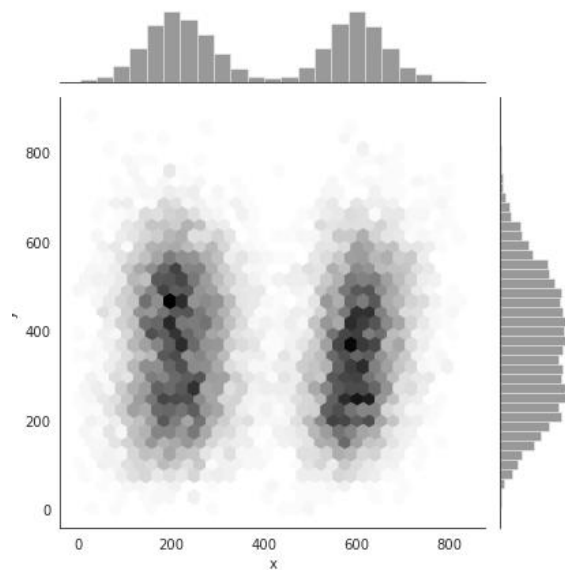
**Fig no 3.10.1**



**Fig no 3.10.2**

From the plotted graph (Fig no 3.10.2), it is evident that the co-ordinates of the bounding box are most likely having their central measure of (200,400) for the left lung and of (600,400) for the right lung.

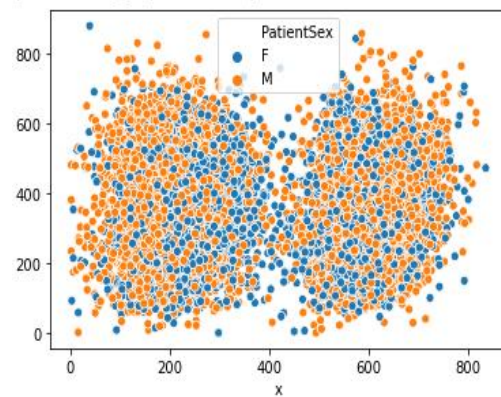
### 3.10.2 Understanding range of x and y coordinates



**Fig No 3.10.3**

In Fig no. 3.10.3 we can observe that on left lung most of the bounding box lies in between range of (400,200) to (550,200) and On right lung most of the bounding box lies in between range of (200,600) to (400,600)

### 3.10.3 Understanding x and y coordinates based on gender



**Fig no 3.10.4**

In fig no 3.10.4 we can observe the below findings:

1. For Females, in their left lung, opacity is being found in mostly right portion
2. For Females, in their right lung, opacity is being found in mostly left portion.

### 3.10.4 Understanding x and y coordinates for females

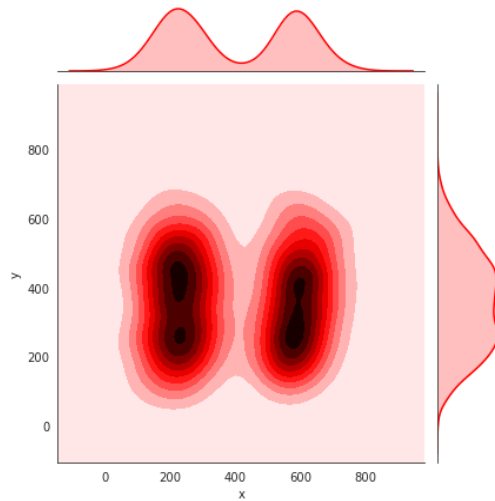


Fig no 3.10.5

Fig 3.10.5 shows the below findings in data:

For Females we can see higher density lies in:

1. **For Left Lung:** Range of Y-coordinate (380-550) & (220-300) & X-coordinate is 200.
2. **For Right Lung:** Range of Y-coordinate (200-550) & X-coordinate is 600, though there is slight decrease in density in between Y-Coordinate (380-390)

### 3.10.5 Understanding x and y coordinates for males

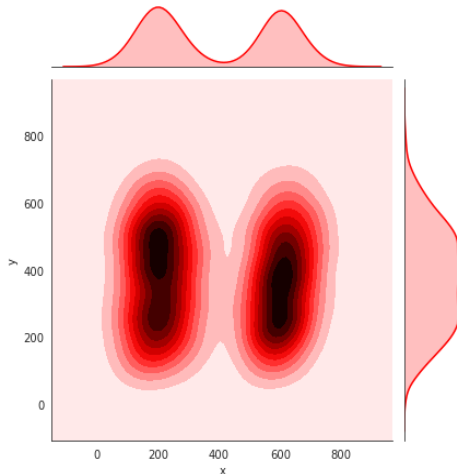
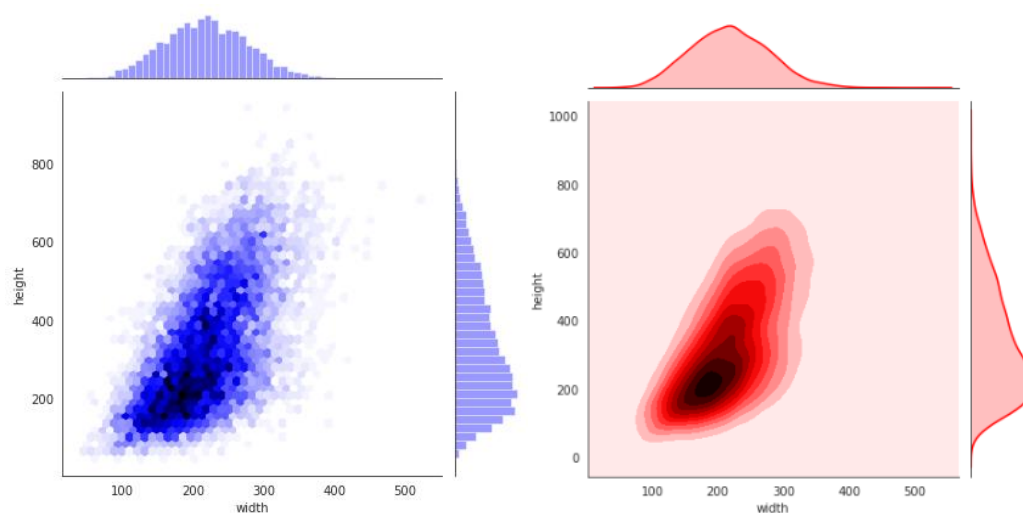


Fig no. 3.10.6

For Males we can see (Fig no. 3.10.6) higher density lies in:

1. **For Left Lung:** Range of between (380,200) & (580,200).
2. **For Right Lung:** Range of Y-coordinate (200,600) (450,600)

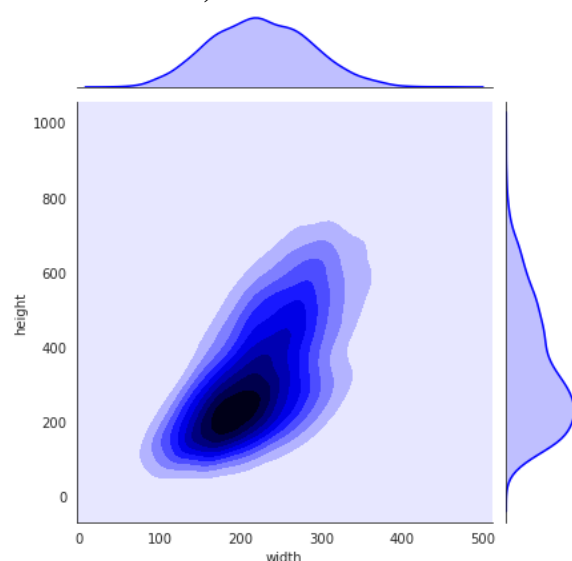
### 3.10.6 Understanding height and width of bounding box coordinates



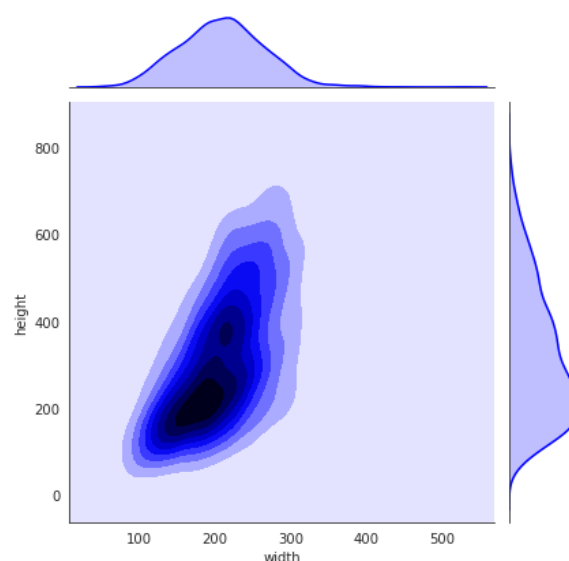
**Fig No. 3.10.7**

In the graph (Fig no. 3.10.7) The shape tells us the level of opacity. If the shape is rectangle, we assume that the opacity will be more, whereas if shape is square, we assume the opacity is less.

### 3.10.7 Understanding height and width of bounding box coordinates (males and females)



**Fig no. 3.10.8**



**Fig no. 3.10.9**

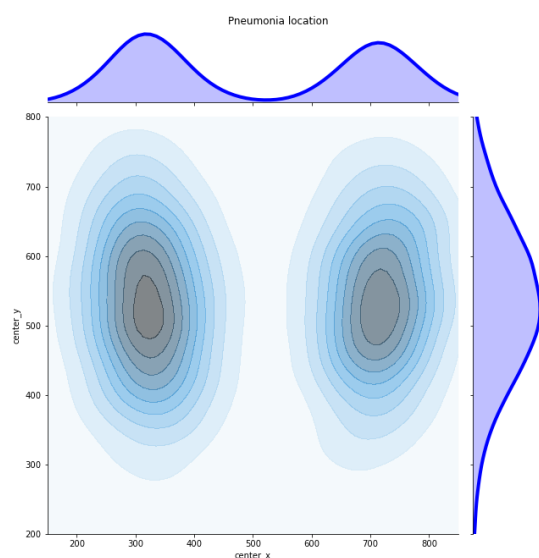
For Males in Fig no. 3.10.8, we see that most of the boxes are squarish, high density can be seen at around (200,200). There is only a single concentration in that area

For Females in fig 3.10.9, there are two cluster forming of concentration, highest concentration is at (200,200) and the smaller concentration is there at (400,210)



## 3.11 Understanding Pneumonia Location

### 3.11.1 Understanding pneumonia location for the overall dataset



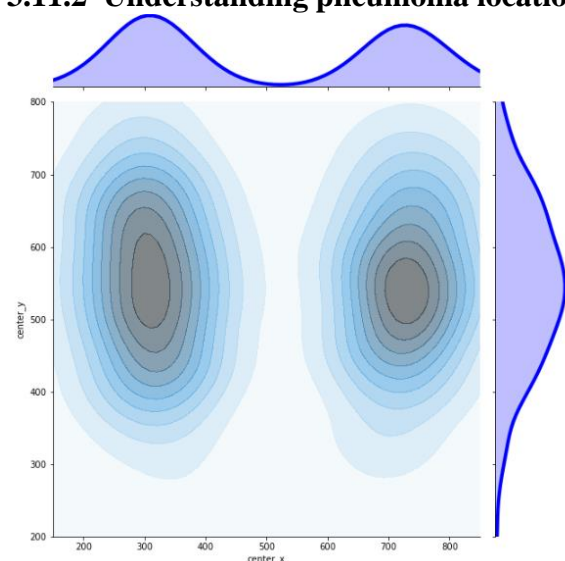
In Fig No. 3.11.1, We notice that higher chances of opacity to be identified is in between:

in left lung: (580,320) & (460,320)

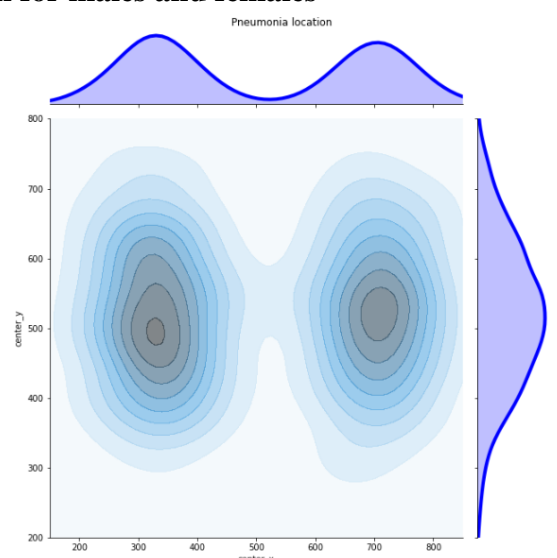
in right lung: (560,700) & (500,700)

**Fig No. 3.11.1**

### 3.11.2 Understanding pneumonia location for males and females



**Fig No. 3.11.2**



**Fig No. 3.11.3**

We notice for males In Fig 3.11.2, there are higher chances of opacity to be identified is in between:

in left lung: (600,320) & (460,320)

in right lung: (570,700) & (500,700)

We notice for females in Fig no. 3.11.3, there are higher chances of opacity to be identified is in between:

in left lung: (500,320) & (480,320)  
in right lung: (550,700) & (490,700)

### 3.12 Focus Age Group between 40 to 65

#### 3.12.1 Based on gender

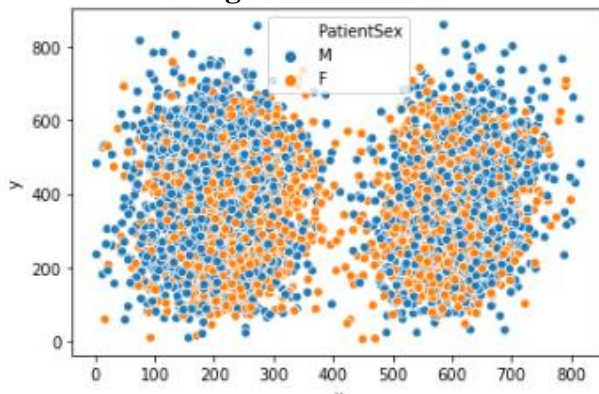
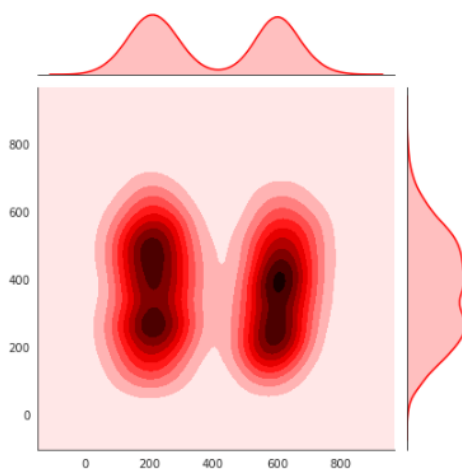


Fig no. 3.12.1 shows Females in the age group of 40 to 65 have higher chances of opacity in the left lung when compared to the right lung.

Fig no. 3.12.1

#### 3.12.2 Based on x and y coordinates



In Fig no. 3.12.2, for patients in age group between 45 to 65, the prominent density is shown in the right lung between coordinates (390,600) to (410,600). There are two clusters in the left lung: first is in between (220, 200) to (300, 200) and second cluster is in (380, 570)

Fig No. 3.12.2

### 3.12.3 Based on x and y coordinates for females and males

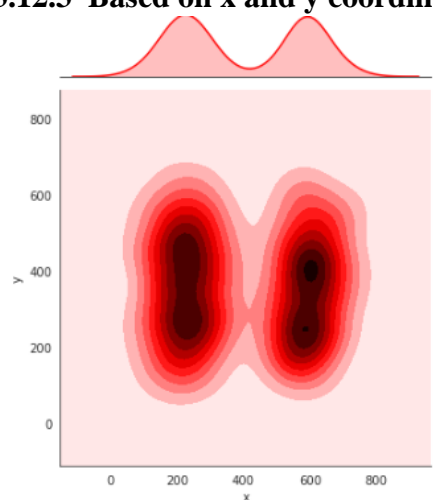


Fig No. 3.12.3

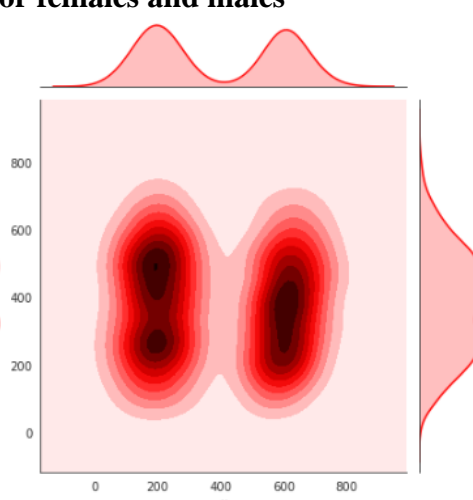


Fig No. 3.12.4

For female patients (See Fig No. 3.12.3) in age group between 45 to 65  
There are two cluster of prominent density in the right lung between, First Cluster: coordinates (210,600) to (220,600) & Second Cluster (380,600) to (440 ,600)

For male patients (See Fig No. 3.12.4) in age group between 45 to 65:  
There is only one prominent density cluster seen in left lung around (500,200) to (505,200)

### 3.12.4 Based on height and width of bounding box coordinates

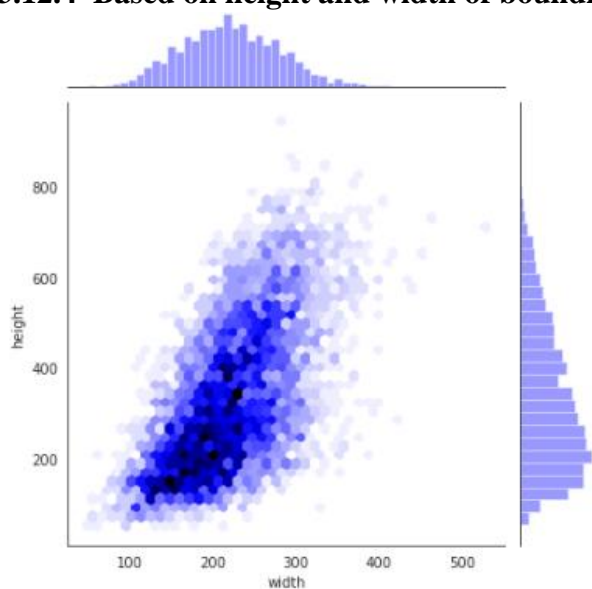


Fig No. 3.12.5

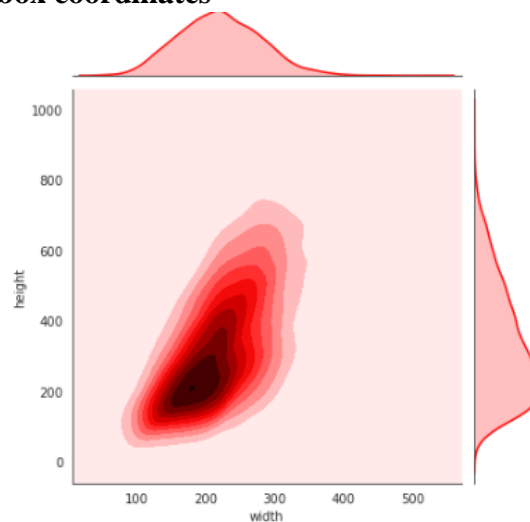


Fig No. 3.12.6

## 4. Approach

In this project, we have tried to blend the state of art models with different concepts of object detection and segmentation. During this process, we have trained models from scratch and also used the state of art models with pretrained weights. Although there are several pretrained models giving good results in the internet, our idea behind this approach is to implement the key learnings from the course and demonstrate it through the process.

Below is a flowchart showing the technique used and the models trained under each category.

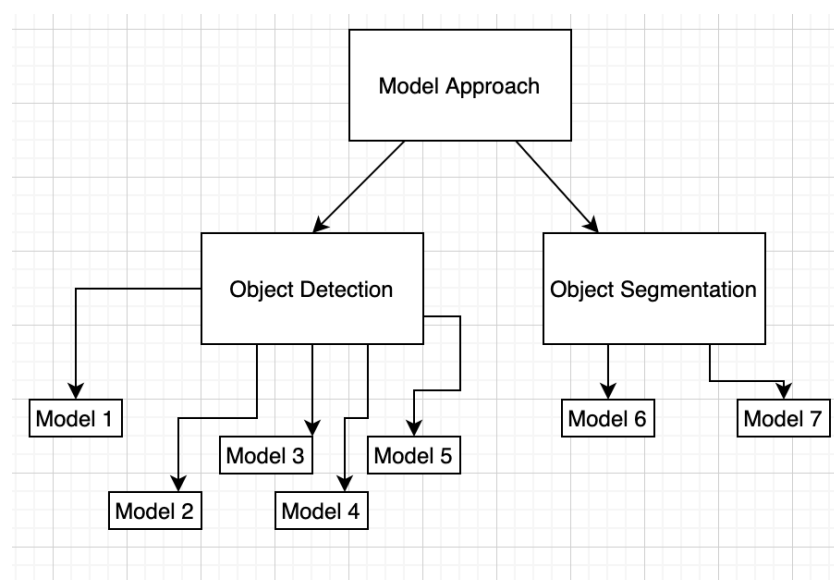


Fig No. 4.1

## 5. Models

In this section, we describe the proposed automatic detection of pneumonia in Chest X-ray images (CXRs) using Convolutional Neural Networks (CNN).

Section 5.1 explains the architecture of the seven different models developed during this process.

Section 5.2 covers the implementation process of the different model.

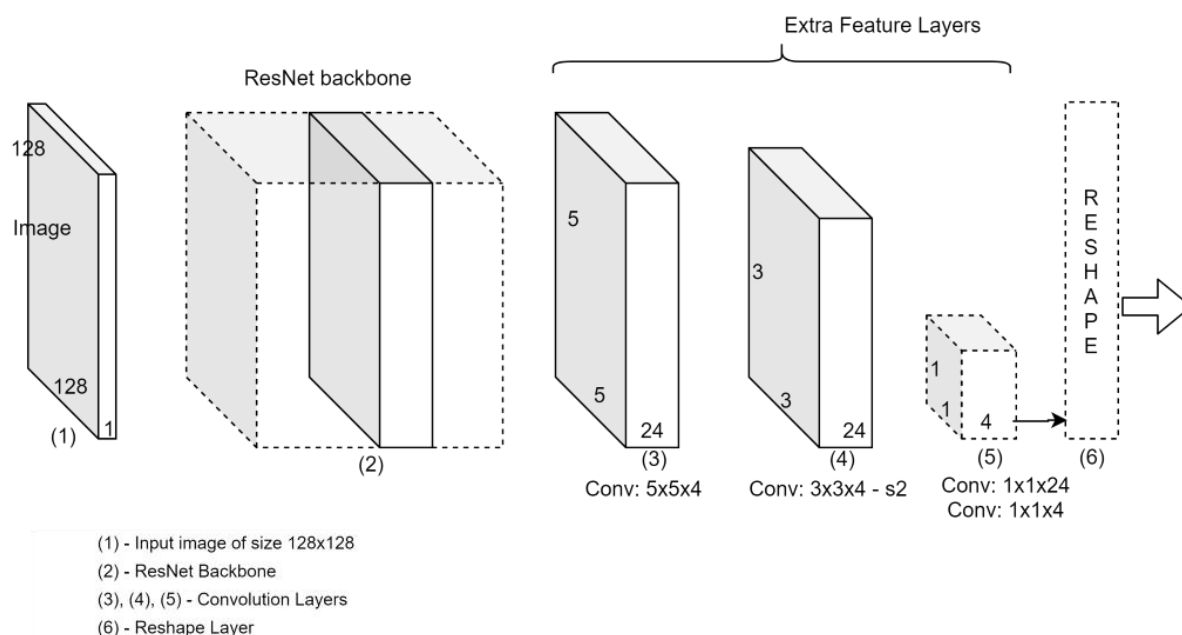
Section 5.3 shows the model performance using loss curves and mean iou curves to evaluate the mode.

Section 5.4 provides a comparison of the different models and highlights the two final models one from each category of object detection and object segmentation.

Section 5.5 evaluates the two final models in terms of the performance, advantages, disadvantages to build up to the final model.

## 5.1 Model Architecture

### 5.1.1 Model 1



**Fig No. 5.1.1**

In this model, we are using the ResNet state of art architecture model as the backbone followed by the convolution layers for predicting the coordinates of the bounding box. The additional convolutional layers form a pyramid-like structure. The advantage of the pyramid structure is to increase the receptive field which would help in detecting and localizing the objects. The common practice is to use a fully connected layer as the output layer, but we have used 1x1 convolution layer as an output layer. This is because the fully connected layer increases the trainable parameters whereas a 1x1 convolution layer helps to control the number of feature maps. Since 1x1 convolution layer can also be used as a fully connected layer, hence we are using it for prediction of four coordinates of the bounding box. Post the last 1x1 convolution layer, the output is of 1x1x4. For the purpose of reshaping the output to 2D, a Reshape layer is being used which will return the number of samples and their four coordinates.

## 5.1.2 Model 2

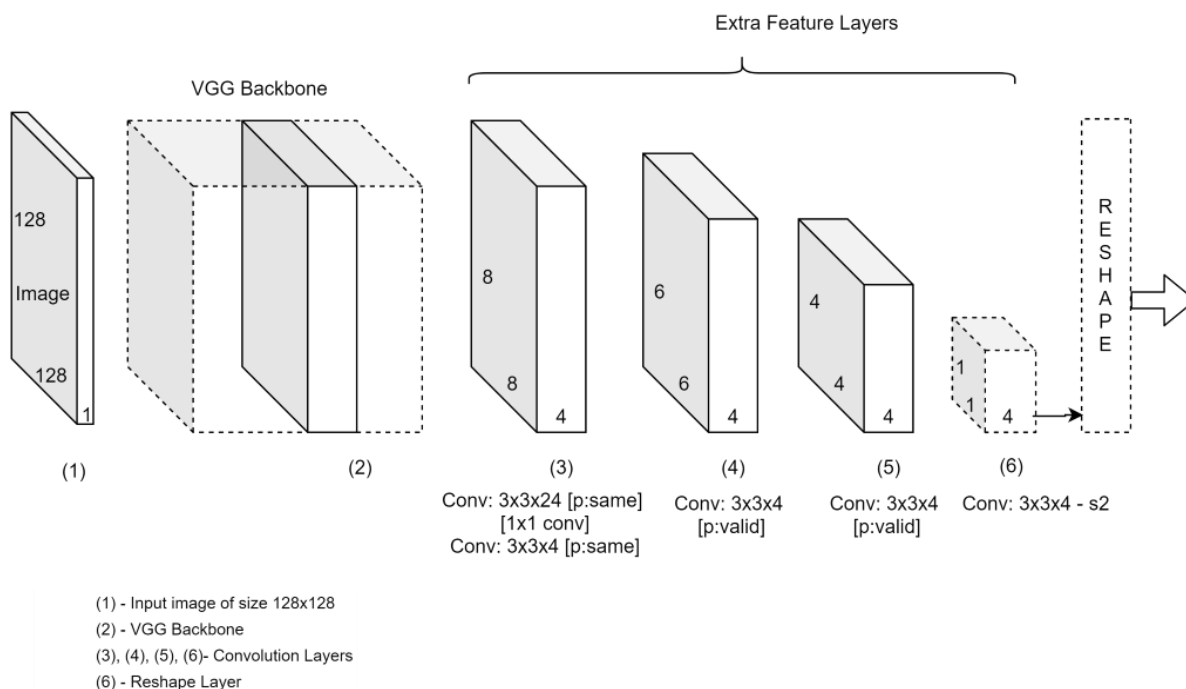


Fig No. 5.1.2

The architecture of this model is similar to Model 1 in Section 5.1.1, except that the backbone architecture used in this model is VGG. The backbone VGG model is followed by a bunch of convolutional layers followed by the final reshape layer which gives the samples along with the bounding box coordinates.

## 5.1.3 Model 3

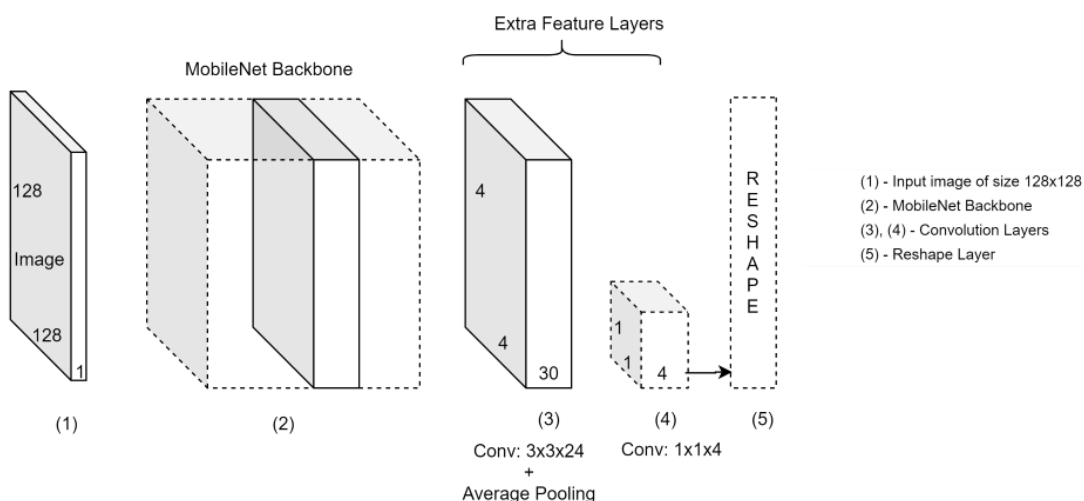
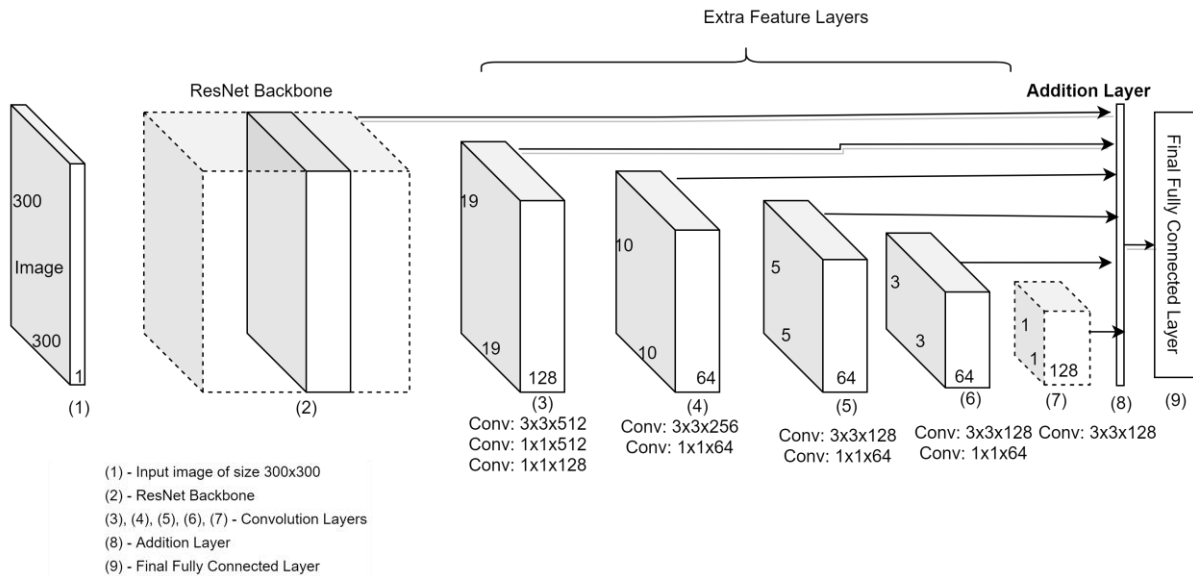


Fig No. 5.1.3

The architecture of this model is similar to Model 1 in Section 5.1.1, except that the backbone architecture used in this model is MobileNet.

### 5.1.4 Model 4



**Fig No. 5.1.4**

In this model, we have extended and updated of model used in section 5.1.1. We have blended it with the concept of multi-box single shot detection. We are using ResNet state of art architecture model as the backbone followed by the blocks of convolution layers consisting of 1x1 convolution layer and 3x3 convolution layer. Each block of convolution layer and the feature map of the ResNet state of art model are being branched out to Fully Connected layer (depicted by the arrows in the diagram). Further the respective fully connected layer is being added together in the Addition layer as shown in the diagram. The output of the Addition layer is fed into a fully connected layer which is the output layer predicting the four coordinates of the bounding box.

## 5.1.5 Model 5

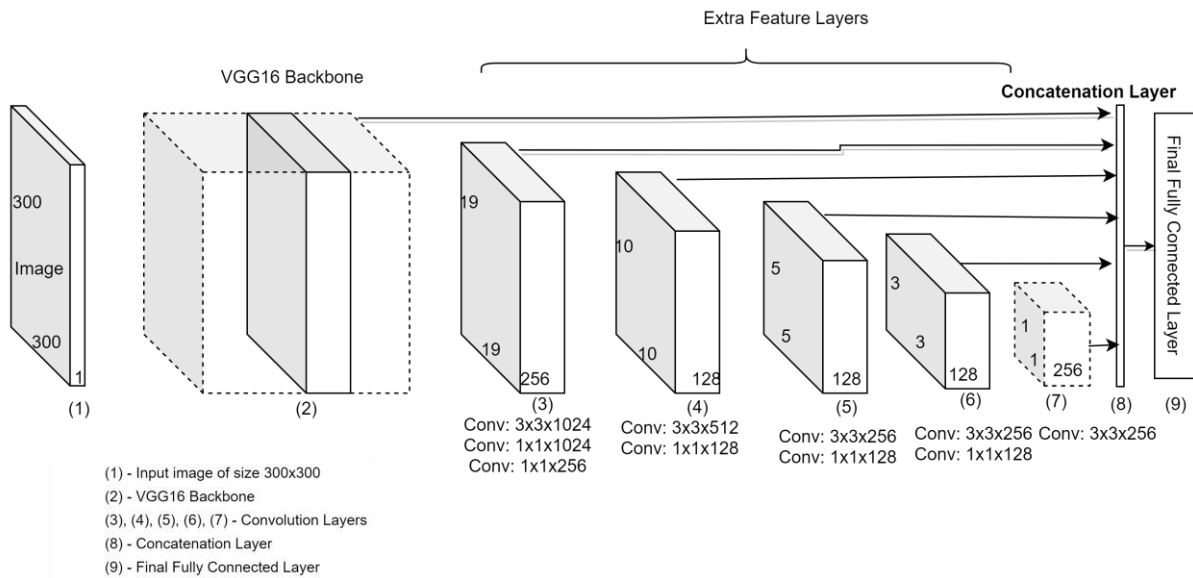


Fig No. 5.1.5

In this model, we have extended and updated of model used in section 5.1.2. We have blended it with the concept of multi-box single shot detection. The model is built on VGG16 using pretrained weights as the backbone followed by the blocks of convolution layers consisting of 1x1 convolution layer and 3x3 convolution layer. Each block of convolution layer and the feature map of the VGG16 model are being branched out to Fully Connected layer (depicted by the arrows in the diagram). Further the respective fully connected layer is being added together in the Concatenation layer as shown in the diagram. The output of the Concatenation layer is fed into a fully connected layer which is the output layer predicting the four coordinates of the bounding box.

## 5.1.6 Model 6

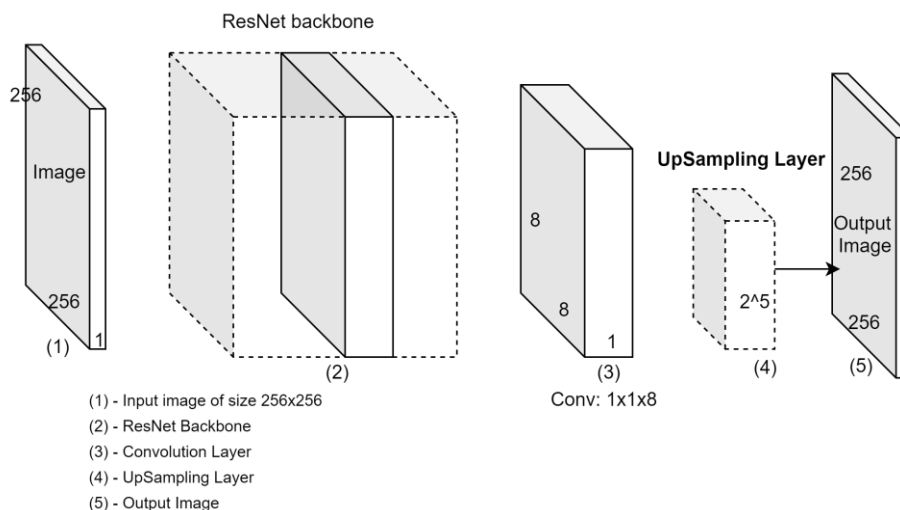


Fig No. 5.1.6

This architecture is a blend of object detection and segmentation. Here, we have used the ResNet state of art architecture as the backbone for detection. The primary objective behind this architecture is to show the pixel wise classification of the opacity in the X-Ray image extracted from the .dicom file.



On the feature map from the backbone model we do downsampling. Downsampling keeps the important features intact without losing too much information from the feature map. The pixel wise classification ends at the  $1 \times 1$  convolution layer. This is followed by upsampling layer, which increases the feature size to its original image size. This adds back the information that was lost during the downsampling process. The reason behind using only one upsampling layer is to keep the number of trainable parameters less. This will help save the computational time.

### 5.1.7 Model 7

The architecture of this model is similar to Model 6 in Section 5.1.6, except that the backbone architecture used in this model is VGG.

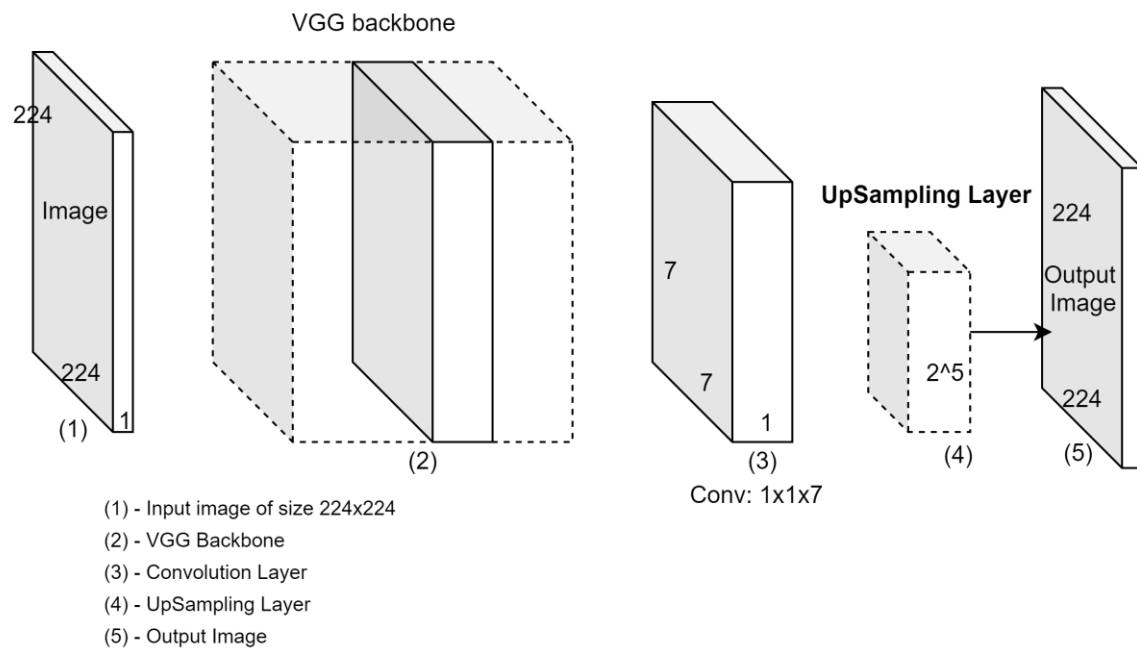


Fig No. 5.1.7

## 5.2 Model Summary

### 5.2.1 Model 1

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 128, 128, 1)]	0	
conv2d (Conv2D)	(None, 128, 128, 32) 288		input_3[0][0]
batch_normalization (BatchNormal	(None, 128, 128, 32) 128		conv2d[0][0]
leaky_re_lu (LeakyReLU)	(None, 128, 128, 32) 0		batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 128, 128, 64) 2848		leaky_re_lu[0][0]
max_pooling2d (MaxPooling2D)	(None, 42, 42, 64) 0		conv2d_1[0][0]
batch_normalization_1 (BatchNor	(None, 42, 42, 64) 256		max_pooling2d[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 42, 42, 64) 0		batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 42, 42, 64) 36864		leaky_re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 42, 42, 64) 256		conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 42, 42, 64) 0		batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 42, 42, 64) 36864		leaky_re_lu_2[0][0]
add (Add)	(None, 42, 42, 64) 0		conv2d_3[0] conv2d_2[0]
batch_normalization_3 (BatchNor	(None, 42, 42, 64) 256		add[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 42, 42, 64) 0		batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 42, 42, 64) 36864		leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 42, 42, 64) 256		conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 42, 42, 64) 0		batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 42, 42, 64) 36864		leaky_re_lu_4[0][0]
add_1 (Add)	(None, 42, 42, 64) 0		conv2d_5[0] conv2d_4[0]
batch_normalization_5 (BatchNor	(None, 42, 42, 64) 256		add_1[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 42, 42, 64) 0		batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 42, 42, 128) 8192		leaky_re_lu_5[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 128) 0		conv2d_6[0][0]
batch_normalization_6 (BatchNor	(None, 14, 14, 128) 512		max_pooling2d_1[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 14, 14, 128) 0		batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 14, 14, 128) 147456		leaky_re_lu_6[0][0]
batch_normalization_7 (BatchNor	(None, 14, 14, 128) 512		conv2d_7[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 128) 0		batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 14, 14, 128) 147456		leaky_re_lu_7[0][0]
add_2 (Add)	(None, 14, 14, 128) 0		conv2d_8[0] conv2d_7[0]
batch_normalization_8 (BatchNor	(None, 14, 14, 128) 512		add_2[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 14, 14, 128) 0		batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 14, 14, 128) 147456		leaky_re_lu_8[0][0]
batch_normalization_9 (BatchNor	(None, 14, 14, 128) 512		conv2d_9[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 14, 14, 128) 0		batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 14, 14, 128) 147456		leaky_re_lu_9[0][0]
add_3 (Add)	(None, 14, 14, 128) 0		conv2d_10[0] conv2d_9[0]
batch_normalization_10 (BatchNor	(None, 14, 14, 128) 512		add_3[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 14, 14, 128) 0		batch_normalization_10[0][0]
conv2d_11 (Conv2D)	(None, 14, 14, 128) 147456		leaky_re_lu_10[0][0]
conv2d_12 (Conv2D)	(None, 6, 6, 64) 6976		conv2d_11[0][0]
conv2d_13 (Conv2D)	(None, 4, 4, 128) 6924		conv2d_12[0][0]
conv2d_14 (Conv2D)	(None, 1, 1, 32) 3488		conv2d_13[0][0]
conv2d_15 (Conv2D)	(None, 1, 1, 4) 132		conv2d_14[0][0]
reshape (Reshape)	(None, 4)		conv2d_15[0][0]

Total params: 778,864  
 Trainable params: 768,868  
 Non-trainable params: 1,996

### 5.2.2 Model 2

Model: "model\_2"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 224, 224, 1)]	0
conv2d_32 (Conv2D)	(None, 224, 224, 12)	120
conv2d_33 (Conv2D)	(None, 224, 224, 12)	1308
max_pooling2d_7 (MaxPooling2	(None, 112, 112, 12)	0
conv2d_34 (Conv2D)	(None, 112, 112, 18)	1962
conv2d_35 (Conv2D)	(None, 112, 112, 18)	2934
max_pooling2d_8 (MaxPooling2	(None, 56, 56, 18)	0
conv2d_36 (Conv2D)	(None, 56, 56, 24)	3912
conv2d_37 (Conv2D)	(None, 56, 56, 24)	5208
max_pooling2d_9 (MaxPooling2	(None, 28, 28, 24)	0
conv2d_38 (Conv2D)	(None, 28, 28, 30)	6510
conv2d_39 (Conv2D)	(None, 28, 28, 30)	8130
conv2d_40 (Conv2D)	(None, 28, 28, 30)	8130
max_pooling2d_10 (MaxPooling	(None, 14, 14, 30)	0
conv2d_41 (Conv2D)	(None, 14, 14, 24)	6504
conv2d_42 (Conv2D)	(None, 14, 14, 24)	5208
conv2d_43 (Conv2D)	(None, 14, 14, 24)	5208
max_pooling2d_11 (MaxPooling	(None, 7, 7, 24)	0
conv2d_44 (Conv2D)	(None, 3, 3, 30)	6510
conv2d_45 (Conv2D)	(None, 3, 3, 30)	8130
conv2d_46 (Conv2D)	(None, 1, 1, 30)	8130
conv2d_47 (Conv2D)	(None, 1, 1, 4)	124
reshape_2 (Reshape)	(None, 4)	0

Total params: 78,028  
 Trainable params: 78,028  
 Non-trainable params: 0

### 5.2.3 Model 3

Model: "model\_L4"

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 128, 128, 1)]	0
conv2d_69 (Conv2D)	(None, 64, 64, 6)	68
batch_normalization_41 (Batch	(None, 64, 64, 6)	24
activation_31 (Activation)	(None, 64, 64, 6)	0
deephwise_conv2d_18 (DepthW	(None, 64, 64, 6)	68
batch_normalization_42 (Batch	(None, 64, 64, 6)	24
activation_32 (Activation)	(None, 64, 64, 6)	0
conv2d_70 (Conv2D)	(None, 64, 64, 12)	84
batch_normalization_44 (Batch	(None, 64, 64, 12)	48
activation_34 (Activation)	(None, 64, 64, 12)	0
deephwise_conv2d_19 (DepthW	(None, 32, 32, 12)	156
batch_normalization_45 (Batch	(None, 32, 32, 12)	48
activation_35 (Activation)	(None, 32, 32, 12)	0
conv2d_72 (Conv2D)	(None, 32, 32, 12)	156
batch_normalization_46 (Batch	(None, 32, 32, 12)	48
activation_36 (Activation)	(None, 32, 32, 12)	0
conv2d_73 (Conv2D)	(None, 32, 32, 18)	234
batch_normalization_47 (Batch	(None, 32, 32, 18)	72
activation_37 (Activation)	(None, 32, 32, 18)	0
deephwise_conv2d_20 (DepthW	(None, 32, 32, 18)	180
batch_normalization_48 (Batch	(None, 32, 32, 18)	72
activation_38 (Activation)	(None, 32, 32, 18)	0
conv2d_74 (Conv2D)	(None, 32, 32, 18)	342
batch_normalization_49 (Batch	(None, 32, 32, 18)	72
activation_39 (Activation)	(None, 32, 32, 18)	0
conv2d_75 (Conv2D)	(None, 32, 32, 18)	342
batch_normalization_50 (Batch	(None, 32, 32, 18)	72
activation_40 (Activation)	(None, 32, 32, 18)	0
deephwise_conv2d_21 (DepthW	(None, 16, 16, 18)	108
batch_normalization_51 (Batch	(None, 16, 16, 18)	72
activation_41 (Activation)	(None, 16, 16, 18)	0
conv2d_76 (Conv2D)	(None, 16, 16, 18)	342
batch_normalization_52 (Batch	(None, 16, 16, 18)	72
activation_42 (Activation)	(None, 16, 16, 18)	0
conv2d_77 (Conv2D)	(None, 16, 16, 24)	456
batch_normalization_53 (Batch	(None, 16, 16, 24)	96
activation_43 (Activation)	(None, 16, 16, 24)	0
deephwise_conv2d_22 (DepthW	(None, 16, 16, 24)	144
batch_normalization_54 (Batch	(None, 16, 16, 24)	96
activation_44 (Activation)	(None, 16, 16, 24)	0
conv2d_78 (Conv2D)	(None, 16, 16, 24)	680
batch_normalization_55 (Batch	(None, 16, 16, 24)	96
activation_45 (Activation)	(None, 16, 16, 24)	0
conv2d_79 (Conv2D)	(None, 16, 16, 24)	680
batch_normalization_56 (Batch	(None, 16, 16, 24)	96
activation_46 (Activation)	(None, 16, 16, 24)	0
deephwise_conv2d_23 (DepthW	(None, 8, 8, 24)	72
batch_normalization_57 (Batch	(None, 8, 8, 24)	96
activation_47 (Activation)	(None, 8, 8, 24)	0
conv2d_80 (Conv2D)	(None, 8, 8, 24)	680
batch_normalization_58 (Batch	(None, 8, 8, 24)	96
activation_48 (Activation)	(None, 8, 8, 24)	0
conv2d_81 (Conv2D)	(None, 8, 8, 30)	750
batch_normalization_59 (Batch	(None, 8, 8, 30)	120
activation_49 (Activation)	(None, 8, 8, 30)	0
deephwise_conv2d_24 (DepthW	(None, 8, 8, 30)	360
batch_normalization_60 (Batch	(None, 8, 8, 30)	120
activation_50 (Activation)	(None, 8, 8, 30)	0
conv2d_82 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_61 (Batch	(None, 8, 8, 30)	120
activation_51 (Activation)	(None, 8, 8, 30)	0
conv2d_83 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_62 (Batch	(None, 8, 8, 30)	120
activation_52 (Activation)	(None, 8, 8, 30)	0
deephwise_conv2d_25 (DepthW	(None, 8, 8, 30)	360
batch_normalization_63 (Batch	(None, 8, 8, 30)	120
activation_53 (Activation)	(None, 8, 8, 30)	0
conv2d_84 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_64 (Batch	(None, 8, 8, 30)	120
activation_54 (Activation)	(None, 8, 8, 30)	0
conv2d_85 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_65 (Batch	(None, 8, 8, 30)	120
activation_55 (Activation)	(None, 8, 8, 30)	0
deephwise_conv2d_26 (DepthW	(None, 8, 8, 30)	360
batch_normalization_66 (Batch	(None, 8, 8, 30)	120
activation_56 (Activation)	(None, 8, 8, 30)	0
conv2d_86 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_67 (Batch	(None, 8, 8, 30)	120
activation_57 (Activation)	(None, 8, 8, 30)	0
conv2d_87 (Conv2D)	(None, 8, 8, 30)	930
batch_normalization_68 (Batch	(None, 8, 8, 30)	120
activation_58 (Activation)	(None, 8, 8, 30)	0
deephwise_conv2d_27 (DepthW	(None, 4, 4, 30)	360
batch_normalization_69 (Batch	(None, 4, 4, 30)	120
activation_59 (Activation)	(None, 4, 4, 30)	0
conv2d_88 (Conv2D)	(None, 4, 4, 30)	930
batch_normalization_70 (Batch	(None, 4, 4, 30)	120
activation_60 (Activation)	(None, 4, 4, 30)	0
conv2d_89 (Conv2D)	(None, 4, 4, 30)	930
batch_normalization_71 (Batch	(None, 4, 4, 30)	120
activation_61 (Activation)	(None, 4, 4, 30)	0
average_pooling2d_1 (Average	(None, 1, 1, 30)	0
conv2d_90 (Conv2D)	(None, 1, 1, 4)	124
reshape_4 (Reshape)	(None, 4)	0

Total params: 16,126  
 Trainable params: 14,794  
 Non-trainable params: 1,332

## 5.2.4 Model 4

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 300, 300, 1) 0]		
conv2d (Conv2D)	(None, 300, 300, 32) 288		input_1[0][0]
batch_normalization (BatchNormaliza	(None, 300, 300, 32) 128		conv2d[0][0]
leaky_re_lu (LeakyReLU)	(None, 300, 300, 32) 0		batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 300, 300, 64) 2048		leaky_re_lu[0][0]
max_pooling2d (MaxPooling2D)	(None, 150, 150, 64) 0		conv2d_1[0][0]
batch_normalization_1 (BatchNor	(None, 150, 150, 64) 256		max_pooling2d[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 150, 150, 64) 0		batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 150, 150, 64) 36864		leaky_re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 150, 150, 64) 256		conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 150, 150, 64) 0		batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 150, 150, 64) 36864		leaky_re_lu_2[0][0]
add (Add)	(None, 150, 150, 64) 0		conv2d_3[0][0] max_pooling2d[0][0]
batch_normalization_3 (BatchNor	(None, 150, 150, 64) 256		add[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 150, 150, 64) 0		batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 150, 150, 64) 36864		leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 150, 150, 64) 256		conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 150, 150, 64) 0		batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 150, 150, 64) 36864		leaky_re_lu_4[0][0]
add_1 (Add)	(None, 150, 150, 64) 0		conv2d_5[0][0] add[0][0]
batch_normalization_5 (BatchNor	(None, 150, 150, 64) 256		add_1[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 150, 150, 64) 0		batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 150, 150, 128) 8192		leaky_re_lu_5[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 128) 0		conv2d_6[0][0]
batch_normalization_6 (BatchNor	(None, 75, 75, 128) 512		max_pooling2d_1[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 75, 75, 128) 0		batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 75, 75, 128) 147456		leaky_re_lu_6[0][0]
batch_normalization_7 (BatchNor	(None, 75, 75, 128) 512		conv2d_7[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 75, 75, 128) 0		batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 75, 75, 128) 147456		leaky_re_lu_7[0][0]
add_2 (Add)	(None, 75, 75, 128) 0		conv2d_8[0][0] max_pooling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 75, 75, 128) 512		add_2[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 75, 75, 128) 0		batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 75, 75, 128) 147456		leaky_re_lu_8[0][0]
batch_normalization_9 (BatchNor	(None, 75, 75, 128) 512		conv2d_9[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 75, 75, 128) 0		batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 75, 75, 128) 147456		leaky_re_lu_9[0][0]
add_3 (Add)	(None, 75, 75, 128) 0		conv2d_10[0][0] add_2[0][0]

batch_normalization_10 (BatchNo	(None, 75, 75, 128) 512		add_3[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 75, 75, 128) 0		batch_normalization_10[0][0]
conv2d_11 (Conv2D)	(None, 75, 75, 256) 32768		leaky_re_lu_10[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 37, 37, 256) 0		conv2d_11[0][0]
batch_normalization_11 (BatchNo	(None, 37, 37, 256) 1024		max_pooling2d_2[0][0]
leaky_re_lu_11 (LeakyReLU)	(None, 37, 37, 256) 0		batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 37, 37, 256) 589824		leaky_re_lu_11[0][0]
batch_normalization_12 (BatchNo	(None, 37, 37, 256) 1024		conv2d_12[0][0]
leaky_re_lu_12 (LeakyReLU)	(None, 37, 37, 256) 0		batch_normalization_12[0][0]
conv2d_13 (Conv2D)	(None, 37, 37, 256) 589824		leaky_re_lu_12[0][0]
add_4 (Add)	(None, 37, 37, 256) 0		conv2d_13[0][0] max_pooling2d_2[0][0]
batch_normalization_13 (BatchNo	(None, 37, 37, 256) 1024		add_4[0][0]
leaky_re_lu_13 (LeakyReLU)	(None, 37, 37, 256) 0		batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 37, 37, 256) 589824		leaky_re_lu_13[0][0]
batch_normalization_14 (BatchNo	(None, 37, 37, 256) 1024		conv2d_14[0][0]
leaky_re_lu_14 (LeakyReLU)	(None, 37, 37, 256) 0		batch_normalization_14[0][0]
conv2d_15 (Conv2D)	(None, 37, 37, 256) 589824		leaky_re_lu_14[0][0]
add_5 (Add)	(None, 37, 37, 256) 0		conv2d_15[0][0] add_4[0][0]
conv2d_16 (Conv2D)	(None, 19, 19, 512) 1180160		add_5[0][0]
conv2d_17 (Conv2D)	(None, 19, 19, 512) 262656		conv2d_16[0][0]
conv2d_18 (Conv2D)	(None, 19, 19, 128) 65664		conv2d_17[0][0]
convsame (Conv2D)	(None, 10, 10, 256) 295168		conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 10, 10, 64) 16448		convsame[0][0]
convsame1 (Conv2D)	(None, 5, 5, 128) 73856		conv2d_19[0][0]
conv2d_20 (Conv2D)	(None, 5, 5, 64) 8256		convsame1[0][0]
convsame2 (Conv2D)	(None, 3, 3, 128) 73856		conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, 3, 3, 64) 8256		convsame2[0][0]
convsame3 (Conv2D)	(None, 1, 1, 128) 73856		conv2d_21[0][0]
flatten (Flatten)	(None, 350464) 0		add_5[0][0]
flatten_1 (Flatten)	(None, 184832) 0		conv2d_17[0][0]
flatten_2 (Flatten)	(None, 25600) 0		convsame[0][0]
flatten_3 (Flatten)	(None, 3200) 0		convsame1[0][0]
flatten_4 (Flatten)	(None, 1152) 0		convsame2[0][0]
flatten_5 (Flatten)	(None, 128) 0		convsame3[0][0]
dense (Dense)	(None, 4) 1401860		flatten[0][0]
dense_1 (Dense)	(None, 4) 739332		flatten_1[0][0]
dense_2 (Dense)	(None, 4) 102404		flatten_2[0][0]
dense_3 (Dense)	(None, 4) 12804		flatten_3[0][0]
dense_4 (Dense)	(None, 4) 4612		flatten_4[0][0]
dense_5 (Dense)	(None, 4) 516		flatten_5[0][0]
add_6 (Add)	(None, 4) 0		dense[0][0] dense_1[0][0] dense_2[0][0] dense_3[0][0] dense_4[0][0] dense_5[0][0]
dense_6 (Dense)	(None, 4) 20		add_6[0][0]

Total params: 7,467,660  
Trainable params: 7,463,628  
Non-trainable params: 4,032

## 5.2.5 Model 5

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 300, 300, 3)]	0	
block1_conv1 (Conv2D)	(None, 300, 300, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(None, 300, 300, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 150, 150, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(None, 150, 150, 128)	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 150, 150, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 75, 75, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(None, 75, 75, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 75, 75, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(None, 75, 75, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 37, 37, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(None, 37, 37, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 37, 37, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(None, 37, 37, 512)	2359808	block4_conv2[0][0]
conv2d (Conv2D)	(None, 19, 19, 1024)	4719616	block4_conv3[0][0]
conv2d_1 (Conv2D)	(None, 19, 19, 1024)	1049600	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 19, 19, 256)	262400	conv2d_1[0][0]
convsame (Conv2D)	(None, 10, 10, 512)	1180160	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None, 10, 10, 128)	65664	convsame[0][0]
convsame1 (Conv2D)	(None, 5, 5, 256)	295168	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 5, 5, 128)	32896	convsame1[0][0]
convsame2 (Conv2D)	(None, 3, 3, 256)	295168	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 3, 3, 128)	32896	convsame2[0][0]
convsame3 (Conv2D)	(None, 1, 1, 256)	295168	conv2d_5[0][0]
flatten (Flatten)	(None, 700928)	0	block4_conv3[0][0]
flatten_1 (Flatten)	(None, 369664)	0	conv2d_1[0][0]
flatten_2 (Flatten)	(None, 51200)	0	convsame[0][0]
flatten_3 (Flatten)	(None, 6400)	0	convsame1[0][0]
flatten_4 (Flatten)	(None, 2304)	0	convsame2[0][0]
flatten_5 (Flatten)	(None, 256)	0	convsame3[0][0]
dense (Dense)	(None, 4)	2803716	flatten[0][0]
dense_1 (Dense)	(None, 4)	1478660	flatten_1[0][0]
dense_2 (Dense)	(None, 4)	204804	flatten_2[0][0]
dense_3 (Dense)	(None, 4)	25604	flatten_3[0][0]
dense_4 (Dense)	(None, 4)	9220	flatten_4[0][0]
dense_5 (Dense)	(None, 4)	1028	flatten_5[0][0]
concatenate (Concatenate)	(None, 24)	0	dense[0][0] dense_1[0][0] dense_2[0][0] dense_3[0][0] dense_4[0][0] dense_5[0][0]
dense_6 (Dense)	(None, 4)	100	concatenate[0][0]

Total params: 20,387,132  
Trainable params: 12,751,868  
Non-trainable params: 7,635,264

## 5.2.6 Model 6

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 1)]	0	
conv2d (Conv2D)	(None, 256, 256, 8)	72	input_1[0][0]
batch_normalization (BatchNormaliza	(None, 256, 256, 8)	32	conv2d[0][0]
leaky_re_lu (LeakyReLU)	(None, 256, 256, 8)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	128	leaky_re_lu[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 16)	64	max_pooling2d[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 128, 128, 16)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 16)	2304	leaky_re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 128, 128, 16)	64	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 128, 128, 16)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 16)	2304	leaky_re_lu_2[0][0]
add (Add)	(None, 128, 128, 16)	0	conv2d_3[0][0] max_pooling2d[0][0]
batch_normalization_3 (BatchNor	(None, 128, 128, 16)	64	add[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 128, 128, 16)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 16)	2304	leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 128, 128, 16)	64	conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 128, 128, 16)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 128, 128, 16)	2304	leaky_re_lu_4[0][0]
add_1 (Add)	(None, 128, 128, 16)	0	conv2d_5[0][0] add[0][0]
batch_normalization_5 (BatchNor	(None, 128, 128, 16)	64	add_1[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 128, 128, 16)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 128, 128, 32)	512	leaky_re_lu_5[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_6[0][0]
batch_normalization_6 (BatchNor	(None, 64, 64, 32)	128	max_pooling2d_1[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 64, 64, 32)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 32)	9216	leaky_re_lu_6[0][0]
batch_normalization_7 (BatchNor	(None, 64, 64, 32)	128	conv2d_7[0][0]
leaky_re_lu_7 (LeakyReLU)	(None, 64, 64, 32)	0	batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 64, 64, 32)	9216	leaky_re_lu_7[0][0]
add_2 (Add)	(None, 64, 64, 32)	0	conv2d_8[0][0] max_pooling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 64, 64, 32)	128	add_2[0][0]
leaky_re_lu_8 (LeakyReLU)	(None, 64, 64, 32)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 64, 64, 32)	9216	leaky_re_lu_8[0][0]
batch_normalization_9 (BatchNor	(None, 64, 64, 32)	128	conv2d_9[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 64, 64, 32)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 64, 64, 32)	9216	leaky_re_lu_9[0][0]

## 5.2.7 Model 7

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 1)]	0
conv2d (Conv2D)	(None, 224, 224, 8)	80
conv2d_1 (Conv2D)	(None, 224, 224, 8)	584
max_pooling2d (MaxPooling2D)	(None, 112, 112, 8)	0
conv2d_2 (Conv2D)	(None, 112, 112, 16)	1168
conv2d_3 (Conv2D)	(None, 112, 112, 16)	2320
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 16)	0
conv2d_4 (Conv2D)	(None, 56, 56, 32)	4640
conv2d_5 (Conv2D)	(None, 56, 56, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 32)	0
conv2d_6 (Conv2D)	(None, 28, 28, 64)	18496
conv2d_7 (Conv2D)	(None, 28, 28, 64)	36928
conv2d_8 (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_9 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_10 (Conv2D)	(None, 14, 14, 128)	147584
conv2d_11 (Conv2D)	(None, 14, 14, 128)	147584
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 128)	0
batch_normalization (BatchNormalizatio	(None, 7, 7, 128)	512
leaky_re_lu (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_12 (Conv2D)	(None, 7, 7, 1)	129
up_sampling2d (UpSampling2D)	(None, 224, 224, 1)	0
=====		
Total params: 480,057		
Trainable params: 479,801		
Non-trainable params: 256		

### 5.3 Implementation Process

**Data Augmentation.** In our implementation, before feeding the images into the model, the original image size is resized from 1024 x 1024 pixels down to different sizes such as 512 x 512, 300 x 300, 256 x 256, 224 x 224, 128 x 128. We have applied normalization as follows:

Normalized bounding box coordinates by dividing by the image size

Normalized the images by using the preprocess\_input method.

This allows us to make the training on the hardware more efficient. In addition to normalization, we have also defined a user defined function for augmenting the images in the segmentation models using flipping method.

**Model Structure.** For all models, we have used one of the state of art model architectures of ResNet, VGG and MobileNet as the backbone over which we have used convolution layers for localization and detection of opacity in the X-rays. For one model, we have also trained using pretrained models of VGG16 over its 5th layer.

**Model Parameters.** We randomly split the dataset into training (75%) and validation (25%). The model was trained using Adam and Stochastic Gradient Descent (SGD) optimizers. We have used batch size of 32 with different epoch sizes. Initially a low learning rate of 0.01 was selected which was reduced after the successful completion of every epoch. In order to update the learning rate after every epoch, we have used two learning rate schedulers:

ReduceLROnPlateau

Cosine Annealing. It is a technique for learning rate scheduling and is used for:

Faster Training of Network

Finer Understanding of the Optimal learning rate

During training the learning rate is inversely scaled from its minimum to its maximum value and then back again. At the end of training, learning rate will be reduced even further below the minimum learning rate in order to squeeze out last bit of convergence. Hence after each cycle(epoch) the learning rate scheduler finds the new good values and then fit them to the new cycle.

Learning rate annealing is a technique where the learning rate is gradually reduced allowing the model to settle into the point of minimality in the cost surface. Hence cyclic learning rate scheduling combines discovery of the maximum practical learning rate with learning rate annealing. This reduces the learning rate in a non-linear way using cosine function.

**Model Selection.** We are using IOU as the model evaluation metrics. We are calculating IOU for each epoch and in the callbacks we are monitoring on the validation IOU. For each improvement of validation IOU, the model weights are saved. Mean IOU is the average of the IOU's of the sample images.

For segmentation, we are also using the accuracy metrics but it is not used that in our model comparison. We also have tried different threshold values (ranging from 0.3 to 0.7) for pixel wise classification for segmenting the opacity in the X-ray images



**Loss Function.** We have a user defined loss function defined over the concept of Jaccard Loss Function along with Binary Cross Entropy loss as follows:

Step 1: Calculate IOU loss based on Jaccard Loss Function

Step 2: Calculate Binary Cross Entropy loss

Step 3: Multiply (1) by a factor of 0.5.

Step 4: Multiply (2) by a factor of 0.5

Step 5: Add Step 3 + Step 4. This is our IOU Binary Cross Entropy Loss

**Data Generator:** We have 2 user defined functions:

Data Generator for Object Detection: returns a 1D Column array with 4 coordinates of bounding box

Data Generator for Object Segmentation: returns a mask which is of the same size as the resized image and the purpose is for pixel wise classification.

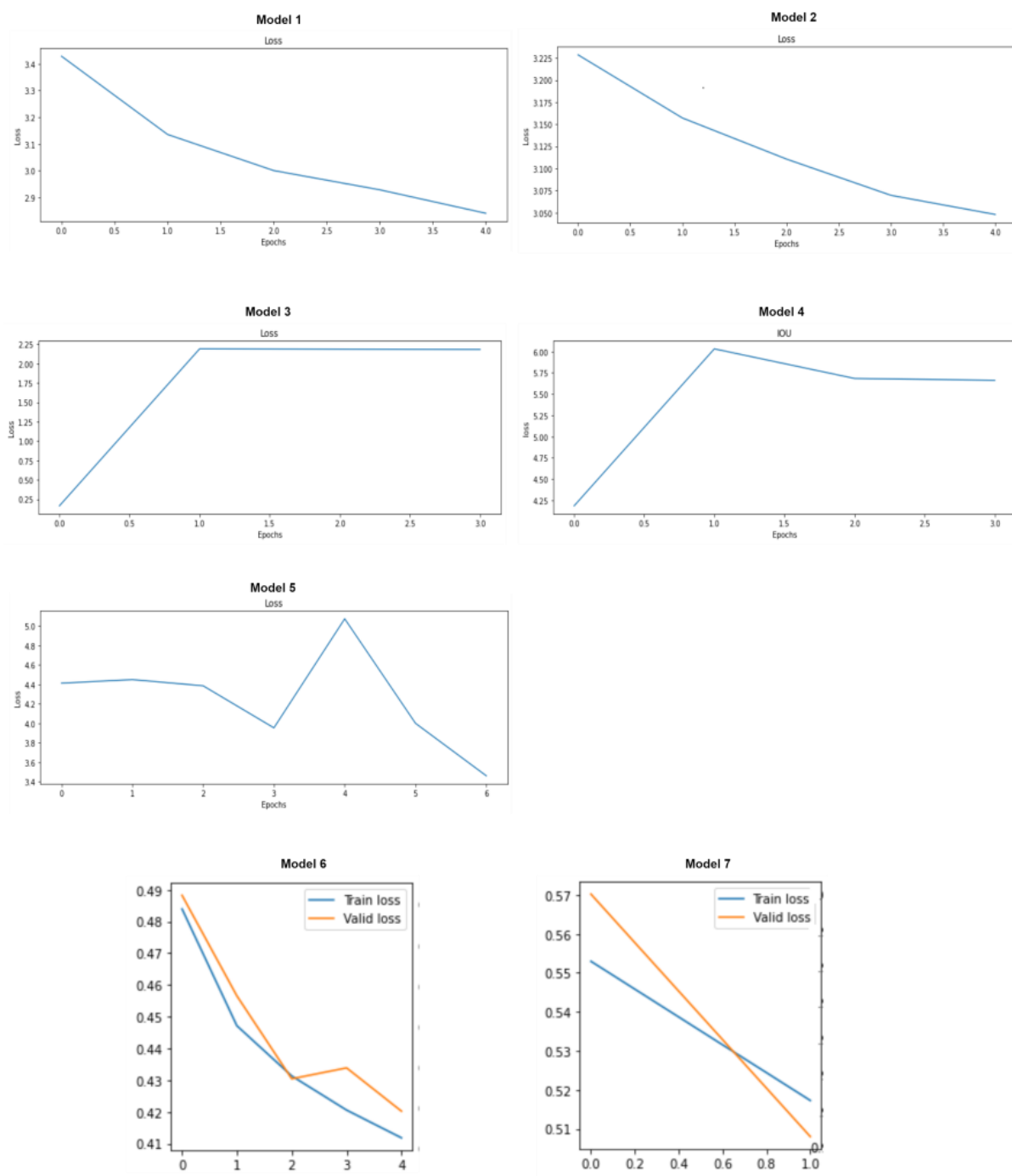
## 5.4 Model Comparison

Parameter	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Reference Architecture	ResNet	VGG	MobileNet	ResNet	VGG16 + TL	ResNet	VGG
Input Image Size	128x128	128x128	128x128	300x300	300x300	256x256	224x224
Total Parameters	7,70,844	12,982	16,126	7,467,660	20,387,132	3,196,585	480,057
Trainable Parameters	7,68,860	12,982	14,794	7,463,628	12,751,868	3,191,609	479,801
Non-Trainable Parameters	1,984	0	1,332	4,032	7,635,264	4,976	256
Epochs	5	5	5	4/5*	7/15*	5	5
IOU	0.2	0.185	2	0.6893	0.56	0.7383	0.6988
Val IOU	0.2	0.19	0.2	1	1	0.7324	0.7106
Loss	2.84	0.78	3.4	5.6622	3.45	0.4118	0.4403
Val Loss	-	-	-	-	-	0.4202	0.4447
Accuracy	-	-	-	-	-	0.9711	0.9692
Val Accuracy	-	-	-	-	-	0.9684	0.9703

\*this represents runs which stopped due to early stopping as there was no improvement in IOU

## 5.5 Model Performance

### 5.5.1 Loss Curves



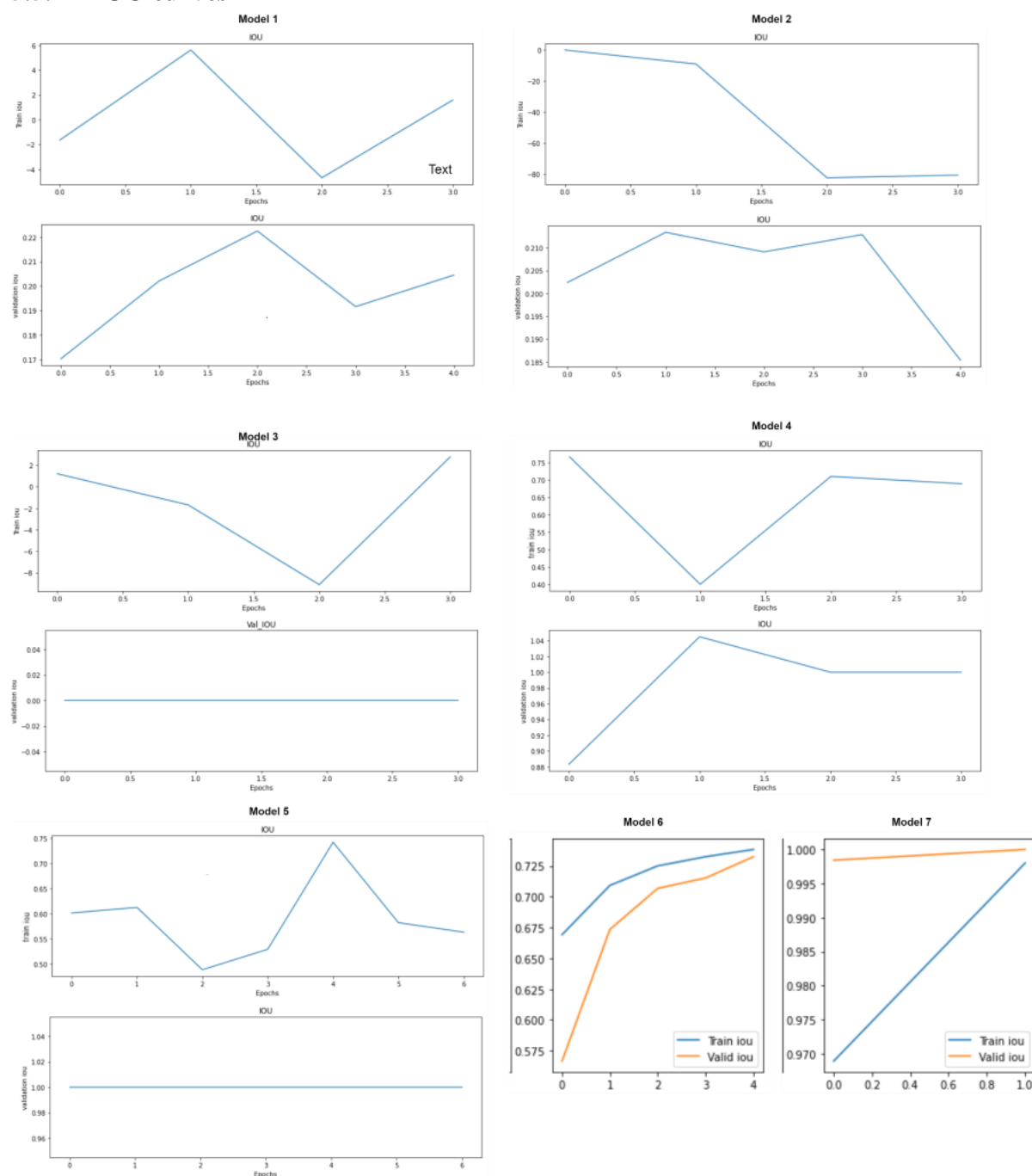
**Object Detection:** The base models built over ResNet, VGG and VGG16 with pretrained weights have shown decrease in losses as epoch run progresses. Additionally, VGG16 with pretrained weights showed a sudden spike in the loss at the 4<sup>th</sup> epoch which could be due to vanishing or exploding gradients. The rest of the models, loss gradually increases and remains stagnant, which is not good.



**Segmentation:** We see for both models trained on ResNet and VGG backbone, validation and training loss are decreasing as epoch run progresses. They are converging at 2<sup>nd</sup> and 3<sup>rd</sup> epoch respectively. This convergence is a good sign.

From these graphs, we can clearly see that the object detection models have not performed well for this study. We also feel that the image segmentation is a better model as it makes pixel wise classification which increases precision in locating and detecting opacity keeping in mind as the study is for the health care industry.

## 5.5.2 IOU curves



Of all the models, models 5, 6 and 7 have a good value of IOU. Although model 5 shows a value of validation IOU as 1, which shows that the model has underfitted. Therefore, model 6

and 7 are better comparatively. Had we ran models 6 and 7 for more than 5 epochs, the train and validation IOU might have converged.

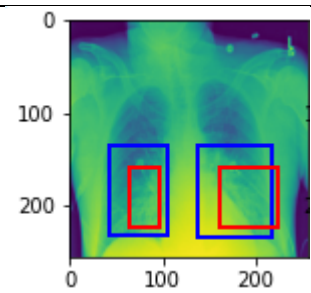
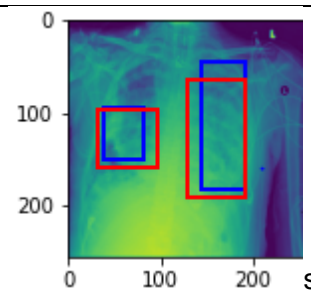
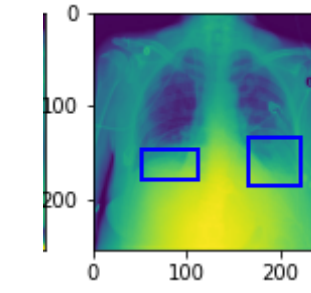
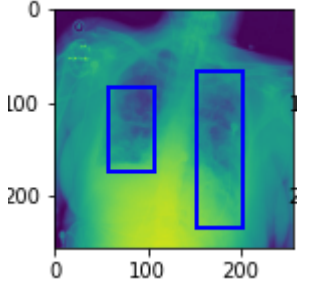
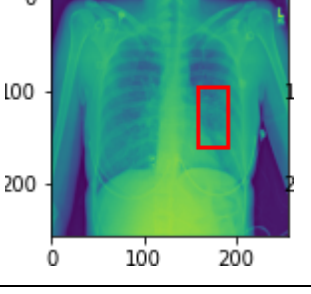
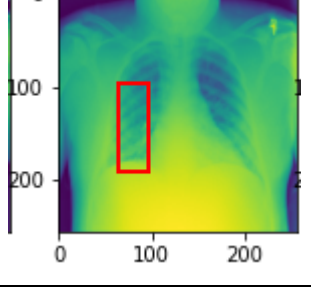
For the remaining models, we see that the training and validation IOU has either exceeded value of 1 or has become negative. This could be because of the following reasons:

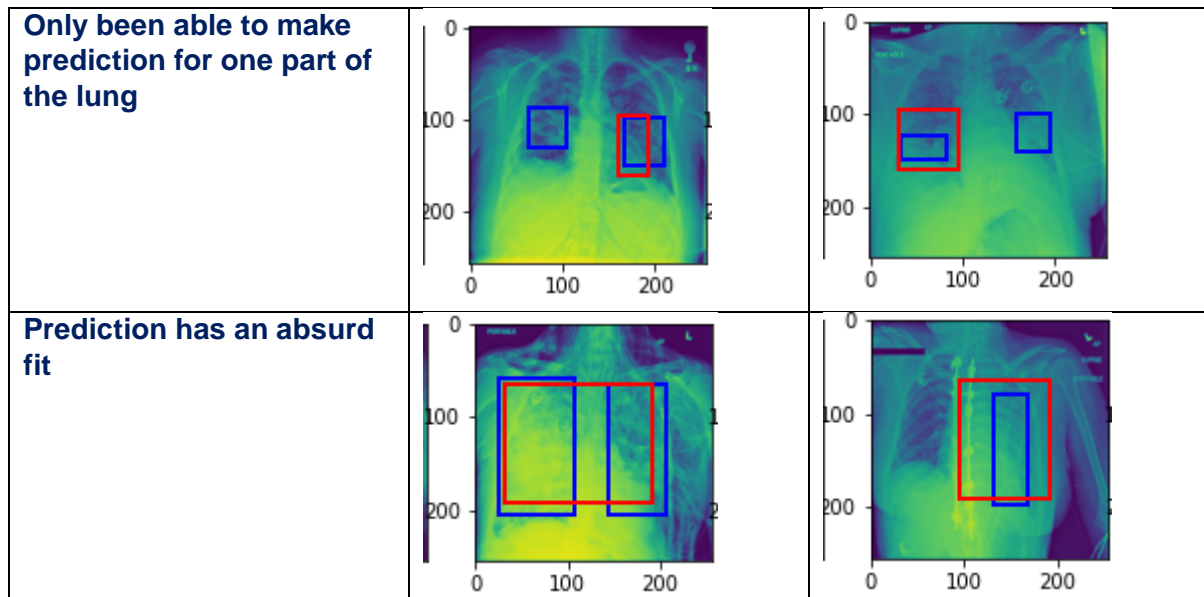
1. Huge difference between ground truth and the predicted
2. Size of predicted bounding box is greater than the ground truth
3. Vanishing and exploding gradients

## 5.6 Model Evaluation

Out of seven models, we have selected model 6 (ResNet Segmentation) and model 7 (VGG Segmentation).

The table below shows the comparative results seen during the study.

Evaluation Parameter	ResNet Segmentation	VGG Segmentation
Loss	0.4118	0.4403
Validation Loss	0.4202	0.4447
IOU	0.7383	0.6989
Val IOU	0.7324	0.7106
Best Fit		
No Prediction made		
Predicted but no ground truth present		



Despite the fact that the above two models have been trained on lesser number of parameters when compared to the other models, yet they have been able to fetch the important features and give training and validation IOU with lesser loss.

Out of the two segmentation models, we choose ResNet segmentation over VGG Segmentation for the below reasons:

1. Training and validation IOU are slightly better for ResNet segmentation when compared to VGG segmentation.
2. ResNet segmentation has low loss when compared to VGG Segmentation.

## 6. Conclusion

### 6.1 Work Summary

The dataset was uploaded into drive which was mounted on colab. We extracted the images and metadata from the dataset, EDA was performed on the data. We have defined a user defined function for the Image Data Generator which also preprocess the data. We defined the parameters for the model such as batch size, number of epochs, image size etc. We tried different model architectures for object detection and segmentation. Finally, we trained and validated the models.

### 6.2 Implications

- Models with pretrained weights are trained over RGB images. As the chest X-rays are grayscale images, we had a challenge of converting Gray scale to RGB.
- When trying to run ResNet model over pretrained weights for object detection, we had the problem of vanishing and exploding gradients
- The dataset was highly imbalanced which could have affected the mean\_IOU.

### 6.3 Limitations

- Google Colab GPU limit exceeded which interrupts the current run. The google account cannot be used for running the notebook using GPU for around 12 hours
- Whenever we are training models requiring long execution time, Google Colab session timeout after a period of 90 mins if there is no activity.

## 6.4 Closing Reflections

We have finalized the ResNet Segmentation for this study. This model gave a training and validation IOU of 0.73.

## 6.5 Future Enhancements

- Use Time Distributed Layer over the Dense layer which would act as a sliding window for object detection
- Use transformers, encoder and decoder architectures for object detection
- Explore Attention Mechanism (BERT) along with the above points
- Explore DERT
- YOLO v5

## 7. References

1. [https://medium.com/@jonathan\\_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06#:~:text=In%20SSD%2C%20small%20objects%20can,at%20the%20cost%20of%20speed.](https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06#:~:text=In%20SSD%2C%20small%20objects%20can,at%20the%20cost%20of%20speed.)
2. <https://developers.arcgis.com/python/guide/how-ssd-works/>
3. <https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>
4. <https://cv-tricks.com/object-detection/single-shot-multibox-detector-ssd/>
5. [https://www.researchgate.net/figure/Base-network-architecture-a-SSD-convolutional-layers-b-the-corresponding-conv\\_fig2\\_326764687](https://www.researchgate.net/figure/Base-network-architecture-a-SSD-convolutional-layers-b-the-corresponding-conv_fig2_326764687)